Overparametrization, architecture and dynamics of deep neural networks : from theory to practice

Stéphane d'Ascoli

PSL UNIVERSITÉ PARIS

THÈSE DE DOCTORAT
DE L'UNIVERSITÉ PSL

Préparée à l'École Normale Supérieure de Paris

Overparametrization, Architectures and Dynamics of Deep Neural Networks: from Theory to Practice

Soutenue par
Stéphane d'Ascoli
Le 24 juin 2022

École doctorale n°564
Ecole Doctorale "Physique en Ile-de-France"

Spécialité
Physique

Composition du jury :

Marc LELARGE INRIA Paris	*Président*
Kyle CRANMER New York University	*Rapporteur*
Kamalika CHAUDHURI University of California San Diego	*Rapporteur*
Andrew SAXE Oxford University	*Examinateur*
Lenka ZDEBOROVA École Polytechnique Fédérale de Lausanne	*Examinateur*
Léon BOTTOU Meta AI	*Examinateur*
Giulio BIROLI École Normale Supérieure	*Directeur de thèse*
Levent SAGUN Meta AI	*Co-encadrant de thèse*

ENS | PSL LPENS LABORATOIRE DE PHYSIQUE DE L'ÉCOLE NORMALE SUPÉRIEURE Meta

To be able to see everything in its slightest detail and remember every detail: what an amazing ability and what a torture. He could see aging occur by the second. His own hands one second are not the same in the next. [...]
But, indeed, here is the rub. With perfect perception and perfect memory, how does one actually comprehend? How can one create thought? Does not thought involve generalization and categorization and so forth? Does it not involve organizing? In short, does not thought necessitate a certain degree of forgetting?

Jorge Luis Borges, *Funes the Memorious*

Si tout se passe comme prévu, rien ne va se passer comme prévu.

Claude Marthaler, *Voyages sellestes*

Acknowledgements

For this dissertation For accepting to read and report my work, I warmly thank Kyle Cranmer and Kamalika Chadhuri. I also thank the rest of the defence jury: Marc Mézard, Lenka Zdeborova, Andrew Saxe and Léon Bottou.

Pour la thèse Un grand merci d'abord à mes encadrants, Giulio et Levent, pour m'avoir offert cette opportunité unique de concilier physique et intelligence artificielle, d'avoir toujours fait preuve d'un grand soutien, notamment à travers les débuts difficiles de la thèse, de m'avoir laissé la liberté de mener les projets que je souhaitais (y compris la rédaction de livres de vulgarisation), et de m'avoir tant apporté sur le plan scientifique. Merci aussi à François Charton pour m'avoir transmis sa passion des maths durant cette dernière année de thèse et m'avoir embarqué dans des projets aussi farfelus que fascinants.

J'ai eu la chance de pouvoir collaborer avec des personnes exceptionnelles tout au long de ma thèse: je remercie notamment Pierre-Alexandre, Marylou, Mario, Stefano, Hugo, Maria et Sebastian. Plus largement, je suis redevable à l'équipe du Sphinx pour m'avoir inclus au sein de leur grande famille: Florent, Lenka, Antoine, Benjamin, Stefano et les autres. Merci aussi à Antoine, Benjamin, Vassilis, Clément, Victor, et tous les autres compagnons de galère qui ont su égayer les journées de travail.

Merci enfin à Alice et Francesco pour m'avoir intégré au sein de leur sympathique équipe à Snips, et à Marc pour m'avoir permis d'enseigner au sein de l'ENS.

Pour tout le reste Je termine en remerciant ceux qui ont accompagné ma vie pendant ces années si intenses. Laëtitia, pour notre complicité indéfectible. Arthur et Adrien, pour cette colocation inoubliable, dont l'émulation aura abouti à une petite collection de livres, ainsi que Michaël, Vassilis, Robin, Simon, Juliette, Hortense pour ces belles années à l'ENS. Clémentine, Sandra, Dora, Juliette, pour avoir partagé ma passion pour la musique au COGE puis à Elektra. Maxine, pour son tendre soutien et toutes les belles choses que nous avons eu la chance de vivre ensemble. Ferdi, Théo et Robin, mes 3 *alter ego*, avec qui j'ai hâte d'entamer la plus belle aventure de ma vie. En enfin, tous les autres qu'ils me faudrait des pages pour énumérer.

Je termine par un hommage à mes parents, ces personnes exceptionnelles pour qui ma reconnaissance n'a d'égal que mon admiration, et le reste de ma famille, notamment mes deux grands-pères, tous deux hommes de sciences, qui ont su m'insuffler chacun à leur manière le goût du savoir.

Notations

Abbreviations

Part 1

DNN	Deep neural network
IT	Interpolation threshold
RF	Random feature (model)
SNR	Signal-to-noise ratio

Part 2

FCN	Fully-Connected Network
CNN	Convolutional Neural Network
ViT	Vision Transformer
(G)(P)SA	(Gated) (Positional) Self-Attention

Part 3

(S)GD	(Stochastic) Gradient Descent
SK	Sherrington-Kirkpatrick (model)
SMT	Spiked Matrix-Tensor (model)
(D)FA	(Direct) Feedback Alignment

Part 4

SR	Symbolic Regression
GP	Genetic Programming
OEIS	Online Encyclopedia of Integer Sequences
E2E	End-to-end

General algebra

1_n	Vector of ones of size N
$\boldsymbol{a} \cdot \boldsymbol{b}$	Dot product of vectors $\boldsymbol{a}$ and $\boldsymbol{b}$
$\|\boldsymbol{a}\|$	L^2-norm of vector $\boldsymbol{a}$
I_n	Identity matrix of size N
J_n	Matrix of ones of size N
$\boldsymbol{A}^\top$	Transpose of matrix $\boldsymbol{A}$
$\boldsymbol{A} \odot \boldsymbol{B}$	Hadamard product of matrices $\boldsymbol{A}$ and $\boldsymbol{B}$
$\|\boldsymbol{A}\|$	Frobenius norm of matrix $\boldsymbol{A}$
$\operatorname{Tr} A$	Trace of matrix A
$\det A$	Determinant of matrix A
∇f	Gradient of f
$\operatorname{Hess}(f)$	Hessian of f

Probability distributions

$\mathcal{N}$	Gaussian distribution
$\mathcal{U}$	Uniform distribution
$\mathbb{P}[X = x]$	Probability of random variable X taking value x
$\mathbb{E}_Y[X]$	Average of random variable X when varying Y
$\mathbb{V}_Y[X]$	Variance of random variable X when varying Y
$\rho_M(\mu)$	Density of eigenvalues of matrix M

Specific to neural networks

N	Number of examples
D	Input dimension
C	Output dimension (or number of classes)
P	Number of parameters
L	Number of layers
η	Learning rate
λ	Regularization constant
$\mathcal{L}$	Loss function
ϵ_t	Training error
ϵ_g	Test (or generalization) error
σ	Activation function
ReLU	Rectified linear unit, $\operatorname{Relu}(x) = \max(0, x)$
$\operatorname{softmax}$	Softmax function, $\operatorname{softmax}(x)_i = \frac{e^{-x_i}}{\sum_j e^{-x_j}}$
$\tanh$	Hyperbolic tangent function, $\tanh(x) = \frac{\sinh(x)}{\cosh(x)}$
erf	Error function, $\operatorname{erf}(x) = \int_0^x \mathrm{d}t e^{-t^2/2}$

Contents

Foreword

Artificial intelligence is arguably the most important technological endeavour of the twenty-first century. Over the past decade, deep learning has become the cornerstone of this sprawling field, fueling breakthroughs in a plethora of contexts such as image classification [1], speech recognition [2], automatic translation [3] and board games [4]. Yet, theory lags far behind practice, and the key reasons underpinning the success of DNNs remain to be clarified.

The main objective of this thesis is to bridge this gap on both sides. In the first part, we analyze the ability of heavily overparametrized deep neural networks (DNNs) to learn without memorizing, armed with the toolbox of statistical physics. In the second part, we investigate the role of various architectural biases, and introduce techniques to combine their respective strengths. In the third part, we study dynamics in non-convex loss landscapes, and point to methods to ease learning in such contexts. As an opening, the last part presents a novel application of deep learning to symbolic regression, the task of inferring the mathematical expression of a function from its values.

Detailed summary

Introduction I have decided to open the thesis with a very basic introduction to deep learning, whose objective is to summarize the main findings of my PhD to a lay audience. The technical introductions to the various topics covered may be found in the subsequent chapters; in this way, each of the four parts of the thesis can be read independently. Note that for optimal readability, most of the technical details and computations are deferred to the Appendices.

Part I One of the the main puzzles of modern deep learning is the following: how do the huge DNNs used these days manage to generalize so well, even when they are able memorize the training data [5] ? In the first part of this thesis, we will tackle this question by building toy models and analyzing them via methods from statistical physics.

- **Chapter 2** We introduce the so-called double descent phenomenon, whereby performance always improves when increasing the number of parameters of a learning model, except near the interpolation threshold (the number of parameters at which the model can perfectly overfit the training data) where one finds an overfitting peak. This chapter is based on the following line of papers, the first relating the interpolation threshold to a jamming transition, the second demonstrating the existence of the peak, and the third studying its properties:

  [1] Geiger, M., Spigler, S., d'Ascoli, S., Sagun, L., Baity-Jesi, M., Biroli, G. & Wyart, M. Jamming transition as a paradigm to understand the loss landscape of deep neural networks. ***Physical Review E*** (2019)

[2] Spigler, S., Geiger, M., d'Ascoli, S., Sagun, L., Biroli, G. & Wyart, M. A jamming transition from under-to over-parametrization affects generalization in deep learning. ***Journal of Physics A*** (2019)

[3] Geiger, M., Jacot, A., Spigler, S., Gabriel, F., Sagun, L., d'Ascoli, S., Biroli, G., Hongler, C. & Wyart, M. Scaling description of generalization with number of parameters in deep learning. ***Journal of Statistical Mechanics*** (2020)

- **Chapter 3** We consider the random feature model, a toy model for DNNs which will be used in the rest of Part I to study the double descent curve from an analytical point of view. We begin by disentangling the various sources of variance affecting the test error, aiming to reconcile the double descent phenomenon with the classic bias-variance tradeoff. This chapter is based on the following publication:

  [4] d'Ascoli, S., Refinetti, M., Biroli, G. & Krzakala, F. Double Trouble in Double Descent: Bias and Variance (s) in the Lazy Regime. ***International Conference on Machine Learning*** (2020)[1]

- **Chapter 4** We study the impact of the activation function on the double descent curve, demonstrating the existence of two distinct kinds of overfitting which can cause peaks in the test error. This chapter is based on the following publications:

  [5] d'Ascoli, S., Sagun, L. & Biroli, G. Triple descent and the two kinds of overfitting: where and why do they appear? ***Advances in Neural Information Processing Systems*** (2020)

  [5′] d'Ascoli, S., Sagun, L. & Biroli, G. Triple descent and the two kinds of overfitting: where and why do they appear? ***Journal of Statistical Mechanics (special issue)*** (2022)

- **Chapter 5** We build a toy model of structured data to assess the impact of low-dimensional structures on learning, as well as studying the role of the loss function. This chapter is based on the following publication:

  [6] d'Ascoli, S., Gabrié, M., Sagun, L. & Biroli, G. On the interplay between loss function and data structure in classification problems. ***Advances in Neural Information Processing Systems*** (2021)

Part II To efficiently extract information from data, learning models usually rely on inductive biases: the typical example being that of convolutional neural networks (CNNs) [13]. However, with the increasing amount of data and compute available, models with weaker inductive biases such as vision transfomers [14] and MLP-mixers [15] have started to challenge the supremacy of CNNs. Hence, a crucial question for practitioners is how to select the right amount of inductive bias.

- **Chapter 6** We analyze the success of CNNs by viewing them as constrained fully-connected networks. We show in particular that convolutional constraints are useful in the early stages of learning, but can become restrictive later on. This chapter is based on the following publication:

  [7] d'Ascoli, S., Sagun, L., Biroli, G. & Bruna, J. Finding the Needle in the Haystack with Convolutions: on the benefits of architectural bias. ***Advances in Neural Information Processing Systems*** (2019)

[1] Underline denotes equal contribution.

- **Chapter 7** Inspired by the previous chapter, we introduce the ConViT, a new kind of vision transformer which can benefit from convolutional inductive biases in the early stages of learning but escape then later on if necessary. The latter was able to reach state-of-the-art sample efficiency among vision transformers. This Chapter is based on the following publication:

  [8] d'Ascoli, S., Touvron, H., Leavitt, M., Morcos, A., Biroli, G. & Sagun, L. ConViT: Improving Vision Transformers with Soft Convolutional Inductive Biases. ***International Conference on Machine Learning*** (2021)

- **Chapter 8** We push the idea of relaxing convolution constraints even further by introducing a method to recast fully pre-trained convolutional networks as transformers to boost their performance while increasing their robustness. This Chapter is based on the following preprint:

  [9] d'Ascoli, S., Sagun, L., Biroli, G. & Morcos, A. Transformed CNNs: recasting pre-trained convolutional layers with self-attention. *arXiv preprint arXiv:2106.05795* (2021)

Part III Aside from their extraordinary generalization performance, the ability of DNNs to find low-loss solutions in non-convex loss landscapes remains poorly understood. In this part, we will investigate the dynamics of learning in two very different scenarios.

- **Chapter 9** We study high-dimension inference problems, which have been much investigated in the statistical physics literature. In these problems, where we can control the complexity of the loss landscape, our objective will be to show the influence learning rate scheduling has on optimization; we will underpin several common empirical choices such as the inverse square root decay. This chapter is based on the following preprint:

  [10] d'Ascoli, S., Refinetti, M. & Biroli, G. On the optimal learning rate schedule in non-convex optimization landscapes. *arXiv preprint arXiv:2202.04509* (2022)

- **Chapter 10** We analyze the dynamics of Direct Feedback Alignment (DFA), a biologically plausible alternative to backpropagation. We uncover the align-then-select mechanism which underlies the success of DFA and underpin the reasons for its well-known inability to train CNNs. This chapter is based on the following publications:

  [11] Refinetti, M., d'Ascoli, S., Ohana, R. & Goldt, S. The dynamics of learning with feedback alignment. ***International Conference on Machine Learning*** (2020)

  [11′] Refinetti, M., d'Ascoli, S., Ohana, R. & Goldt, S. The dynamics of learning with feedback alignment. ***Journal of Physics A (special issue)*** (2022)

Part IV Having delved into many explorations on the success of DNNs, we close this thesis with a more applied section, investigating an innovative use-case: deep learning for symbolic regression, i.e. the task of inferring mathematical formulas from observations.

- **Chapter 11** We train a seq2seq Transformer (usually built for machine translation) to infer the recurrence relation of sequences of numbers, for example $[1, 2, 3, 5, 8, 13] \to u_n = u_{n-1} + u_{n-2}$. We demonstrate better performance than Mathematica on the famous Online Encyclopedia of Integer Sequences, and additionally handle the case of non-integer sequences. This chapter is based on the following preprint:

[12] d'Ascoli, S., Kamienny, P.-A., Lample, G. & Charton, F. Deep symbolic regression for recurrent sequences. ***International Conference on Machine Learning*** (2022)

- **Chapter 12** We apply a similar architecture to the more challenging task of symbolic regression of multi-dimensional functions with non-integer coefficients. Our model approaches the performance of the most advanced evolutionary algorithms with several orders of magnitude smaller inference time. This chapter is based on the following preprint:

  [13] Kamienny, P.-A., d'Ascoli, S., Lample, G. & Charton, F. End-to-end symbolic regression with transformers. *arXiv preprint arXiv:2204.51222* (2022)

Works not included in this manuscript

- I worked on a semi-supervised method to train models for text generation with low data resource during an internship with the french start-up Snips. This lead to the following publication, not included in the present manuscript:

  [14] d'Ascoli, S., Coucke, A., Caltagirone, F., Caulier, A. & Lelarge, M. Conditioned Text Generation with Transfer for Closed-Domain Dialogue Systems. ***Statistical Language and Speech Processing*** (2020)

  Communicating about science to the general audience has been one of the key objectives of my PhD, and I had the chance to write four popularization books on artificial intelligence and physics:

  [15] d'Ascoli, S. *Comprendre la révolution de l'intelligence artificielle* (**First**, 2020)

  [16] d'Ascoli, S. *L'Intelligence Artificielle en 5 minutes par jour* (**First**, 2020)

  [17] d'Ascoli, S. & Touati, A. *Voyage au coeur de l'espace-temps* (**First**, 2021)

  [18] d'Ascoli, S. & Bouscal, A. *Voyage au coeur de l'atome* (**First**, 2022)

Avant-propos

L'intelligence artificielle est sans doute le plus grand enjeu technologique de notre siècle. Ces dernières années, les réseaux de neurones artificiels sont passés maîtres de cette discipline tentaculaire, et ont permis des avancées majeures dans nombre de domaines: reconnaissance d'image [1], de voix [2], traduction automatique [3], jeux de société [4]... Pourtant, la théorie accuse un retard important sur la pratique, et les principales raisons de ce triomphe incontestable demeurent floues.

L'objectif principal de cette thèse est de réduire l'écart grandissant entre la théorie et pratique. Dans la première partie, nous analyserons la capacité des réseaux de neurons sur-paramétrisés à apprendre sans mémoriser, en se basant sur des méthodes de physique statistiques. Dans la seconde partie, nous étudierons le rôle que joue les choix architecturaux dans l'apprentissage, et introduirons des techniques pour combiner leurs avantages respectifs. Dans la troisième partie, nous nous intéresserons à la dynamique d'apprentissage dans les problèmes d'optimisation non-convexes, et proposerons des méthodes pour faciliter l'apprentissage. Enfin dans la dernière partie, nous présenterons une nouvelle application des réseaux de neurons à la régression symbolique, qui consiste à prédire l'expression mathématique d'une fonction à partir de ses valeurs.

Résumé détaillé

Introduction J'ai décidé d'ouvrir cette thèse avec une introduction très générale à l'apprentissage profond, qui permettra de résumer les questions abordées à un public non initié. Les concepts techniques égrénés au cours de ce manuscrit seront introduits au sein des différents chapitres correspondants; de cette façon, chacune des quatres parties de la thèse pourront être lues indépendamment. Pour une lecture plus fluide, j'ai aussi fait en sorte de reléguer les détails et calculs techniques dans les annexes.

Partie I L'un des plus grands mystères de l'apprentissage profond est le suivant: comment les réseaux de neurones gigantesques utilisés de nos jours parviennent-ils à généraliser si bien, même lorsqu'ils mémorisent parfaitement les données d'entraînement [5]? Dans la première partie de cette thèse, nous étudierons cette question à partir de modèles simples, solvables grâce à des méthodes de physique statistique.

- **Chapitre 2** Nous introduirons le phénomène de double descente, selon lequel la généralisation s'améliore sans cesse à mesure qu'on augmente le nombre de paramètres d'apprentissage, sauf près du seuil d'interpolation où les données sont tout juste mémorisées, où l'on observe un pic de surentraînement. Ce chapitre est basé sur les trois publications suivantes, la première étudiant la localisation du seuil d'interpolation, la deuxième démontrant l'existence du pic, et la troisième étudiant ses propriétés:

[1] Geiger, M., Spigler, S., d'Ascoli, S., Sagun, L., Baity-Jesi, M., Biroli, G. & Wyart, M. Jamming transition as a paradigm to understand the loss landscape of deep neural networks. ***Physical Review E*** (2019)

[2] Spigler, S., Geiger, M., d'Ascoli, S., Sagun, L., Biroli, G. & Wyart, M. A jamming transition from under-to over-parametrization affects generalization in deep learning. ***Journal of Physics A*** (2019)

[3] Geiger, M., Jacot, A., Spigler, S., Gabriel, F., Sagun, L., d'Ascoli, S., Biroli, G., Hongler, C. & Wyart, M. Scaling description of generalization with number of parameters in deep learning. ***Journal of Statistical Mechanics*** (2020)

- **Chapitre 3** Nous introduirons le *random feature model*, un modèle jouet de réseau de neurones artificiel qui nous servira de base théorique pour étudier le phénomène de double descente dans le reste de cette première partie. Nous décomposerons dans un premier temps les différentes sources de variance contribuant à l'erreur de généralisation dans le but de réconcilier la double descente avec le dilemme biais-variance. Ce chapitre est basé sur la publication suivante:

  [4] d'Ascoli, S., Refinetti, M., Biroli, G. & Krzakala, F. Double Trouble in Double Descent: Bias and Variance (s) in the Lazy Regime. ***International Conference on Machine Learning*** (2020)[2]

- **Chapitre 4** Nous étudierons l'impact de la fonction d'activation sur le phénomène de double descente, et démontrerons l'existence d'un deuxième pic de surentraînement qui peut entâcher la généralisation. Ce chapitre est basé sur les publications suivantes:

  [5] d'Ascoli, S., Sagun, L. & Biroli, G. Triple descent and the two kinds of overfitting: where and why do they appear? ***Advances in Neural Information Processing Systems*** (2020)

  [5′] d'Ascoli, S., Sagun, L. & Biroli, G. Triple descent and the two kinds of overfitting: where and why do they appear? ***Journal of Statistical Mechanics (special issue)*** (2022)

- **Chapitre 5** Nous étudierons la capacité des modèles d'apprentissage à vaincre la malédiction de la dimensionnalité en s'appuyant sur un modèle simple de données structurées. Ce chapitre est basé sur la publication suivante:

  [6] d'Ascoli, S., Gabrié, M., Sagun, L. & Biroli, G. On the interplay between loss function and data structure in classification problems. ***Advances in Neural Information Processing Systems*** (2021)

Partie II Une autre composante essentielle dans la généralisation des réseaux de neurones est leur architecture: les biais inductifs des réseaux convolutifs leur permettent d'apprendre avec bien moins de données et de paramètres, mais peuvent devenir restrictifs lorsque les données sont abondantes. Ainsi, des modèles aux biais infuctifs moins forts (Transformers [14] et MLP-mixers [15]) sont récemment venus remettre en question la suprématie de la convolution. Dans cette partie, nous étudierons donc l'impact des biais inductifs sur l'apprentissage et introduirons des méthodes pour bénéficier de leurs avantages sans leurs inconvénients.

[2]Underline denotes equal contribution.

- **Chapitre 6** Nous étudierons l'impact des biais convolutifs d'un point de vue théorique, en considérant leur effet sur la dynamique dans l'espace plus large des réseaux denses. Ceci permet de montrer que les biais convolutifs sont surtout utiles dans les premières phases de l'apprentissage, et peuvent se montrer restrictifs par la suite. Ce chapitre est basé sur la publication suivante:

  [7] d'Ascoli, S., Sagun, L., Biroli, G. & Bruna, J. Finding the Needle in the Haystack with Convolutions: on the benefits of architectural bias. ***Advances in Neural Information Processing Systems*** (2019)

- **Chapitre 7** Inspirés par les conclusions du chapitre précédent, nous introduirons le ConViT, une nouvelle architecture de Transformer qui peut bénéficier de biais convolutifs lors des premières phrases de l'apprentissage, puis s'en libérer par la suite. Ce chapitre est basé sur la publication suivante:

  [8] d'Ascoli, S., Touvron, H., Leavitt, M., Morcos, A., Biroli, G. & Sagun, L. ConViT: Improving Vision Transformers with Soft Convolutional Inductive Biases. ***International Conference on Machine Learning*** (2021)

- **Chapitre 8** Nous pousserons cette idée encore plus loin en introduisant une méthode pour reparamétriser n'importe quel réseau convolutif en un Transformer équivalent, qui peut par la suite être raffiné. Ce chapitre est basé sur la pré-publication suivante:

  [9] d'Ascoli, S., Sagun, L., Biroli, G. & Morcos, A. Transformed CNNs: recasting pre-trained convolutional layers with self-attention. *arXiv preprint arXiv:2106.05795* (2021)

Partie III Au-delà de leur extraordinaire généralisation, la capacité des réseaux de neurones à atteindre des minima globaux dans des problèmes d'optimisation a priori hautement non-convexes reste un défi théorique majeur. Dans cette partie, nous étudierons leur dynamique d'apprentissage dans deux contextes différents.

- **Chapitre 9** Nous étudierons plusieurs familles de problèmes d'optimisation emblématiques de la physique statistique des systèmes désordonnés. Dans ce contexte, nous étudierons le protocole optimal de diminution du pas d'apprentissage pour converger aussi rapidement que possible vers le minimum global sans rester coincé dans des minima locaux. Ce chapitre est basé sur la pré-publication suivante:

  [10] d'Ascoli, S., Refinetti, M. & Biroli, G. On the optimal learning rate schedule in non-convex optimization landscapes. *arXiv preprint arXiv:2202.04509* (2022)

- **Chapitre 10** Nous étudierons la dynamique d'un algorithme alternatif à la rétropropagation du gradient intitulé Direct Feedback Alignment (DFA). Ce chapitre identifiera les mécanismes pouvant aboutir au succès ou l'échec de cet algorithme, et se base sur les publications suivantes:

  [11] Refinetti, M., d'Ascoli, S., Ohana, R. & Goldt, S. The dynamics of learning with feedback alignment. ***International Conference on Machine Learning*** (2020)

  [11′] Refinetti, M., d'Ascoli, S., Ohana, R. & Goldt, S. The dynamics of learning with feedback alignment. ***Journal of Physics A (special issue)*** (2022)

Partie IV Après avoir étudié le succès des réseaux de neurones d'un point de vue théorique, nous finirons par une section de nature plus appliquée portant sur la régression symbolique, consistant à prédire l'expression mathématique d'une fonction à partir de ses valeurs.

- **Chapitre 11** Nous entraînement un Transformer (une architecture construite pour la traduction) à prédire relation de récurrence de séquences de nombres, par exemple $[1, 2, 3, 5, 8, 13] \to u_n = u_{n-1} + u_{n-2}$. Ce chapitre est basé sur la pré-publication suivante:

  [13] d'Ascoli, S., Kamienny, P.-A., Lample, G. & Charton, F. Deep symbolic regression for recurrent sequences. ***International Conference on Machine Learning*** (2022)

- **Chapitre 12** Nous présenterons une architecture similaire à la tâche plus difficile de prédiction de fonction à plusieurs variables avec coefficients non-entiers, et montrerons que celle-ci est capable de rivaliser avec les meilleurs algorithmes évolutionnaires actuels tout en réduisant le temps de calcul par plusieurs ordres de grandeur. Ce chapitre est basé sur la pré-publication suivante:

  [12] Kamienny, P.-A., d'Ascoli, S., Lample, G. & Charton, F. End-to-end symbolic regression with transformers. *arXiv preprint arXiv:2204.51222* (2022)

Travaux non inclus dans ce manuscrit

- J'ai eu l'occasion de développer une méthode de génération de texte semi-supervisée lors d'un stage au sein de la start-up Snips. Ces travaux ont abouti à la publication suivante:

  [14] d'Ascoli, S., Coucke, A., Caltagirone, F., Caulier, A. & Lelarge, M. Conditioned Text Generation with Transfer for Closed-Domain Dialogue Systems. ***Statistical Language and Speech Processing*** (2020)

- Raconter et vulgariser la science fut l'un des enjeux majeurs de ma thèse. J'ai eu la chance de pouvoir écrire quatre livres de vulgarisation, portant respectivement sur l'intelligence artificielle, la relativité générale et la physique quantique:

  [15] d'Ascoli, S. *Comprendre la révolution de l'intelligence artificielle* (**First**, 2020)

  [16] d'Ascoli, S. *L'Intelligence Artificielle en 5 minutes par jour* (**First**, 2020)

  [17] d'Ascoli, S. & Touati, A. *Voyage au coeur de l'espace-temps* (**First**, 2021)

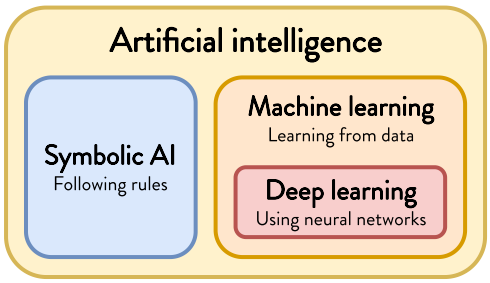
  [18] d'Ascoli, S. & Bouscal, A. *Voyage au coeur de l'atome* (**First**, 2022)

Chapter 1

Introduction

The objective of this chapter is to introduce deep learning and make the main contributions of this thesis understandable to the lay person.

1.1 A brief history of AI



Artificial Intelligence (AI) is a rather loose concept whose meaning has evolved with technological progress. Generally speaking, an AI is an algorithm whose purpose is to make "clever" decisions – although the relevance of "clever" is often debatable. A classic example of AI is the algorithm one faces in a game of chess against a computer. Two types of approaches exist:

- **Rule-based:** the computer chooses its next move according to a set of rules (in this situation, move your pawn; in this situation, capture the bishop; etc.).
- **Machine learning:** the computer chooses its next move according to its past experience. In other words, it learns by trial and error to estimate which move is most appropriate in each situation.

Machine Learning is therefore a type of AI algorithm which learns from examples to perform tasks for which it was not explicitly programmed. **Deep learning** is a sub-branch of machine learning where the algorithm learns by adjusting the parameters of an artificial neural network. These simulated brains are extremely powerful for complex tasks such as understanding images or language, and have fuelled most of the breakthroughs in AI since 2012. Deep learning will be the core of this manuscript, but before diving in, let us first take a step back to describe where this sprawling field originates from.

1.1.1 The birth of AI

The first mathematical algorithms appeared during Antiquity; Euclid's famous algorithm allowing one to find the greatest common divisor of two numbers is still taught in schools these days. Binary logic, which underlies modern computers, was developed in incremental steps, first with Gottfried Leibniz in 1702, then Georges Boole in 1854. The first computer program was written in 1842 by Ada Lovelace,

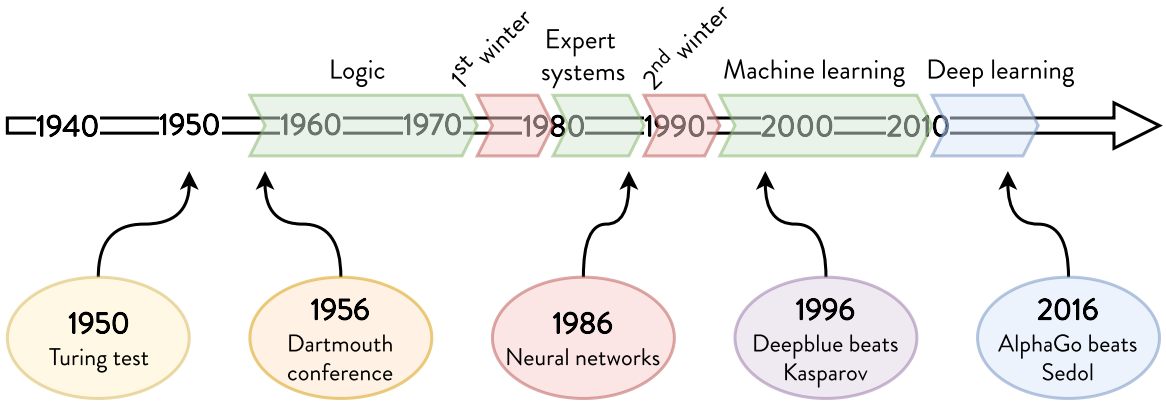


Figure 1.1: **A timeline of AI.**

daughter of famous poet Lord Byron. Her program was designed to run on the analytical machine, an ancestor of the modern computer imagined by her mathematician friend Charles Babbage.

Computers as we know them today only appeared in the 1940s. Those same years, famous English mathematician Alan Turing proved that computers could simulate any logical process, and neuroscientists Warren McCulloch and Walter Pitts devised the first simplified mathematical model of the neurons in our brain.

In the summer of 1956, American researchers Marvin Minsky and John McCarthy organized a scientific workshop in the small American town of Dartmouth, near Boston, to discuss the idea of programming machines to “think” autonomously. It is on this occasion that McCarthy coined the words "Artificial Intelligence" as the name of a field of study in its own right.

1.1.2 Connectionnist and symbolic AI

From its birth to the 1970s, AI steadily flourished, pioneered by two confronting approaches:

- **Symbolist AI**, which aims to reason via a “symbolic” (human-readable) language that computers can comprehend. This relates to the rule-based algorithms described earlier.
- **Connectionist AI**, which follows a more pragmatic approach: endowing the machine with a set of parameters (or, by analogy with the brain, "connections") which can be adjusted by trial-and-error. This approach belongs to what is nowadays called machine learning (however, not all machine learning algorithms are parameter-based).

In 1957, American psychologist Frank Rosenblatt laid the foundations for connectionnism with the perceptron, a large machine that mechanically simulates a biological neuron. Those same years, symbolic algorithms were devised to solve simple puzzles, such as the backtracking algorithm designed to escape mazes.

1.1.3 AI winters

Early AI researchers had a tendency to be over-optimistic, often claiming that solving human intelligence would be a matter of a few decades. During the 1970s, it became clear that AI was not going to be as easy

a challenge as they imagined, and the field entered a rollercoaster of alternating phases of enthusiasm and pessimism.

First winter From the mid-1970s onwards, a major obstacle stood in the way of progress: the sluggishness of the computers of the time. Pessimism took over, funding dried up and so came the first "winter of AI". On one hand, perceptron-based algorithms lost credibility as Minsky pointed out their inherent limitations. On the other hand, symbolic algorithms failed as soon as the problems involved too large solution spaces to search through.

Expert systems In the early 1980s, AI came back to life with expert systems. These symbolic algorithms excel in very specific areas thanks to rules established by human experts. The first industrial applications of AI started sprouting and research funding picked up. Connectionism also benefitted from this sudden resurgence. In 1982, American neuroscientist John Hopfield designed a neural network capable of simulating memory. In 1989, Yann LeCun introduced the first neural network capable of recognizing handwritten numbers. But the explosion of deep learning was still a long way off: LeCun's contribution – as well as his colleagues Geoffrey Hinton and Yoshua Bengio – was only rewarded with the Turing Prize in 2019.

Second winter At the end of the 1980s, the craze slowed down again due to the limitations of expert systems. Not only were they specialized to a single, very specific task - they were often pretty bad at it. Expert systems were far from fulfilling the hopes of a true general AI that had motivated fundings, and along came the second winter of AI.

1.1.4 The emergence of deep learning

In the 1990s, spring returned again. Computing power, which was hindering the progress of AI, began to double every two years thanks to the advances in miniaturization, a phenomenon known as Moore's law. The first AI breakthrough hit the headlines in 1996 when DeepBlue, a bulky computer designed by IBM, defeated chess legend Garry Kasparov. However, this famous event was no revolution from an AI point of view: DeepBlue was simply an over-powered expert system, capable of crunching through hundreds of millions of options per second.

In 1993, Vladimir Vapnik introduced one of the most famous machine learning algorithms: the support vector machine, essentially an improved version of the perceptron. Connectionist algorithms began to gather more attention than symbolic algorithms. The real revolution occured in the 2010's, when AI entered the deep learning era. Why did it happen so late, when artificial neural networks had been around for decades? Mainly because deep learning algorithms were lacking two crucial resources:

- **Compute power.** Moore's law helped considerably in this matter, but also a brilliant new idea: running computations on graphical processing units (GPUs), which allow huge numbers of simple calculations to be carried out in parallel. This is particularly suited to the training of neural networks, which learn by adjusting millions or billions of parameters.
- **Training data.** These are the examples that neural networks learn from. To perform complex tasks such as image recognition, a large amount of data is needed. Luckily, data started flowing with the internet era, and large public datasets were released for research purposes, such as the ImageNet image bank in 2009.

In 2012, Geoffrey Hinton and his students took the whole AI community by surprise by crushing the prestigious ImageNet image recognition competition using artificial neural networks running on GPUs. This marked the beginning of the deep-learning era: the following year, the same competition was utterly dominated by neural networks. In 2016, two decades after the triumph of DeepBlue, british company DeepMind made the headlines across the world when its algorithm AlphaGo defeated South-Korean champion Lee Sedol by four games to one. Mastering the game of Go, far more complex than chess, was until then considered far beyond the reach of AI.

1.2 Basics of machine learning

Let us now explain the basics of machine learning with a very simple example: selling an old motorcycle, bought three years ago. To estimate its resale price, one needs to figure out how much it decreases with time. One way to do this is to visit a second-hand sales website, collect a few examples of motorcycles, and record their age and resale price as dots on a graph (see Fig. 1.2).

It is easy to notice that the points are almost aligned along an imaginary line. Once this line is drawn, it can be used to predict the resale price of the motorcycle, as shown in the figure below. Machine learning algorithms do exactly this: they learn from examples we show them, and hope to generalize what they learn to examples they have never seen before.

1.2.1 Training

Adjusting parameters The example above is called linear regression, and happens to be both the simplest and one of the most widespread machine learning algorithms. Its goal is to find the "best line", the one that fits through the points as closely as possible. To do this, the algorithm only needs to adjust two variables: the slope of the line and its height. The idea behind machine learning is to fiddle with these two parameters until the best line is found.

In this example, the data is very simple, and two parameters are enough to do the job. But for complex tasks, many more parameters are needed: modern neural networks use up to a trillion parameters, and our brain uses even more. The question is: how can one find a good configuration with so many parameters to deal with?

The loss function A very common principle in machine learning is to quantify the performance of the algorithm with a score, which is called the loss function (the higher the loss, the poorer the performance). In the example above, the loss function will be higher if the line is far away from the points in the graph.

Another example, more akin to deep learning, is that of an algorithm learning to distinguish between cats and dogs. Given a picture of a cat, the algorithm would output something like: “80% cat, 20% dog”. To obtain a perfect score, i.e. bring its loss function down to zero, the algorithm would need to predict “100% cat, 0% dog” when it is shown a cat and “0% cat, 100% dog” when it is shown a dog. When shown a cat, predicting “80% cat, 20% dog” would entail a small penalty - i.e. increase in the loss- for being under-confident. Predicting “20% cat, 80% dog” would entail a stronger penalty.

Gradient descent To learn, the algorithm needs to adjust its parameters to decrease the loss function. Think of the set of parameter configurations as a gigantic mountain range we explore by adjusting the

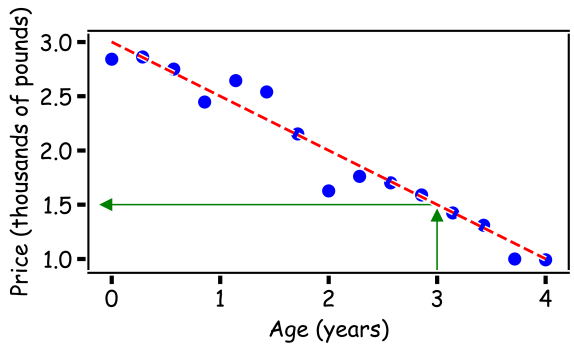


Figure 1.2: **The simplest machine learning algorithm: linear regression.** The dots are more or less aligned along the red line. From this line one can predict the resale price of the motorcycle bought 3 years ago: 1500 pounds.

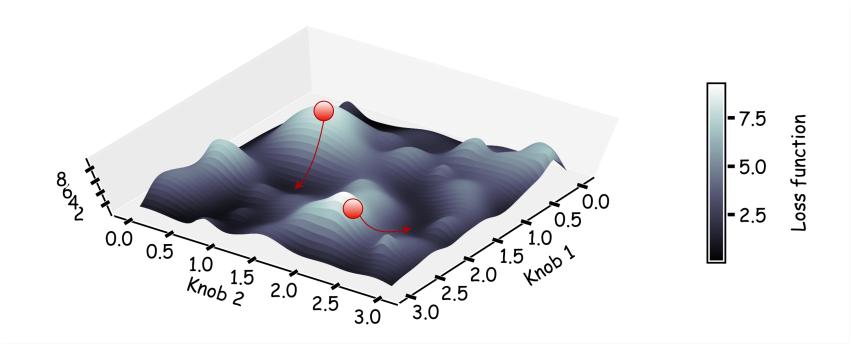


Figure 1.3: **Gradient descent algorithm:** The idea is to start from a random point, then minimize the loss by following the slope (i.e. the gradient) of the loss landscape. The final configuration reached strongly depends on the starting point!

parameters – we call this the loss landscape. At each position, the altitude represents the value of the loss function.

To minimize the loss, the idea is to estimate the *gradient* of the loss, i.e. the direction of the slope at a given point. Luckily, the gradient can be computed with a rather simple mathematical formula. Then, take a small step in the downwards direction of the slope: we call this algorithm *gradient descent*. Once we reach a point where the landscape is flat , we declare that the learning process is over. This ideally occurs when we reach the bottom of the mountain (i.e. reach the lowest possible value of the loss), but one can also get trapped in a basin somewhere higher.

This procedure is illustrated in Fig. 1.3. Note that the point reached strongly depends on where we start from: this is why the initialization of the parameters is a crucial step in machine learning, which will be thoroughly studied in this thesis.

1.2.2 Generalization

The more parameters an algorithm has at its disposal, the greater its ability to decrease the loss function and therefore solve complex problems. However, in machine learning, decreasing the loss function too far is not always a good idea...

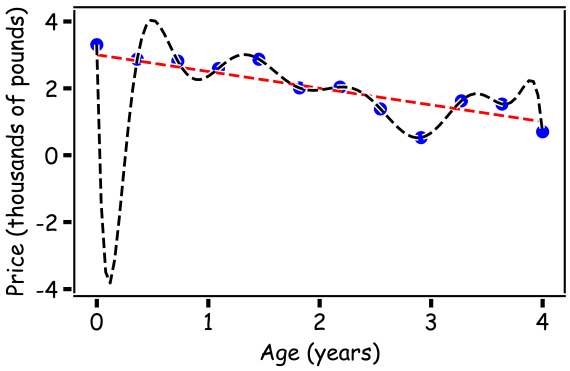


Figure 1.4: **Overifitting:** The algorithm predicting the red line (linear regression) is much better than the one predicting the black line (polynomial regression). In the first case, the algorithm had just the right number of parameters (two), while in the second case, it had too many (twelve), which led to overfitting.

Overfitting The loss function reflects how well the algorithm learnt the training examples. However, achieving a low loss does not guarantee that the algorithm will perform well on data it has never seen, i.e. be able to generalize.

Let's go back to the task of predicting the price of motorcycles, and imagine that the algorithm has succeeded in reaching the optimal value of zero for the loss function. This means the algorithm has adjusted its parameters so accurately that it has memorized the exact selling-price of each of the thousands of motorcycles presented to it during the training (see Fig. 1.4), instead of simply estimating the decreasing trend of the resale price over the years. This kind of memorization is what we call overfitting in machine learning, and is the reason why the training phase is always followed by a testing phase, where we evaluate the predictions of the model on unseen data.

Bias-variance tradeoff Overfitting is what makes machine learning both frustrating and interesting. Trying to over-optimize the loss function by using too many parameters or training for too long can result in diverting the algorithm from its real objective, which is generalization.

Indeed, there is some random variability in any dataset: the price of motorcycles is not perfectly aligned along a straight line: it is "noisy". If an algorithm tries to over-optimize its loss function, it will eventually learn the noise -i.e. non-essential features- rather than the meaningful features in the data. In other words, it over-specializes on the training data it has been shown[1].

[1]Argentinian author Jorge Luis Borges illustrates this rather subtle notion of generalization in his short story Funes or Memory, in which the eponymous hero suffers from infinite memory. Funes pays so much attention to detail that he doesn't understand how the generic name "dog" can encompass creatures of such different sizes and shapes. His hyperthymesia prevents him from identifying general features: he cannot understand the world surrounding him because according to the

When choosing the number of parameters in a learning model, one needs to find a sweet spot between:

- **Underfitting:** an excessively simple algorithm - one with too few parameters - will not extract enough information: it suffers from what is called high bias.
- **Overfitting:** an excessively complicated algorithm - one with too many parameters - will store unnecessary information. It suffers from what is called high variance: it is very sensitive to noise, as shown in Fig. 1.4.

These are the principles underlying the bias-variance trade-off, illustrated in Fig. 1.5. Luckily, learning meaningful features is often easier than learning noise, which is often chaotic and unpredictable (it's easier to memorize a poem than the random pattern of a phone number). Using just the right amount of parameters leads the algorithm to detect the general trend of the data, before trying to memorize the superfluous noise.

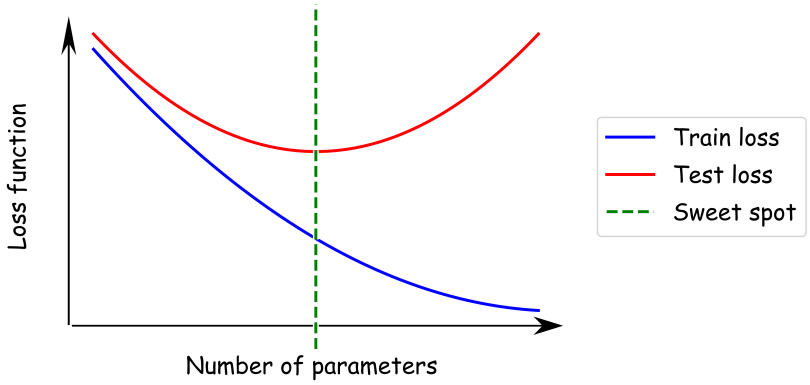


Figure 1.5: **Bias-variance trade-off:** simple algorithms, with few parameters, suffer from a high bias (blue curve), whereas algorithms that are too complex, with too many parameters, suffer from a high variance (red curve). In the middle, there is a sweet spot where the test error (black curve), which is the addition of these two values, is rather low, as shown in green.

Regularization Going back to our example of motorcycles, one could have suspected that two parameters were sufficient to assess the resale price, since the points are distributed around a straight line. But figuring out the right number of parameters is very hard in general. One thing is for sure: if the learning model has too few parameters, it will miss out on some pieces of information. With too many parameters, all the relevant information can be captured, but there is a risk of overfitting.

One solution is be to give the algorithm more than enough parameters, to make sure it doesn't miss out on anything, but encourage it to use as little as possible[2]. This is the idea behind regularization methods, among which two are very common:

author, "thinking is about forgetting differences".

[2]In the 14th century, English philosopher William of Occam introduced a method of reasoning known as Occam's razor: Pluralitas non est ponenda sine necessitate (multiples should not be used without necessity). This is exactly the principle of regularization: amongst all possible solutions to a problem, choose the simplest one.

- **Early stopping:** Interrupt the training prematurely so that the algorithm has a limited time to adjust its parameters. Just like taking the plate away before the end of the meal.
- **L1 / L2 regularization:** Entice the algorithm to use as little parameters as possible (L1), or use each parameter parsimoniously (L2). In other words, punish the algorithm for every bite it takes.

1.2.3 Hyperparameters

We have just seen that selecting the size of a learning model is not an easy matter: too few parameters leads to underfitting, too many leads to overfitting. Regularization allow to combat overfitting, but only shifts the problem, which becomes: how to select the right amount of regularization?

Experimental settings such as the size of a learning model or the amount of regularization are called *hyperparameters*, to avoid confusion with the parameters of the learning model, those which are adjusted automatically during training. Other famous hyperparameters are the *learning rate* (the size of the steps taken during gradient descent) and the *batch size* (how many examples are considered to compute each step).

Hyperparameters cannot be learnt automatically, and need to be manually selected by us, machine learners. They are ultimately selected in order to provide the best possible generalization. The most common workflow, named *grid search*, is to test a few values for each hyperparameter, and select the best combination. However, the number of hyperparameters involved in training modern neural network is often very large, which makes hyperparameter selection a challenging engineering problem.

Note that if one selects hyperparameters based on the results on the test dataset, we may bias our selection towards the specific examples which it contains – in some sense, *overfit* the test dataset. This is why one generally uses two distinct datasets: a *validation set* to select the best hyperparameters, and a fresh *test set* to assess the true performance of our model in an unbiased way.

1.3 Deep learning

Deep learning is at the crux of modern machine learning and has fuelled most of the recent breakthroughs in AI. In this section, we present its inner workings, starting from the very basics of a single neuron.

1.3.1 The perceptron

The perceptron is a mathematical model for a neuron, the basic building block of neural networks. Take a very concrete case: detection of melanoma (cancerous moles). Dermatologists generally recommend the ABCDE rule for self-examination, which consists in looking out for five distinctive features: Asymmetry, uneven Borders, unhomogenous Colour, large Diameter, fast Evolving.

To decide whether a mole is benign or malignant, one can use a perceptron. It works in a very similar way to linear regression, simply by assessing the impact of each of the ABCDE features thanks to a parameter adjusted during training (see Fig. 1.6). The difference here is that we want to classify (predict a category), not regress (predict a value)! In the example of motorcycles, the output variable was a price, which needed to be as close as possible to the market resale price. In the present case, the output variable is a score which must be positive (larger than zero) for melanomas and negative (smaller than zero) for harmless moles. If the score is zero, the perceptron is uncertain.

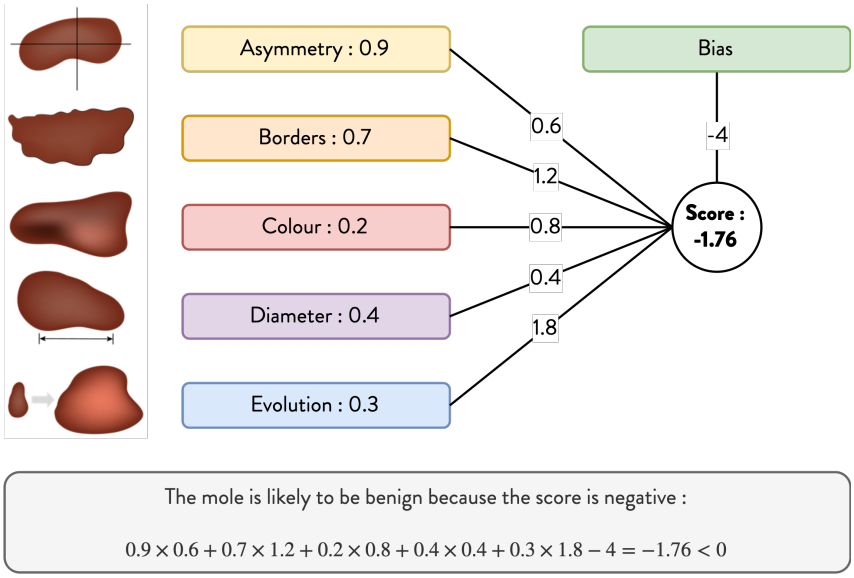


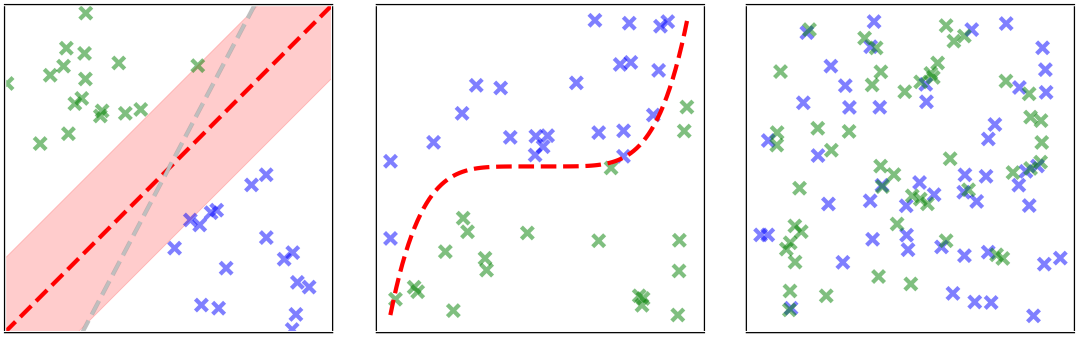
Figure 1.6: **The perceptron:** it outputs a single number (here, the likelihood that the mole is cancerous) by calculating a weighted sum of the input features.

Limitations of the perceptron Mathematically, the perceptron looks for a decision boundary separating moles and melanomas in the five-dimension space of ABCDE features. However, the strong limitation of the perceptron is that it can only represent *linear* decision boundaries, i.e. straight planes instead of curved surfaces, as illustrated in Fig. 1.7. This will only work in the case of *linearly separable* data, which is rather rare in practice (see Fig. 1.7).

Furthermore, even in the case of linearly separable data, there are several ways to draw a line separating the categories. If you were asked to draw a decision boundary in the left panel of the figure below, you would probably naturally choose the one in red rather than the one in grey. It seems more natural, because it is halfway between the two clusters: it maximizes its margin (the red area) with respect to the most ambiguous points and is more likely to classify new points correctly. The problem with the perceptron is that it doesn't care about maximizing the margin: any line which correctly separates the categories will do!

Improving the perceptron Fortunately, there are solutions to overcome the perceptron's limitations. To satisfy the margin maximization principle, one can use algorithms such as *support vector machines*. To deal with non-linearly separable data, *nearest neighbour* methods can be used, which consists in probing the surroundings of the cross whose colour we want to infer: if most of its neighbouring crosses are blue, then it will be declared blue, otherwise it will be declared green. Finally, a more sophisticated way to go are *kernel methods*, which probe all the other crosses, not just the nearest neighbours, giving less weight to far away crosses.

Support vector machines and kernel methods largely contributed to the victory of machine learning over symbolic AI. But to cope with even more difficult situations such as that in the right panel of Fig. 1.7, artificial neural networks have become essential nowadays.



(a) Linearly separable (b) Nonlinear boundary (c) Even trickier boundary

Figure 1.7: **Linear separability.** Left: Linearly separable data, for which there are several possible separators. While the perceptron separates the data according to the grey border, vector support machines prefers the red border, which maximizes the margin (red zone) with respect to the most ambiguous data. Center: Non-linearly separable data: the boundary is curved. Right: non-separable data.

1.3.2 Neural networks

Artificial neural networks are built from layers of neurons stacked in a pile. Each neuron (i.e. each perceptron) in a given layer sums the signals it receives from the neurons in the previous layer and transmits a new signal to the neurons in the next layer. Just like in our brain, information is sent from one neuron to the next through synapses, which are the parameters (often called "weights" in the context of deep learning) to be adjusted during training. In the most powerful neural networks, several hundred successive layers are sometimes used – hence the name “deep” learning.

Activation function There is a magical ingredient which allows neural networks to escape the linear world of the perceptron. Between each layer, the signal goes through an activation function, which modulates it in such a way as to introduce non-linearity, i.e "bend" the decision boundary. The word “activation” comes from neuroscience, because this is very similar to what happens in our brain. Our neurons also operate in a non-linear way: the signal they transmit is not proportional to the signal they receive. Instead they follow an all-or-nothing principle, transmitting a signal only when the sum of the signals they receive exceeds a certain threshold.

Feature extraction The great advantage of neural networks is that they learn to extract relevant features from the data they are given autonomously. Until they became mainstream, a large part of machine learning tasks was to manually select these features, a painstaking process known as feature engineering.

Consider again melanoma detection. Before the deep learning era, the algorithm would be given the ABCDE features of the mole. But with deep learning, things are much simpler: we can directly feed a raw picture of the mole to the neural network, which then takes care of identifying the most relevant features. In addition to saving time and effort, this procedure often turns out to be much more effective: when reducing the complexity of a picture to a few features, we potentially miss out on other important features! Being able to effortlessly compare millions of pictures of moles, neural networks can identify

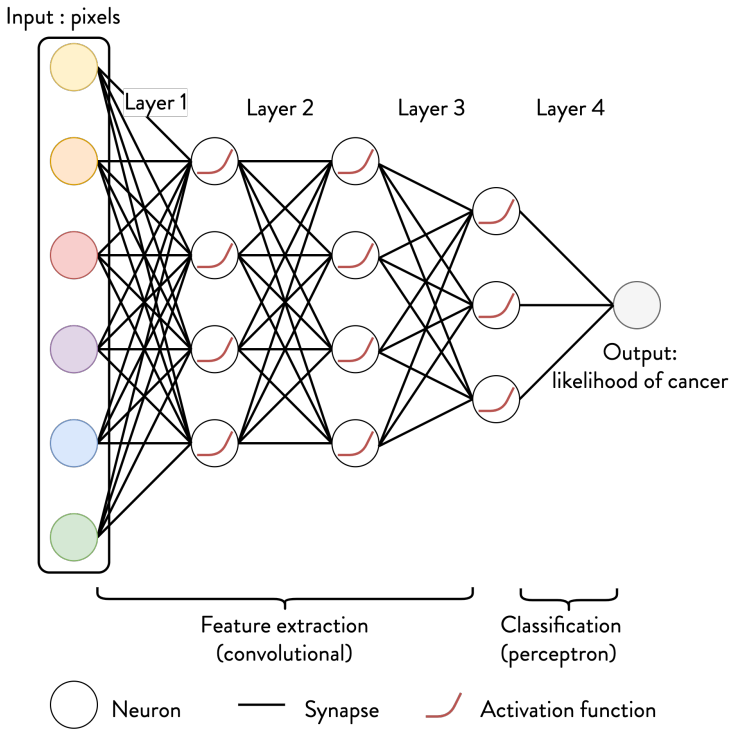


Figure 1.8: **An artificial neural network.** The pixels of the input image are analysed by a first layer of neurons, which transmits the information to a second layer then to a third. These first three layers extract important information from the image and pass it on to the last layer, known as the decision layer.

complex distinctive patterns which even the best pathologists wouldn't notice.

A hierarchical process Feature extraction is generally performed in a hierarchical way. Intuitively, the first layer extracts coarse features, such as the borders of the mole. The second layer extracts slightly more subtle features, such as the shape and texture of the mole. The deeper we go inside the neural network, the more complex the features become. After a few layers, it is generally impossible from our viewpoint to understand what the neural network is doing. The layers nearest the output are generally responsible for combining together all the extracted features and condensing them down to a smaller, neater bunch of features. Hopefully, once they reach the final layer, the features are linearly separable and can easily be classified.

Transfer learning Another great asset of neural networks is that just like our brain, they have a certain degree of plasticity. Transfer learning enables algorithms trained on a given task to adapt to a new one. For example, a facial recognition algorithm could be reused to detect melanomas. Indeed, the feature extraction taking place in the first layers happens to be a rather universal procedure: by tweaking its final layers only, the algorithm can adapt to a new task. This provides a cheap way to re-use pre-trained networks !

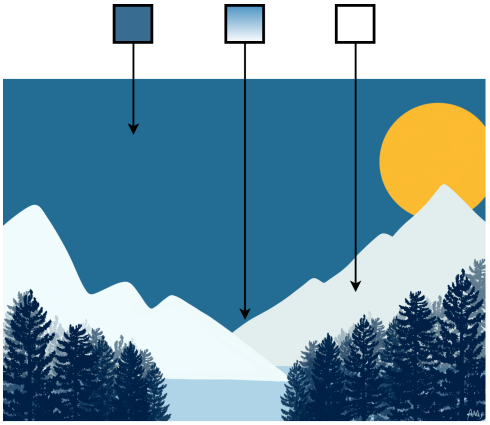


Figure 1.9: **The convolution mechanism.** The blue filter detects the sky, the white filter detects the snow, the half-blue half-white filter detects the sky-snow interfaces.

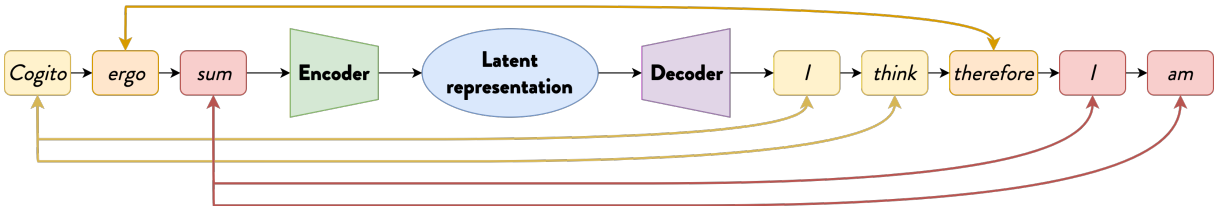


Figure 1.10: **The attention mechanism.** The original sentence, in Latin, is encoded into a latent representation then decoded using the attention mechanism. The attention mechanism tells the decoder to turn its attention to "cogito" for the beginning of the sentence "I think", to "ergo" for "therefore", and finally to "sum" for the end of the sentence "I am".

1.3.3 Deep learning architectures

Convolutional networks The neural network depicted in Fig. 1.8 is the most basic kind of architecture, named *fully-connected* because the neurons of a given layer are connected to all the neurons of the neighboring layers. The issue here is that the number of connections becomes prohibitively large when dealing with high-dimensional inputs such as images (several million pixels with three colours each).

To handle such long sequences of inputs while keeping a reasonable number of parameters to learn, the field of computer vision generally relies on convolutional networks, which scan images by patches, and reuse the same parameters across different patches. They extract local features using convolutional filters, which scan the whole image looking for a specific feature. For example, in a picture of snowy mountains, blue filters will detect the sky; white filters will detect snow; half-blue half-white filters will detect boundaries between sky and snow, as illustrated in Fig. 1.9. This procedure is very similar to how the neural networks in our visual cortex work.

Attention-based networks Convolutional networks use a very strong inductive bias: each layer only captures local features. Luckily, these features are usually the most important (neighboring pixels are more correlated than far-away ones), which explains the empirical success of convolutional netorks. This inductive bias significantly reduces the number of parameters and the number of images required to learn, but limits their ability to capture long-range information.

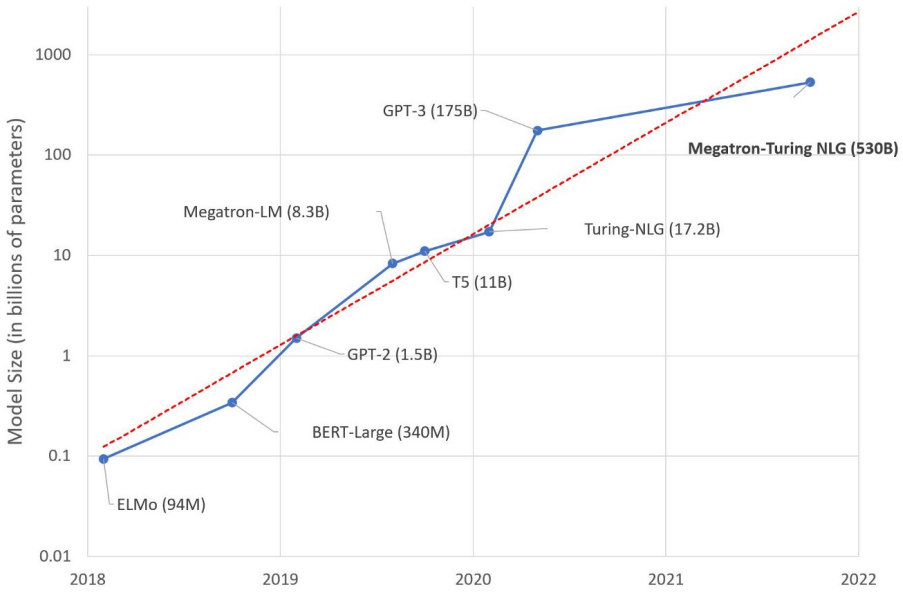


Figure 1.11: **Deep learning beats Moore's law:** since a few years, the size of the largest language model available increases by an order of magnitude every year. Source: *Medium*.

One way to circumvent this limitation is to use a mechanism called *attention*, allowing the network to decide which parts of the input are semantically connected. This kind of architecture was initially introduced as an add-on to machine translation models, enabling them to better capture which words of the original sentence words the current word of the translated sentence relates to, as illustrated in Fig. 1.10. Yet, it has rapidly spread to all areas of deep learning, including computer vision, and will be central in Parts II and IV of this thesis.

Again, this kind of architecture is human-inspired: attention is what governs our perception of the world. Amongst all the sensory impulses triggered by our skin, sounds perceived by our ears and light received by our eyes at each instant, we retain only a tiny fraction: that which is the most useful at each particular instant.

1.4 My contribution in a nutshell

1.4.1 Overparametrization

Modern deep learning is characterized by neural networks of ever-increasing size. This is illustrated in Fig. 1.11: every year since 2018, the size of the largest language model available increases by an order of magnitude. The largest of them at the time this manuscript is written are approaching a trillion parameters. This is still a few orders of magnitude smaller than the number of synapses in a human brain (which is around 1000 trillion [29]), but if we could keep the current pace, the gap could be closed in less than a decade.

There is however something paradoxical in this race for large models: in Sec. 1.2.2, we mentioned that machine learning models tend to overfit when they have too many parameters. The extreme case of overfitting occurs when the model has enough parameters to bring the training loss down to zero:

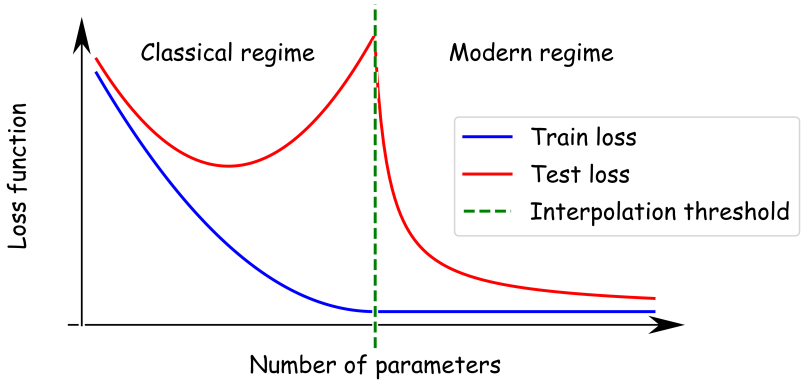


Figure 1.12: **The double descent curve.** In deep learning, generalization (performance on unseen data) is poor near the interpolation threshold due to overfitting, but improves as we move beyond.

we call this the *interpolation threshold*. Yet, recent work suggests that modern deep learning models operate in a highly *overparametrized* regime, well beyond the interpolation threshold, and still manage to generalize well, even without regularization [5]! Two questions naturally arise from this observation: (i) how and when are deep learning models able to reach zero loss solutions, and (ii) why do these solutions generalize so well?

In Chap. 2, we show that the interpolation threshold occurs when the number of parameters is of same order as the number of training data, independently of the architecture of the model or the dimension of the data. At this point, one typically observes overfitting and generalization will be poor, as predicted by the bias-variance tradeoff. However, if one keeps increasing the size of the model, something remarkable happens: performance starts improving again! In other words, Fig. 1.5 is incorrect: instead of observing a continuously decreasing blue curve, one observes a "double descent" curve, as sketched in Fig. 1.12.

In Chap. 3, we study this curve from a theoretical point of view, to try and understand how performance can improve beyond the interpolation threshold. We underpin the following mechanism: when we overparametrize beyond the interpolation threshold, the model has already perfectly overfit the data, so overfitting is not a problem anymore. In fact, the model memorizes the data in a smoother way (i.e. with lower variance) as we give it more parameters, which enables it to generalize better!

In Chap. 4 and 5, we study how this phenomenology is impacted by the activation function, the loss function and the properties of the data the model is trained on. The latter point is of utmost importance: in principle, the *curse of dimensionality* should make learning problems exponentially harder as the dimensionality of the input data increases[3], yet neural networks are able to scale seamlessly to million-pixel images by taking advantage of the low-dimensional structure of the data.

1.4.2 Architectural bias

In the first part, we studied overparametrization as key to the power of deep neural networks. Yet, size is not the only thing that matters. Another key aspect in building a successful neural network

[3] since the volume of a sphere of radius ϵ in dimension D scales as ϵ^D, the number of input data required to fill in a sphere scales exponentially with D.

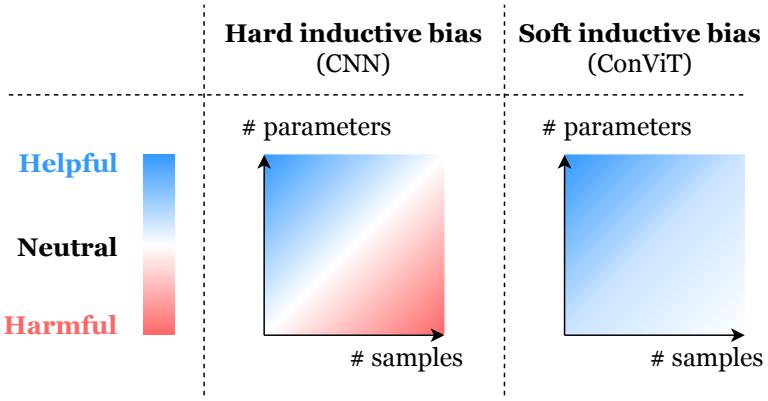


Figure 1.13: **Soft inductive biases can help models learn without being restrictive.** Hard inductive biases, such as the architectural constraints of CNNs, can greatly improve the sample-efficiency of learning, but can become constraining when the size of the dataset is not an issue. The soft inductive biases introduced by the ConViT avoid this limitation by vanishing away when not required.

is to choose the right architecture given a particular type of data. As explained in Sec. 1.3.3, a very important technique to efficiently extract information from data is to use inductive biases, i.e. exploit prior knowledge on the structure of the data at hand. For example, convolutional networks assume two things:

- **Locality:** two neighboring pixels are more closely correlated than two far-away neighbors, i.e. most of the information in an images lies in local correlations
- **Translation invariance:** a dog will remain a dog whether it is in the center of the image or in the top left corner: i.e. the category of an object is insensitive to its position.

These inductive biases are hard-wired into the architecture of convolutional networks, and help them learn from less data. However, they can restrict the ability of the learning model to learn features which do not fall under one of these two categories. In particular, it hinders the ability of neural networks to capture long-range information, e.g. understand the relationship between distant parts of the image.

In Chap. 6, we demonstrate this by introducing a method to remove the convolutional constraints during training – namely, recasting the convolutional network as a fully-connected one. We show that performance degrades when the constraints are removed in the first stages of learning, but improves when they removed half-way through learning. Hence, the constraints are helpful early on, but become restrictive later on. Yet, our method is not suited for practical applications: extracting long-range information comes at the cost of using a very large fully-connected network.

A smarter way to capture long-range information is to use attention-based networks. In Chap. 7, we employ a very similar procedure to infuse convolutional constraints within an attention-based network in a *soft* way – the network can escape the constraints if necessary, see Fig. 1.13. The resulting architecture, named ConViT, significantly improves the ability of Transformers to learn from small datasets, while also boosting their performance on large datasets.

In Chap. 8, we use the opposite approach of relaxing the constraints of convolutional network at late times. In both cases, we obtain competitive results in image classification tasks, and our methods have been applied to many different tasks, ranging from volcano detection [30] to skin lesion classification [31].

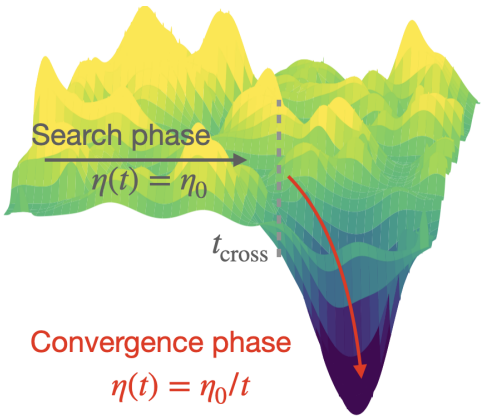


Figure 1.14: **The optimal learning rate schedule depends on the structure of the landscape.** One must first keep a large constant learning rate to escape the rough parts of the landscape as quickly as possible, then decay the learning rate as $\eta(t) = \eta_0/t$ once inside a convex basin.

1.4.3 Optimization

Remember from Sec. 1.2.1 that reducing the loss function is akin to walking down a mountain. The size of the steps we take is key to descend as quickly as possible. In the early stages of training, it is desirable to take large steps to speed up the descent, but when we reach a basin, we need to take smaller steps in order to not overshoot and miss the bottom. A key question is the following: what is the optimal rate to reduce the size of the steps? This question has been studied extensively for simple landscapes, but much less so for rough landscapes.

In Chap. 9, we show that the answer turns out to strongly depend on how rough the terrain is. In smooth regions, which do not have local basins where one can get trapped in, one can decay the step size rather quickly. However, in rough regions, decaying the learning rate too fast can slow down the dynamics. In practice, we show that a broad class of optimization problems exhibit a search phase where the landscape is rough and the learning rate should be kept constant, followed by a convergence phase where the landscape is smooth and the learning rate can safely be reduced (see Fig. 1.14).

In Chap. 10, we study how the descent works for an algorithm called Direct Feedback Alignment, which enables to update the parameters of neural networks in a way which is more compatible with the mechanisms of biuological brains. We show that this algorithm drags the optimization path to a special region of the landscape where the neural networks can indeed learn.

1.4.4 Symbolic mathematics

The final part of this thesis explores the following idea: teaching a neural network to predict the mathematical formula underlying a set of observations (see Fig. 1.15). This task, known as *symbolic regression*, is particularly interesting in light of our discussion in Sec. 1.1.2 on connectionnist versus symbolic artificial intelligence: a mathematical formula is the archetype of a symbolic expression, and such tasks are usually tackled with classic approaches such as genetic programming (exploring the space of formulas by applying random mutations to the current guess and selecting the fittest offspring at each new generation).

In Chap. 11, we train attention-based models usually built for translation to predict the recurrence relation behind number sequences such as [1,1,2,3,5,8,13...] (here, the recurrence relation is that the last

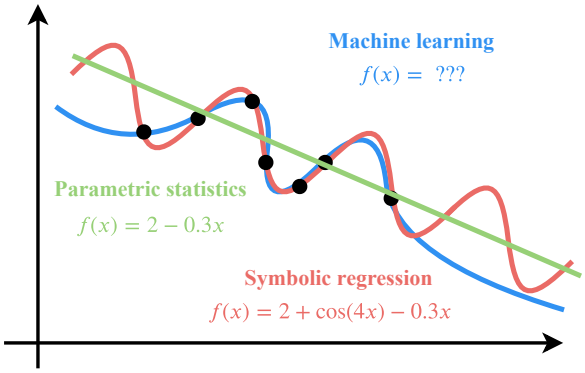


Figure 1.15: **Symbolic regression** can be seen as a middle ground between parametric statistics, where one fixes a (generally interpretable) functional form and infers its parameters, e.g. linear regression, and machine learning, where one searches in a vast space of (generally non-interpretable) overparametrized functions, e.g. deep learning. In symbolic regression, we search through a vast space of interpretable mathematical expressions to fit the data as well as possible: this is a difficult task which cannot simply be expressed as a minimization problem, at odds with the two other methods.

term can be obtained by summing the previous two terms). We show that our model not only succeeds in retrieving the correct relation better than Mathematica, but also learns to approximate complex relations which it cannot express.

Finally, in Chap. 12, we tackle the more challenging task of symbolic regression on multi-dimensional functions with non-integer coefficients. We show that our model is competitive with the most advanced genetic algorithms, while offering at least a ten-fold speedup: instead or retraining on each set of observations, our model leverages the experience gained during training.

A demonstration of our symbolic regression models can be found at the following address: `https://symbolicregression.metademolab.com/`.

Part I

A Theory of Overparametrization

Chapter 2

From the Jamming Transition to the Double Descent Curve

In this first chapter we will study the *interpolation threshold* (IT) of neural networks, i.e. the transition point from under-parametrized networks which have too few parameters to fit the training data to over-parametrized networks. By relating this threshold to the jamming transition of granular media, well-studied in the statistical physics litertature, we provide an upper bound for its location and show that in the whole over-parametrized regime, poor minima of the loss are not encountered during training because the number of constraints that hinders the dynamics is insufficient to allow for the emergence of stable minima.

We then study systematically how the IT affects the generalization properties of the network (i.e. predictive power). As we increase the number of parameters of a given model, starting from an under-parametrized network, we observe that the test error displays three phases: *(i)* initial decay, *(ii)* increase until the IT — where it displays a peak — and *(iii)* slow decay towards an asymptote as the network size diverges. This phenomenon was later coined the *double descent* curve and observed in a variety of machine learning models [32, 33].

Finally, we study the phenomenology of the double descent curve. First, the peak can be suppressed by using early stopping, showing that is due to overfitting. Second, if one ensembles the predictions of differently initialized networks, test error becomes roughly constant after the IT, showing that the improvement of generalization with overparametrization comes from reducing the variance due to the initialization of the parameters.

2.1 Introduction

A neural network is a high-dimensional function $f : \mathbb{R}^D \to \mathbb{R}^C$ that depends on a large number of parameters P. These parameters are learned so as to fit N training data points by minimizing some loss function $\mathcal{L} : \mathbb{R}^P \to \mathbb{R}$, generally via gradient descent (a noisy version of gradient descent). There is great flexibility in the network architecture, loss function and minimization protocol one can use. These features are ultimately selected to optimize the generalization performance on previously unseen data. Although the current progress in designing [34, 35] and training [36] networks that generalize well is undeniable, it remains mostly empirical. A general theory explaining and fostering this success is lacking, and central questions remain to be clarified, among which two puzzles stand out.

Interpolation threshold A first puzzle is the following: since the loss function is generally non-convex, when and how can the learning dynamics reach zero-loss solutions without getting stuck in local minima? Generally, one can expect that increasing the number of parameters P will increase the capacity of the model to memorize the training data, until a point called the *interpolation threshold* (IT), denoted as $P^\star$, where the data is perfectly memorized. Given a number of data N, what is the value of the IT $P^\star(N)$, and how does it depend on the depth of the network?

Generalization The second puzzle is the excellent generalization performance of heavily over-parametrized deep neural networks able to fit random labels [5]. Such *interpolating* estimators—that can reach zero training error— have attracted a growing amount of theoretical attention in the last few years, see e.g. [32, 37, 38]. Indeed, classical learning theory suggests they should generalize poorly due to overfitting [39], but they seem to avoid this trap, and their generalization performance improves relentlessly with their size [40].

2.1.1 Contributions

In the present work we tackle both puzzles in succession. We begin by studying the IT separating the (N, P) phase space into two regions: the over-parametrized regime, and the under-parametrized regime – see Fig. 2.1. By comparing it with the jamming transition which occurs in packings of particles, we characterize the critical line $P^\star(N)$ delimiting the two regions, yielding an upper-bound of the form $P^\star \propto N$ above which stable local minima do not exist. Empirically, we find that the exact location of the critical line is almost insensitive to the depth of the network, but crucially depends on the structure of the data. Just like in the jamming transition of particles, we observe that the number of non-memorized data jumps discontinuously from 0 to a large value at the IT.

Then, we show that the IT affects the most crucial aspect of learning, namely the test error. Indeed, generalization properties are strongly affected by the proximity to the IT: as we increase P, the test error first decreases, then displays a peak at $P^\star$, then decays monotonically. If early stopping is used, the peak disappears, implying that the IT is precisely the point where over-fitting is very strong.

Finally, we study that ensembling (averaging the predictions of networks with different random initialization of the weights) leads to a qualitatively very different curve, which becomes flat after the IT. This shows that the benefit of overparametrization beyond the IT stems from reducing the variance due to initialization of the weights.

Reproducibility The codes used to produce the results presented in this chapter are available at `https://github.com/mariogeiger/nn_jamming.git`.

2.1.2 Related work

Jamming transition and supervised learning An analogy has been established between the loss landscape of the perceptron (the simplest network, without any hidden layers) and the energy landscape of spherical particles [41]. The critical behavior of these granular systems, although very general, is of easier understanding when considering particles that interact only within a finite range: upon increasing their density, such systems undergo a *jamming transition* [42, 43] when there is no longer space to accommodate all the particles without them touching one another. Before the transition the energy is zero, and after it increases with the density. The inclusion of longer-range interactions blurs the transition[42].

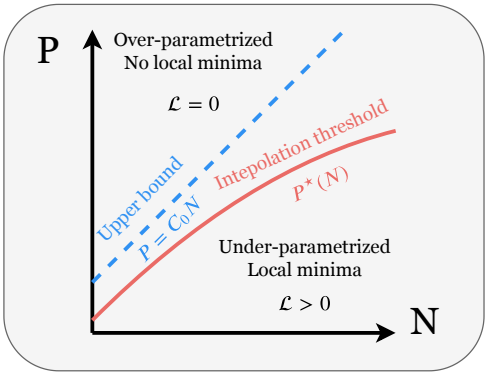


Figure 2.1: (P, N) phase space (P: number of parameters, N: number of training examples).

Deep networks behave similarly when we look at the training loss, and, again, a clear criticality emerges when considering a "finite-range" loss function — the hinge loss: when the number N of training points is small enough, the network is able to learn the whole training set and reaches zero training loss, but upon increasing N we find a critical "jamming" point where perfect training does not occur and learning gets stuck in a local minimum of the the training loss.

An analogy between deep learning and glasses has also been proposed [44, 45], in which the learning dynamics is expected to slow down and to get stuck in the highest minima of the loss. Yet, in the regime where the number of parameters is large (often considered in practice), several numerical and rigorous works [46–50] suggest a different landscape geometry where the loss function is characterized by a connected level set. Furthermore, studies of the Hessian of the loss function [51–53] and of the learning dynamics [54, 55] support that the landscape is characterized by an abundance of flat directions, even near its bottom, at odds with traditional glasses.

Effect of depth On one hand, it is often argued, and proved in some cases, that the advantage of deep networks stems from their enhanced expressive power, i.e. their ability to build complex functions with a much smaller number of parameters than needed for shallow networks [56–60]. Indeed if deep networks are able to fit data with less parameters, then they are likely to generalize better.

On the other hand, one can handcraft neural networks that fit even structure-less, random data with a rather small number of parameters $P \sim N$ [61–64]. These results for the static capacity of networks appear to be independent of depth [63, 64]. Yet, it is unclear whether such parsimonious solutions can be found dynamically in practice simply by descending the loss function, and whether depth can help finding them.

Generalization of overparametrized models The benefit of overparametrization has been the topic of study for several other works. Some of them focus on the effects of various ways of regularizing the network, thereby effectively reducing the dimension [65–68]. Yet another body of works focus on the effects of sheer size of a neural network [69–71].

The peak in the test error near the IT has previously been observed in several simple settings: least squares regression [72–76] and teacher-student regression with two-layer networks [37]. However, no evidence of such behavior has been observed in realistic tasks involving deep neural networks. Additionally, the generalization behavior of linear models is at odds with what we observe: for the

perceptron, test error asymptotically *increases* with P. As will be shown in detail in Chap. 4, despite strong resemblences, the overfitting peaks observed for linear models and neural networks are in fact of rather different nature.

Follow-ups of this work Since the initial preparation of the present work, the field progressed quickly within a matter of months. The described peak in the test error was observed empirically in [32] for a variety of other machine learning models such as random forests, and named "double descent". This phenomenon was also observed in [33] for state-of-the-art deep neural networks such as ResNets [77] and Transformers [78]. A line of theoretical studies on regression presented a precise mathematical description of double descent [79–82], albeit on models which are rather far from the practical setting of modern neural networks.

2.2 A warm-up with toy models

Before studying the two puzzles above in the context of deep learning, we gain some intuition by analyzing two very simple setups: 1-dimensional polynomial and high-dimensional linear least-squares regression.

2.2.1 Polynomial regression

Consider the setup of 1-dimensional polyomial regression, where the objective is to fit the best polynomial $\mathcal{P}$ given a set of observations $\{x_\mu, y_\mu\}_{\mu=1\dots N}$ by minimizing the squared error:

$$\mathcal{L} = \frac{1}{2N}\sum_{\mu=1}^{N}(\mathcal{P}(x_\mu) - y_\mu)^2 \tag{2.1}$$

Here, we can choose the number of degrees of freedom by adjusting the degree of the polynomial, which we will denote as P.

Interpolation threshold It is well known that when trying to fit a polyomial of degree P through N points, in the general case there will be no solutions if $P < N$, a single solution when $P = N$ and several solutions when $P > N$. Hence, we see that the interpolation threshold, i.e. the point at which the loss $\mathcal{L}$ vanishes, is located at $P = N$.

Generalization Although Lagrange polyomials guarantee the existence of an interpolating polynomial for $P = N$, the latter will generally have high *variance*: the slightest change in one of the points will completely change the form of the polynomial. Hence, such polynomials will generalize poorly to unseen data points: we are faced with *overfitting* at the IT. Now, what happens if we continue increasing the degree of the polynomial ?

Since there are several solutions when $P > N$, the answer depends on which solution we choose. If we choose at random, overfitting will typically become worse and worse as we increase the degree of the polynomial: the solutions will become more and more wiggly, i.e. have high variance. However, among all the solutions, the one with the *smallest* variance will typically become less and less wiggly. Imagine taking $P\to\infty$: by Taylor expansion, the solution space contains all possible (smooth) interpolating functions, including ones with very low variance!

Choosing the solution with smallest variance is the objective of regularization in machine learning. In this very simple example, it makes sense that in absence of regularization, generalization worsens beyond the IT, leading to a U-shaped curve, but with appropriate regularization, generalization can improve beyond the IT, leading to a double descent curve.

However, what is surprising in deep learning is that the double descent curve is observed even *in absence* of explicit regularization. We call this phenomenon *implicit regularization*, and it is generally believed to stem from the properties of the optimization algorithm (SGD), see [83] for a review.

2.2.2 Least-squares regression

Let us now move to high-dimensional linear regression, where objective is the infer a vector $\boldsymbol{W} \in \mathbb{R}^P$ from a set of observations $\{\boldsymbol{x}_\mu \in \mathbb{R}^P, y_\mu \in \mathbb{R}\}_{\mu=1\dots N}$, by minimizing the squared error:

$$\mathcal{L} = \frac{1}{2N}\sum_{\mu=1}^{N}(\boldsymbol{W}^\top \boldsymbol{x}_\mu - y_\mu)^2 \tag{2.2}$$

Again we have P degrees of freedom to fit N data points. We denote the matrix of inputs $\boldsymbol{X} \in \mathbb{R}^{P\times N}$, its covariance $\boldsymbol{\Sigma} = \frac{1}{N}\sum_\mu \boldsymbol{x}_\mu^\top \boldsymbol{x}_\mu$, and the vector of targets $(\boldsymbol{Y})_\mu = y_\mu$.

Interpolation threshold In this setup, the interpolation threshold is clearly located at $P = N$. Indeed, if $P = N$, $\boldsymbol{X}$ is a square matrix, which is invertible with high probability. There is a unique solution ensuring $\mathcal{L} = 0$, given by solving the linear system $\boldsymbol{Y} = \boldsymbol{W}\boldsymbol{X}$: the solution is $\boldsymbol{W}^\star = \boldsymbol{X}^{-1}\boldsymbol{Y}$. If $N > P$, then we do not have enough degrees of freedom and the loss will generally be nonzero. Conversely, if $P > N$, then there are multiple solutions ensuring $\mathcal{L} = 0$, just like in the setup of polynomial regression.

Generalization The generalization of least squares regression is a rather complex random matrix problem which has been studied in great detail in a variety of settings, see e.g. [79]. However, it is rather easy to understand why overfitting occurs at the IT $N = P$ by inspecting the expression for the optimal estimator:

$$\boldsymbol{W}^\star = \boldsymbol{X}^+ \boldsymbol{y}, \tag{2.3}$$

where $\boldsymbol{X}^+$ stands for the Moore-Penrose inverse of $\boldsymbol{X}$, i.e. $\boldsymbol{X}^+ = \lim_{\lambda\to 0}(\boldsymbol{X}\boldsymbol{X}^\top + \lambda \boldsymbol{I})^{-1}\boldsymbol{X}^\top$. Importantly, the eigenvalues of this matrix scale inversely with the eigenvalues of the covariance $\boldsymbol{\Sigma} = \frac{1}{N}\boldsymbol{X}\boldsymbol{X}^\top$.

Let us imagine that the inputs $\boldsymbol{x}_\mu$ are sampled from a random gaussian distribution $\mathcal{N}(0, \sigma)$. Define the distribution of eigenvalues of the covariance matrix $\boldsymbol{\Sigma}$ as $\rho(\mu) = \sum_{i=1}^{P}\delta(\mu - \mu_i)$, where $\{\mu_i\}_{i=1\dots P}$ are the eigenvalues of $\boldsymbol{\Sigma}$.

Now consider the high dimensional limit where $N\to\infty, P\to\infty$ with their ratio $\lambda = N/P$ fixed. Then the eigenvalue distribution becomes a continuous distribution called *spectral density*, and is given the Marcenko-Pastur law [84]:

$$\rho(\mu) = \frac{\lambda}{2\pi\sigma^2}\frac{\sqrt{(x_+ - x)(x - x_-)}}{x}\mathbf{1}_{x\in[x_-,x_+]}, \tag{2.4}$$

$$x_\pm = \sigma^2(1 \pm \sqrt{\frac{1}{\lambda}})^2 \tag{2.5}$$

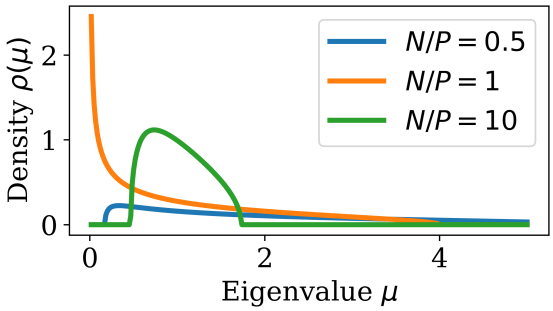


Figure 2.2: **The Marcenko-Pastur distribution.**

The shape of this distribution strongly depends on the parameter λ. Importantly, we see in Fig. 2.2 that the support of the spectral density reaches 0 when $P = N$, as $x_- \to_{\lambda \to 1} 0^+$, and in fact diverges near 0. This causes very small eigenvalues to appear in the spectrum of the covariance, the inverse of which which will lead the norm of $\boldsymbol{W}^\star$ to become very large and sensitive to the sampling of $\boldsymbol{X}$ according to Eq. 2.3. As before, the variance of the estimator becomes very large: the slightest noise in the input data will be severely overfit, and lead to catastrophic generalization on unseen data.

Note that the observation that square matrices tend to be poorly conditioned (i.e. display small singular values) compared to skinny matrices is not specific to the random gaussian setup considered here: it is a rather general phenomenon, which causes overfitting peaks to appear in a variety of contexts [85], as we will see in Chap. 4.

2.3 The interpolation threshold of deep neural networks

In this section we present the analogy between jamming and supervised learning for deep neural networks [41]. For the full analogy we point to the aforementioned paper [86]. This will set the stage for the analysis of the IT and its impact on generalization.

2.3.1 Set-up

We consider a binary classification problem, with a set of N distinct training data denoted $\{(\boldsymbol{x}_\mu, y_\mu)\}_{\mu=1}^N$. The vector $\boldsymbol{x}_\mu$ is the input, which lives in a D-dimensional space, and $y_\mu = \pm 1$ is its label. We denote by $f_{\boldsymbol{W}}(\boldsymbol{x})$ the output of a fully-connected network corresponding to an input $\boldsymbol{x}$, parametrized by $\boldsymbol{W}$. We represent the network as in Fig. 2.3, and the output function is written recursively as

$$f_{\boldsymbol{W}}(\boldsymbol{x}) \equiv a^{(L+1)}, \tag{2.6}$$

$$a_\beta^{(i)} = \sum_\alpha W_{\alpha,\beta}^{(i)} \, \sigma\left(a_\alpha^{(i-1)}\right) + B_\beta^{(i)}, \tag{2.7}$$

$$a_\beta^{(1)} = \sum_\alpha W_{\alpha,\beta}^{(1)} \, x_\alpha + B_\beta^{(1)}, \tag{2.8}$$

where $a_\alpha^{(i)}$ are the preactivations. In our notation the set of parameters $\boldsymbol{W}$ includes, with a slight abuse of notation, both the weights $W_{\alpha,\beta}^{(i)}$ and the biases $B_\alpha^{(i)}$. $\sigma(z)$ is a non-linear activation function, e.g. the

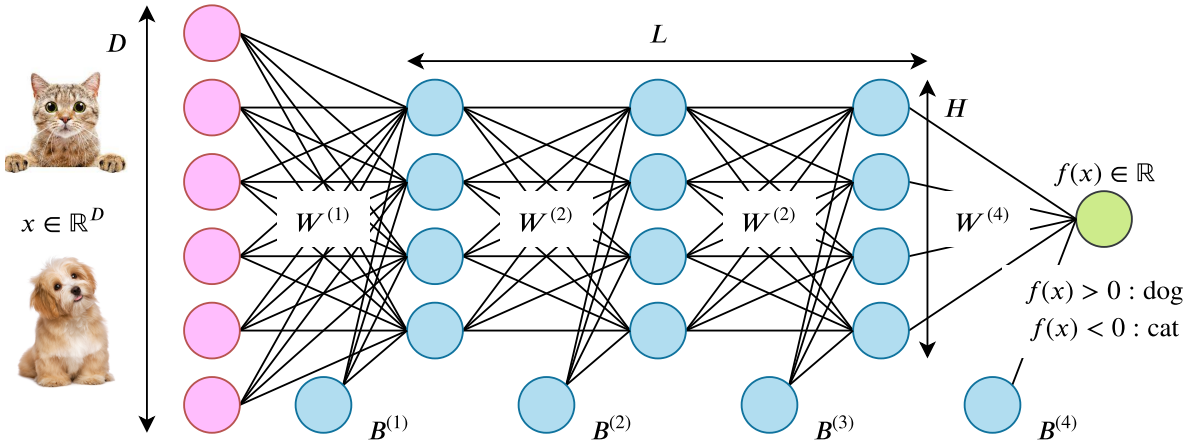


Figure 2.3: Architecture of a fully-connected network with L hidden layers of constant size h. Points indicate neurons, connections between them are characterized by a weight. Biases are not represented here.

ReLU $\sigma(z) = \max(0, z)$ or the hyperbolic tangent $\sigma(z) = \tanh(z)$. In this chapter, the parameters are learned by minimizing the quadratic hinge loss:

$$\mathcal{L}(\boldsymbol{W}) = \frac{1}{N}\sum_{\mu=1}^{N}\frac{1}{2}\max\left(0, \Delta_\mu\right)^2 \equiv \frac{1}{N}\sum_{\mu\in m}\frac{1}{2}\Delta_\mu^2, \tag{2.9}$$

where $\Delta_\mu \equiv 1 - y_\mu f_{\boldsymbol{W}}(\boldsymbol{x}_\mu)$ and m is the set of patterns with $\Delta_\mu > 0$ and contains N_Δ elements. These patterns describe *unsatisfied constraints*: they are either incorrectly classified or classified with an insufficient margin (whereas patterns with $\Delta_\mu < 0$ are learned with margin 1). We adopt this loss function since it makes the IT simpler to analyze[1], but this choice does not influence the performance of the network, as we have reported in [86].

We are interested in the transition between the under-parametrized phase where the network cannot satisfy all the constraints ($\mathcal{L} > 0$) and an over-parametrized regime where all constraints are unsatisfied ($\mathcal{L} = 0$) as we increase the number of parameters P. Note that as discussed in App. A.1.1, the effective number of degrees of freedom P_{eff} of the network may be smaller than P. In the following analysis, we disregard this: as reported in A.1.1, for a reasonable initialization of the weights and constant-width fully connected networks, $P_{\text{eff}} \approx P$ (the difference is small and equal to the number of hidden neurons, and only results from the symmetry associated with each ReLU neuron). Henceforth to simplify notations we will use the symbol P to represent the effective number of parameters.

2.3.2 Constraints on the stability of minima

Let us suppose (this will be justified later) that, for a fixed number of data N and with proper initialization of weights, if P is large enough then gradient descent leads to $\mathcal{L} = 0$, whereas if N is small after training

[1]The often used cross-entropy loss function also displays a point where all data are well-fitted. However, in the over-parametrized regime the dynamic never stops, as the total loss vanishes only if the output and therefore the weights diverge.

$\mathcal{L} > 0$. Imagine increasing P starting from a small value: at some $P^\star$ the loss obtained after training approaches zero [2], i.e. $\lim_{P\to P^\star} \mathcal{L} = 0$. We refer to this point as the IT. A vanishing training loss implies that $\Delta_\mu \to 0$ for each pattern $\mu = 1, \dots, N$.

As argued in [87], for each $\mu \in m$ the constraint $\Delta_\mu \approx 0$ defines a manifold of dimension $P - 1$[3]. Satisfying N_Δ such equations thus generically leads to a manifold of solutions of dimension $P - N_\Delta$[4]. Imposing that a solution exists implies that at the IT:

$$P^\star \geq N_\Delta. \tag{2.10}$$

An opposite bound can be obtained by considerations of stability (as was done for the jamming of repulsive spheres in [89]), by imposing that in a stable minimum the Hessian must be positive definite if the activation function is smooth (see below for the situation where the activation function displays cusps, as occurs for ReLU neurons). The Hessian matrix, i.e. the matrix of second derivatives of the loss with respect to the parameters, is given by

$$\begin{aligned} \mathrm{Hess}(\mathcal{L}) \equiv \mathcal{H} &= \tfrac{1}{N}\textstyle\sum_{\mu\in m} \nabla\Delta_\mu \otimes \nabla\Delta_\mu + \tfrac{1}{N}\sum_{\mu\in m} \Delta_\mu \nabla \otimes \nabla\Delta_\mu \\ &\equiv \mathcal{H}_0 + \mathcal{H}_p, \end{aligned} \tag{2.11}$$

where ∇ is the gradient operator and $\otimes$ stands for tensor product. The first term $\mathcal{H}_0$ is positive semi-definite: it is the sum of N_Δ rank-one matrices, thus $\mathrm{rk}(\mathcal{H}_0) \leq N_\Delta$, implying that the kernel of $\mathcal{H}_0$ is at least of dimension $P - N_\Delta$.

Let us denote by E_- the negative eigenspace[5] of $\mathcal{H}_p$ and call P_- its dimension. Stability imposes that $\ker(\mathcal{H}_0) \cap E_- = \{0\}$, which is only possible if $N_\Delta \geq P_-$. Hence, minima with positive training loss can only occur for:

$$N \geq N_\Delta \geq P_-, \tag{2.12}$$

where the first inequality trivially follows from the fact that the N_Δ patterns belong to the training set of size N). As reported in [86], we observe empirically that the spectrum of $\mathcal{H}_p$ is statistically symmetric in the cases that we consider in the present work, i.e. for ReLU activation function, both for MNIST and random data, both at initialization and at the end of training. In A.1.2 we provide a non-rigorous argument supporting that in the case of ReLU activation functions and random data the spectrum of $\mathcal{H}_p$ is indeed symmetric with $\lim_{P\to\infty} P_-/P_+ = 1$ independently of depth, where P_+ is the number of positive eigenvalue. We conjecture that in general the limiting spectrum of H_p as $P, N \to \infty$ (for any fixed ratio N/P) has a finite fraction $C_0 = P_-/P$ of negative eigenvalues for generic architectures and datasets. In [86] we observed $C_0 = 1/2$ for the ReLU activation function as expected, for tanh activation functions at jamming and at the end of training we found $C_0 \approx 0.43$. Thus, C_0 is not universal[6].

[2] For finite P, $P^\star$ will present fluctuations induced by differences of initial conditions. The fluctuations of $P/P^\star$ are however expected to vanish in the limit where P and $P^\star$ become large. This phenomenon is well-known for the jamming of particles, and is an instance of finite size effects.

[3] Related arguments were recently made for a quadratic loss [50]. In that case, we expect the landscape to be related to that of floppy spring networks, whose spectra are predicted in [88].

[4] Note that this argument implicitly assumes that the N_Δ constraints are independent. In disordered systems this assumption is generally correct, but it may break down if symmetries are present.

[5] The negative eigenspace is the subspace spanned by the eigenvectors associated with negative eigenvalues.

[6] The conjecture does not hold in some pathological cases, for instance if two data points can be identical with different labels (a case we exclude here). Indeed even in the over-parametrized case, if $\boldsymbol{x}_\mu = \boldsymbol{x}_\nu$ but $y_\mu \neq y_\nu$ then an exact cancellation of terms occurs in the sum defining $\mathcal{H}_p$, which can then be zero while $\mathcal{L} > 0$, leading to $C_0 = 0$. Further details are discussed in A.1.3

Finally we assume that the spectrum of $\mathcal{H}_p$ does not display a finite density of zero eigenvalues (once restricted to the space of parameters that affect the output, of dimension P_{eff}, supposed here to be equal to N). Under this assumption we obtain from Equation (2.12) that local minima with positive training loss cannot be encountered if $P > N/C_0$, implying in particular that:

$$P^\star \leq N/C_0. \tag{2.13}$$

Hence sufficiently overparametrized networks reach a flat global minimum with zero training loss. Note that this does not imply that all such global minima must have also the same test loss.

Non-smooth activation functions: With ReLU activation functions, the output function $f_{\boldsymbol{W}}(\boldsymbol{x})$ is not smooth and presents cusps, so that the Hessian needs not be positive definite for stability. A minimum can lie on a point where the second derivative is not defined along some directions (because of the cusp), and we say that the cusp stabilizes those directions. Equation (2.12) needs to be modified accordingly: introducing the number of directions $P_c \equiv \beta P$ presenting cusps near jamming, stability implies $N_\Delta > P_- - P_c$ and:

$$N_\Delta \geq P(C_0 - \beta) \tag{2.14}$$

implying in turn that:

$$P^\star(N) \leq \frac{N}{C_0 - \beta} \tag{2.15}$$

Numerically, we find that at jamming the fraction of directions along which there is a cusp is $P_c/P^\star \equiv \beta \in (0.21, 0.25)$ both for random data and images as reported in the A.1.4. Using $C_0 = 1/2$ for ReLU, we obtain the bounds:

$$N_\Delta \geq P/4 \tag{2.16}$$

$$P^\star \leq 4N. \tag{2.17}$$

2.3.3 Numerical validation

Here we present numerical results on random data (uniformly distributed on a hypersphere of dimension D and with random labels $y_\mu = \pm 1$) and on the MNIST dataset of handwritten digits (partitioned into two groups according to the parity of the digits, with labels $y_\mu = \pm 1$). With MNIST, in order not to have most of the weights in the first layer, we reduce the actual input size by retaining only the first $D = 10$ principal components that carry the most variance (this hardly diminishes the performance for such a task). Further description of the protocols is given in A.2.

In Fig. 2.4A,C we show the location of boundary $P^\star$ versus the number of samples P. $P^\star$ is estimated numerically for each P by starting from a large value of P and progressively decreasing it until $\mathcal{L} > 0$ at the end of training. Varying input dimension, depth and loss function (cross entropy or hinge) has little effect on the IT. This result indicates that in the present setup the ability of fully-connected networks to fit random data does not depend crucially on depth. Fig. 2.4C shows also a comparison of random data with MNIST. A difference between random data and images is that the minimum number of parameters $P^\star$ needed to fit the real data is significantly smaller and grows less fast as P increases — for $N \gg 1$, $P^\star(N)$ could be sub-linear or even tend to a finite asymptote, as the new data becomes increasingly redundant: how the structure in the data and in the labels affects $P^\star(N)$ is discussed in Chap. 5.

Interestingly, the IT shares an important phenomenon with the jamming transition of particles. When one increases the particle density, the number of contacts suddenly jumps from zero to a large

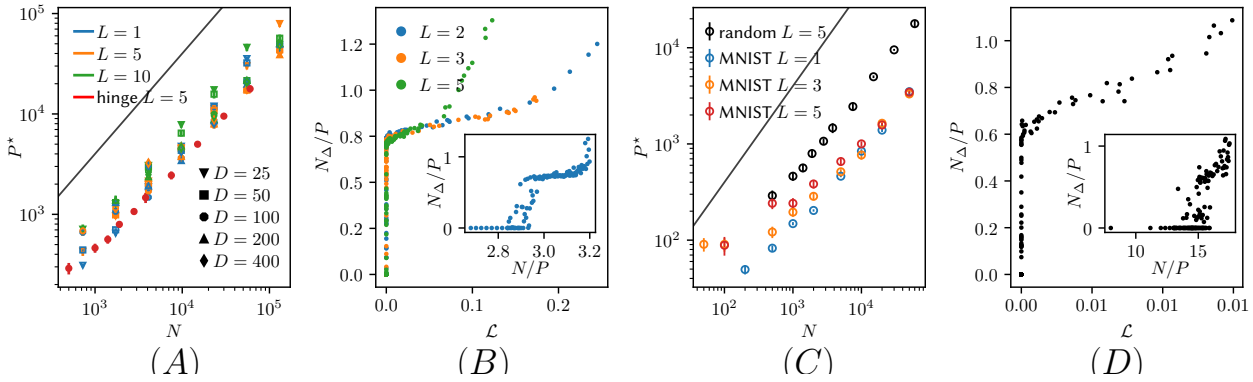


Figure 2.4: **The empirical interpolation threshold is below the theoretical upper bound $P^\star = 4N$, and depends on the structure of data but not on the depth or input dimension**. *(A) and (B):* random data in various dimensions D. *(C) and (D):* MNIST dataset. *(A) and (C)* depict the location $P^\star$ (that is, the number of parameters) of the IT as a function of the number N of training points, for networks of various depths L. *(B) and (D)* show that in the ($\mathcal{L}$-N_Δ/P) and (N/P-N_Δ/P) planes the jamming transition displays a discontinuous jump.

number, proportional to the number of particles. In the experiments above, we observe something very similar: the number of constraints per parameter N_Δ/P jumps discontinuously at the IT, as shown in the insets of Fig. 2.4B,D. The scatter in these plots presumably reflects finite size effects known to occur near the jamming transition of particles [43]. All this scatter is however gone when plotting N_Δ/P as a function of the loss itself, as shown in the main panels of Fig. 2.4B,D.

Summary So far, our analysis supports that for smooth activation functions there exists a constant C_0 such that the IT occurs for $P^\star(N) \leq N/C_0$. We also see that at the IT, the fraction N_Δ/P of unsatisfied constraints per degree of freedom jumps discontinuously to a finite value satisfying $C_0 \leq N_\Delta/P \leq 1$.

For non-smooth activation functions such as ReLU, but the analysis is complicated by the presence of cusps. However, for ReLU we still find $C_0 = 1/2$ and the transition is sharp, i.e. characterized by a discontinuous jump in constraints as specified by Eq. 2.16.

The complete list of results is included in [86], and shows that that the IT of deep neural networks is akin the the jamming transition of *elliptical* (rather than spherical) particles.

2.4 The double descent curve

In this section, we study the impact of the IT on test error, defined as follows:

$$\epsilon_g(\boldsymbol{W}) = \frac{1}{N_{\text{test}}} \sum_{\mu=1}^{N_{\text{test}}} \mathbb{1}_{y_\mu = f_{\boldsymbol{W}}(\boldsymbol{x}_\mu)} \tag{2.18}$$

We present the first evidence of a double descent curve in deep learning.

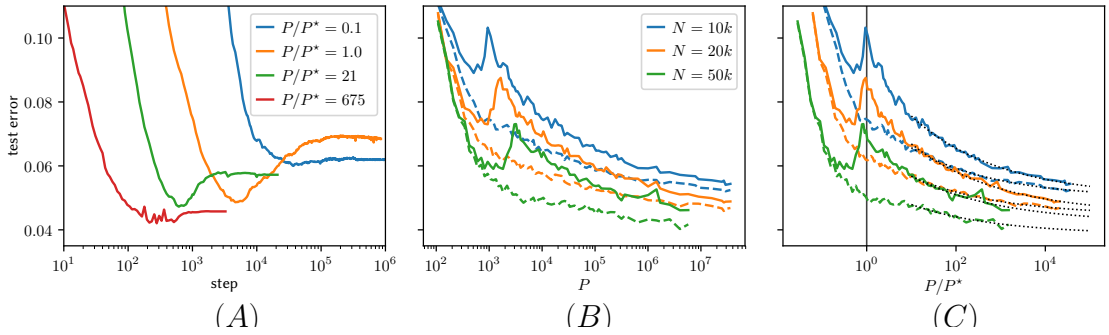


Figure 2.5: **The double descent curve: generalization performance improves with over-parametrization, except near the interpolation threshold where it displays a peak**. We trained a 5 hidden layer fully-connected network on MNIST. *(A)* Typical evolution of the test error over training time, for systems located at different points relatively to the IT (for $N = 50k$): over-fitting is marked by the gap between the value at the end of training and the minimum at prior times. Notice that training of over-parametrized systems halts sooner because the networks have achieved zero loss over the training set. *(B)* Test error at the final point of training (solid line) and minimum error achieved during training (dashed line) vs. system size. *(C)* When N is scaled by $P^\star(N)$ it is clear that over-fitting occurs at the IT.

.

2.4.1 Generalization at and beyond the interpolation threshold

In Fig. 2.5A we show the evolution of the test error for networks at four different locations in the (N, P) plane. The networks are trained on MNIST at fixed $N = 50k$, and at different values N, both above, at and below jamming. Training is run for a fixed number of steps of vanilla gradient descent (the simulation details are in A.2). The profile of these curves is typical of most learning problems (if one does not resort to early stopping): notice that the point of minimum test error happens before the end of training. The increase of test error at late times is referred to as "over-fitting" in the field. Very interestingly, it is clear from this figure that at small and large N, over-fitting is a weak effect, which however becomes very significant at intermediate N.

To study this effect, we systematically vary P at fixed N. In Fig. 2.5B the solid curve shows the test error against the network size P for three different values of N (we sampled subsets of MNIST). The dashed curve represents the value of the smallest error obtained during training, at prior time-steps (extracted from the profiles shown in Fig. 2.5A). The former displays a peak at the IT $P^\star$, as one can see clearly after rescaling the N-axis of each curve by the corresponding value of $P^\star(N)$.

Strong over-fitting, corresponding to the difference between the solid and dashed lines, takes place only in the vicinity of the IT (Fig. 2.5B-C). We thus posit that at fixed N, the benefit of early stopping [66] should diminish in the large-size limit. Beyond the IT point, the accuracy keeps steadily improving as the number of parameters increases [69–71], although it does so quite slowly[7]. To understand the origin of this improvement, we study the impact of ensembling in what follows.

[7] In A.2.2 we have verified that the overall trends described here hold qualitatively for other depths.

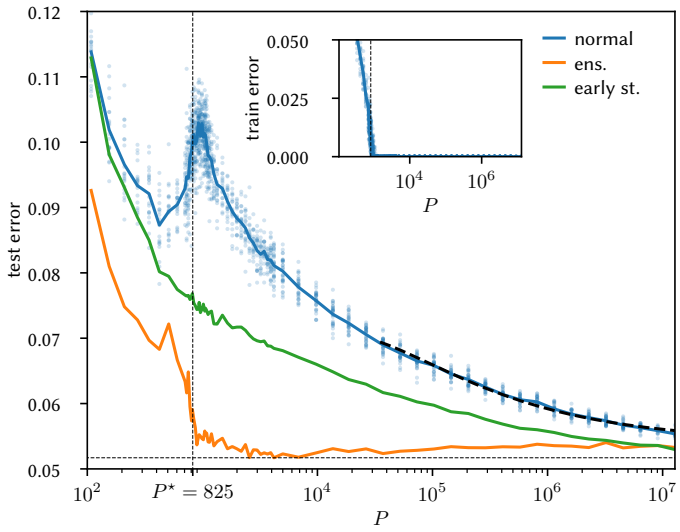


Figure 2.6: **In presence of ensembling, the overfitting peak vanishes and optimal generalization occurs just after the interpolation threshold.** Empirical test error (main figure) and train error (inset) *v.s.* number of parameters: average curve (blue, averaged over 20 runs); early stopping (green); ensemble average (orange) over 20 independently initialized networks. The vertical dashed line corresponds to the IT: at that point the test error peaks. Ensemble averaging leads to an essentially constant behavior when P becomes larger than $P^\star$.

2.4.2 Effect of ensembling

Ensembling amounts to training several different models, then averaging their predictions. The various models of the ensemble can differ in many different ways: architecture, training hyperparameters, weight initialization, etc. Here we consider the latter scenario: we investigate the effet of averaging the predictions of 20 fully-connected networks with different weight initializations on the double descent curve described above.

Our results are shown in Fig.2.6. As before, we observe a double descent curve (in blue) when only considering the predictions of a single model, and the peak near the IT vanishes when performing early stopping. Remarkably, in our experiments ensemble-averaging led to a nearly flat test error for $P > P^\star$; this supports that the improvement of generalization performance with N in this classification task originates from reduced variance of f_P when N gets large, as recently observed for mean-square regression [38].

An observation of potential practical interest is that near-optimal generalization is obtained by ensemble averaging slightly above $P^\star$. Thus the intuition that the most predictive and parsimonious models have just enough parameters to fit the data may indeed be correct, once one averages over differently initialized networks[8].

[8]This observation carries over to convolutional networks, as well. We train CIFAR10 on a vanilla architecture with 3 convolutional layers with f filters at each layer and a single fully-connected layer. For each f, we train 20 models at different random initial conditions. Just after $P^\star$, the mean accuracy is $\sim$ %66, accuracy of the ensemble averaging is $\sim$ %80, and the average accuracy of widest models we could train (which has 5 orders of parameters more) is a little bit less than $\sim$ %77.

2.5 Conclusion

In this chapter, we recast the minimization of the loss function of neural networks as a constraint-satisfaction problem, enabling us to predict an upper bound for the IT of the form $P^\star \leq N/C_0$, where the specific value of C_0 depends on the activation function, but not on the depth of the network. In practice, for random data $P^\star(N)$ scales linearly with P (in this case, the bound is tight), but sublinearly for structured data with non-random labels. Obtaining a tighter, data-dependent upper bound for the curve $P^\star(N)$ remains an important challenge for future work.

Finally, we related the IT to the generalization of the model, presenting the first reported evidence of a double descent curve in deep learning: generalization improves with overparametrization except near the IT, where a peak appears. By studying the impact of early stopping and ensembling, we were able to show that the peak is related to overfitting, whereas the decay of test error with overparametrization is comes from a reducting of the variance due to the initialization of the weights. Establishing this from a quantitative point-of-view will be the objective of the following chapter.

Chapter 3

Double Trouble in Double Descent: Bias and Variance(s) in the Lazy Regime

Deep neural networks can achieve remarkable generalization performances while interpolating the training data; rather than the U-curve emblematic of the bias-variance trade-off, their test error often follows a “double descent" curve — a mark of the beneficial role of overparametrization.

In this chapter, we develop a quantitative theory for this phenomenon in the context of high-dimensional random features regression. We obtain a precise asymptotic expression for the bias-variance decomposition of the test error, and show that the bias displays a phase transition at the IT, beyond it which it remains constant. We disentangle the variances stemming from the sampling of the dataset, from the additive noise corrupting the labels, and from the initialization of the weights.

Following up on the results of the previous Chapter, we demonstrate that the latter two contributions are the crux of the double descent curve: they lead to the overfitting peak at the IT and to the decay of the test error upon overparametrization. We quantify how they are suppressed by averaging the outputs of independently initialized estimators, and compare this ensembling procedure with overparametrization and regularization. Finally, we present numerical experiments on a standard deep learning setup to show that our results are relevant to the lazy regime of deep neural networks.

3.1 Introduction

As we have seen above, the generalization performance of deep neural networks improves relentlessly as they become larger. The reasons behind the performance of deep neural networks in the over-parametrized regime are still poorly understood, even though some mechanisms are known to play an important role, such as the implicit regularization of stochastic gradient descent which allows to converge to the minimum norm solution, and the convergence to mean-field limits [37, 90–93].

Optimization plays an important role in neural networks by inducing implicit regularization [69] and fluctuations of the learnt estimator [94]. Disentangling the variance stemming from the randomness of the optimization process from that the variance due to the randomness of the dataset is a crucial step towards a unified picture, as suggested in [38]. In this chapter, we address this issue and attempt to reconcile the behavior of bias and variance with the double descent phenomenon by providing a quantitative theory for random feature model (RF), introduced by [95]. The latter that can be viewed either as a randomized approximation to kernel ridge regression, or as a two-layer neural network

whose first layer contains fixed random weights.

The latter provides a simple model for deep learning in the *lazy* regime [91] where the weights stay close to their initial value during training – as opposed to *feature learning* regime where the weights change enough to learn relevant features [96, 97]. . Indeed, suppose that a neural network learns a function $f_{\boldsymbol{W}}(\boldsymbol{x})$ that relates labels (or responses) y_i to inputs $\boldsymbol{x}_i$ with, $i = 1, \dots, N$ via a set of weights $\boldsymbol{W}$. The lazy regime is defined as the setting where the model can be linearized around the initial conditions $\boldsymbol{W}_0$. Assuming that the initialization is such that $f_{\boldsymbol{W}_0} \approx 0$[1], one obtains:

$$f_{\boldsymbol{W}}(\boldsymbol{x}) \approx \nabla_{\boldsymbol{W}} f_{\boldsymbol{W}}(\boldsymbol{x})|_{\boldsymbol{W}=\boldsymbol{W}_0} \cdot (\boldsymbol{W} - \boldsymbol{W}_0)\,. \tag{3.1}$$

In other words, the lazy regime corresponds to a linear fitting problem with a random feature vector $\nabla_{\boldsymbol{W}} f_{\boldsymbol{W}}(\boldsymbol{x})|_{\boldsymbol{W}=\boldsymbol{W}_0}$.

Although replacing learnt features by random features may appear as a crude simplification, empirical results show that the loss in performance can be rather small in some cases [92, 98]. A burst of recent papers showed that in this regime, neural networks behave like kernel methods [99–101] or equivalently random projection methods [91, 92, 95]. This mapping makes the training analytically tractable, allowing, for example, to prove convergence to zero error solutions in overparametrized settings [102–108].

3.1.1 Contributions

In the setup described above, our contributions are:

- We demonstrate how to disentangle quantitatively the contributions to the test error of the bias and the various sources of variance of the estimator, stemming from the sampling of the dataset, from the additive noise corrupting the labels, and from the initialization of the random feature vectors.
- We give a sharp asymptotic formula for the effect of ensembling on these various terms, and show that overfitting near the IT comes from the interplay between noise and initialization variance – hence the *double trouble*. These two terms decay beyond the IT, whereas the sampling variance and the bias reach a plateau at the IT, and remain constant in the overparametrized regime. Hence, the benefit of overparametrization beyond the IT is solely due to a reduction of the noise and initialization variances.
- We show in particular how the over-fitting peak at the IT can be attenuated by ensembling, as observed in real neural networks in the previous Chapter. Finally, we compare the variance-reduction effects of ensembling, overparametrization, regularization and bagging (a form of ensembling where the different models learn from different splits of the original dataset).

Reproducibility The code used to produce the results presented in this chapter are available at `https://github.com/mariaref/Random_Features.git`.

[1] One can alteratively define the estimator as $f_{\boldsymbol{W}} - f_{\boldsymbol{W}_0}$ [91].

3.1.2 Related work

Replica method The analytical results we present are obtained using a heuristic method from Statistical Physics called the Replica Method [109], which despite being non-rigorous has shown its remarkable efficacy in many machine learning problems [76, 110–112] and random matrix topics, see e.g. [113–115]. See [116] and [117] for recent reviews. While it is an open problem to provide a rigorous proof of our computations, we check through numerical simulations that our asymptotic predictions are extremely accurate at moderately small sizes.

Analytic works on overparametrization On the theoretical side, our paper builds on the results of [82], which provide an analytic expression of the test error of the RF model in the high-dimensional limit where the number of random features, the dimension of the input data and the number of data points are sent to infinity with their relative ratios fixed. The double descent was also studied analytically for various types of linear models, both for regression [37, 79, 80, 118, 119] and classification [120–122].

Bias and variance(s) In the usual formulation of the bias-variance trade-off in machine learning, the variance stems from the random sampling of the data points. Indeed, for simple machine learning models (think of linear regression, logistic regression, support vector machines), there is little to no dependency on the initialization of the model parameters. However, in deep learning, the initialization of the weights is crucial, as shown in the previous chapter [94]. This was also noticed in [38], where the authors empirically disentangles various sources of variance and the bias in the process of training deep neural networks. Just like us, they find that the initialization variance reaches a peak then decreases in the overparametrized regime, and dominates over the sampling variance.

Follow-ups of this work Since its initial publication, this work has sparked many follow-ups. A few works use rigorous methods to confirm our results [123, 124]. Using the toolbox of statistical physics, [125] extended our results to classification tasks. The authors of [126] present a *symmetric* decomposition, in contrast with our *sequential* decomposition where the noise variance is averaged out first, followed by the initialization variance and the sampling variance. Finally, [127] goes further by obtaining a *fully disentangled* decomposition where all interaction terms are analyzed separately; their results suggest that the *interaction* between sampling variance and initialization variance can dominate the error in some cases.

3.2 Analytical results

In this section, we present our main result, which is an analytical expression for all terms appearing in the decomposition of the test error in terms of its bias and variance components.

3.2.1 Setup

This chapter and the subsequent ones are centered around the RF model first introduced in [95]. Although simpler settings such as linear regression display the double descent phenomenology [79], this model is more appealing in several ways. First, the presence of two layers allows to freely disentangle the dimensionality of the input data from the number of parameters of the model. Second, it closely relates to

the lazy learning, as described above. Third, and most importantly for our specific study, the randomness of the first layer weights enables to study the impact of ensembling.

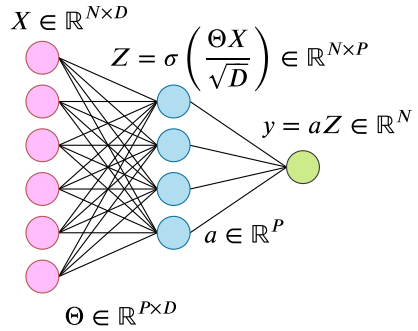


Figure 3.1: **A random feature network.**

The RF model can be viewed as a two-layer neural network whose first layer contains fixed random weights[2] (see Fig. 3.1):

$$\hat{f}(\boldsymbol{x}) = \sum_{i=1}^{P} \boldsymbol{a}_i \sigma\left(\frac{\langle \boldsymbol{\theta}_i, \boldsymbol{x}\rangle}{\sqrt{D}}\right). \tag{3.2}$$

In the above, $\boldsymbol{\theta}_i$ is the i^{th} *random feature*, i.e the i^{th} column of the random feature matrix $\boldsymbol{\Theta} \in \mathbb{R}^{P\times D}$ whose elements are drawn i.i.d from $\mathcal{N}(0,1)$. σ is a pointwise activation function, which we will take to be $\mathrm{ReLU} : x \mapsto \max(0, x)$ in this chapter. Note however that our computation is valid for any activation function, and the impact of the latter will be the object of the next chapter.

The training data is collected in a matrix $\boldsymbol{X} \in \mathbb{R}^{N\times D}$ whose elements are drawn i.i.d from $\mathcal{N}(0,1)$[3]. We assume that the labels are given by a linear ground truth corrupted by some additive Gaussian noise:

$$\begin{aligned} &y_\mu = \langle \boldsymbol{\beta}, \boldsymbol{X}_\mu\rangle + \varepsilon_\mu, \quad ||\boldsymbol{\beta}|| = F, \quad \varepsilon_\mu \sim \mathcal{N}(0,\tau), \\ &\mathrm{SNR} = F/\tau. \end{aligned} \tag{3.3}$$

The generalization to non-linear functions can easily be performed as in [82]: the non-linear part simply amounts to an added noise term.

The second layer weights, i.e the elements of $\boldsymbol{a}$, are calculated by the means of ridge regression:

$$\begin{aligned} \mathcal{L}(\boldsymbol{a}) &\equiv \frac{1}{N}\sum_{\mu=1}^{N}\left(y_\mu - \sum_{i=1}^{P} \boldsymbol{a}_i \sigma\left(\frac{\langle \boldsymbol{\theta}_i, \boldsymbol{X}_\mu\rangle}{\sqrt{D}}\right)\right)^2 + \frac{P\lambda}{D}\|\boldsymbol{a}\|_2^2, \\ \hat{\boldsymbol{a}} &\equiv \underset{\boldsymbol{a}\in\mathbb{R}^P}{\arg\min}\, \mathcal{L}(\boldsymbol{a}). \end{aligned}$$

Note that as $P\to\infty$, the random feature regression considered here is equivalent to kernel ridge regression with respect to the following kernel:

$$K(\boldsymbol{x}, \boldsymbol{x}') = \mathbb{E}_{\boldsymbol{\Theta}\sim\mathcal{P}}\left[\sigma(\langle \boldsymbol{x}, \boldsymbol{\Theta}\rangle/\sqrt{D})\sigma\left(\langle \boldsymbol{x}', \boldsymbol{\Theta}\rangle/\sqrt{D}\right)\right],$$

where $\mathcal{P}$ is the uniform distribution over the $D-1$-dimensional sphere of radius $\sqrt{D}$.

The key quantity of interest is the *test error* of this model, defined as the mean square error evaluated on a fresh sample $\boldsymbol{x} \sim \mathcal{N}(0,1)$ corrupted by some target noise $\tilde{\varepsilon}$:

$$\epsilon_g = \mathbb{E}_x\left[\left(\langle \boldsymbol{\beta}, \boldsymbol{x}\rangle + \tilde{\varepsilon} - \hat{f}(\boldsymbol{x})\right)^2\right], \quad \tilde{\varepsilon} \sim \mathcal{N}(0,\tilde{\tau}). \tag{3.4}$$

[2] Note the closeness between the RF model and the "*hidden manifold model*" introduced in [128]. The task studied here can be seen as a linear regression task on a structured data set $\boldsymbol{Z} \in \mathbb{R}^P$, obtained by projecting the original *latent* features $\boldsymbol{X} \in \mathbb{R}^D$. The difference here is that the dimension of the latent space, denoted as D here, is sent to infinity together with the dimension of the ambient space.

[3] The impact of data structure will be studied in Chap. 5

3.2.2 Decomposition of the test error

In the standard bias-variance decomposition for machine learning, the "variance" term only describes the variability of the estimator with respect to the input data. This is because classic machine learning methods can often be solved with convex optimization methods. However, in the non-convex setup of deep learning, there is another source of randomness: the initialization of the weights. Taking the latter into account, the test error (3.4) can be decomposed into five terms:

$$\mathbb{E}_{\boldsymbol{\Theta},\boldsymbol{X},\varepsilon}\left[\epsilon_g\right] = \mathcal{E}_{\text{Noise}} + \mathcal{E}_{\text{Init}} + \mathcal{E}_{\text{Samp}} + \mathcal{E}_{\text{Bias}} + \tilde{\tau}^2. \tag{3.5}$$

The first three contribute to the variance, the fourth is the bias, and the final term $\tilde{\tau}^2$ is simply the error of an *oracle* predictor. It does not play any role and will be set to zero in the rest of the paper: the only reason it was included is to avoid confusion with $\mathcal{E}_{\text{Noise}}$ defined below.

Noise variance: The first term is the variance associated with the additive noise corrupting the labels of the *training* examples, $\boldsymbol{\varepsilon}$:

$$\mathcal{E}_{\text{Noise}} = \mathbb{E}_{\boldsymbol{x},\boldsymbol{X},\boldsymbol{\Theta}}\left[\mathbb{E}_{\varepsilon}\left[\hat{f}(\boldsymbol{x})^2\right] - \left(\mathbb{E}_{\varepsilon}\left[\hat{f}(\boldsymbol{x})\right]\right)^2\right]. \tag{3.6}$$

Initialization variance: The second term encodes the fluctuations stemming from the random initialization of the random feature vectors, $\boldsymbol{\Theta}$:

$$\mathcal{E}_{\text{Init}} = \mathbb{E}_{\boldsymbol{x},\boldsymbol{X}}\left[\mathbb{E}_{\boldsymbol{\Theta}}\left[\mathbb{E}_{\varepsilon}\left[\hat{f}(\boldsymbol{x})\right]^2\right] - \mathbb{E}_{\boldsymbol{\Theta},\varepsilon}\left[\hat{f}(\boldsymbol{x})\right]^2\right].$$

Sampling variance: The third term measures the fluctuations due to the sampling of the training data, $\boldsymbol{X}$:

$$\mathcal{E}_{\text{Samp}} = \mathbb{E}_{\boldsymbol{x}}\left[\mathbb{E}_{\boldsymbol{X}}\left[\mathbb{E}_{\boldsymbol{\Theta},\varepsilon}\left[\hat{f}(\boldsymbol{x})\right]^2\right] - \mathbb{E}_{\boldsymbol{X},\boldsymbol{\Theta},\varepsilon}\left[\hat{f}(\boldsymbol{x})\right]^2\right]. \tag{3.7}$$

Bias: Finally, the fourth term in (3.5) is the bias, i.e. the error that remains once all the sources of variance have been averaged out. It can be understood as the approximation error of our model and takes the form:

$$\mathcal{E}_{\text{Bias}} = \mathbb{E}_{\boldsymbol{x}}\left[\left(\langle\boldsymbol{\beta},\boldsymbol{x}\rangle - \mathbb{E}_{\boldsymbol{X},\boldsymbol{\Theta},\varepsilon}\left[\hat{f}(\boldsymbol{x})\right]\right)^2\right]. \tag{3.8}$$

Note that this decomposition is sequential: we first remove the noise variance, then remove the initialization variance from the noise averaged predictor, and finally remove the residual sampling variance from the noise and initialization averaged predictor. This order was chosen for simplicity, and to enable comparison with existing bias-variance decompositions [38, 82].

Other sources of variance By performing ridge regression, we are missing out on two sources of variance which could be incurred by SGD dynamics. First, the noise in SGD creates an extra source of variance. Second, even noiseless GD would add an extra contribution to initialization variance. Indeed, in the overparametrized regime, the ridge regression problem is underdetermined: there is a frozen part of the estimator which cannot be learnt [37]. This part, which is set to zero in ridge regression via the pseudo-inverse ($\lambda \to 0^+$) or via explicit regularization ($\lambda > 0$), adds an extra (harmful) dependency on initialization in GD.

3.2.3 Main result

Consider the high-dimensional limit where the input dimension D, the hidden layer dimension P (which is equal to the number of parameter in our model) and the number of training points N go to infinity with their ratios fixed:

$$N, P, D \to \infty, \quad \frac{P}{D} = \mathcal{O}(1), \quad \frac{N}{D} = \mathcal{O}(1). \tag{3.9}$$

We obtain the following result:

$$\mathbb{E}_{\boldsymbol{x},\boldsymbol{\varepsilon},\boldsymbol{\Theta},\boldsymbol{X}} \left[\langle \boldsymbol{\beta}, \boldsymbol{x} \rangle \hat{f}(\boldsymbol{x}) \right] = F^2 \Psi_1, \tag{3.10}$$

$$\mathbb{E}_{\boldsymbol{x},\boldsymbol{\Theta},\boldsymbol{X}} \left[\mathbb{E}_{\boldsymbol{\varepsilon}} \left[\hat{f}(\boldsymbol{x}) \right]^2 \right] = F^2 \Psi_2^v, \tag{3.11}$$

$$\mathbb{E}_{\boldsymbol{x},\boldsymbol{\Theta},\boldsymbol{X}} \left[\mathbb{E}_{\boldsymbol{\varepsilon}} \left[\hat{f}(\boldsymbol{x})^2 \right] - \mathbb{E}_{\boldsymbol{\varepsilon}} \left[\hat{f}(\boldsymbol{x}) \right]^2 \right] = \tau^2 \Psi_3^v, \tag{3.12}$$

$$\mathbb{E}_{\boldsymbol{x},\boldsymbol{X}} \left[\mathbb{E}_{\boldsymbol{\varepsilon},\boldsymbol{\Theta}} \left[\hat{f}(\boldsymbol{x}) \right]^2 \right] = F^2 \Psi_2^e, \tag{3.13}$$

$$\mathbb{E}_{\boldsymbol{x},\boldsymbol{X}} \left[\mathbb{E}_{\boldsymbol{\varepsilon},\boldsymbol{\Theta}} \left[\hat{f}(\boldsymbol{x})^2 \right] - \mathbb{E}_{\boldsymbol{\varepsilon},\boldsymbol{\Theta}} \left[\hat{f}(\boldsymbol{x}) \right]^2 \right] = \tau^2 \Psi_3^e, \tag{3.14}$$

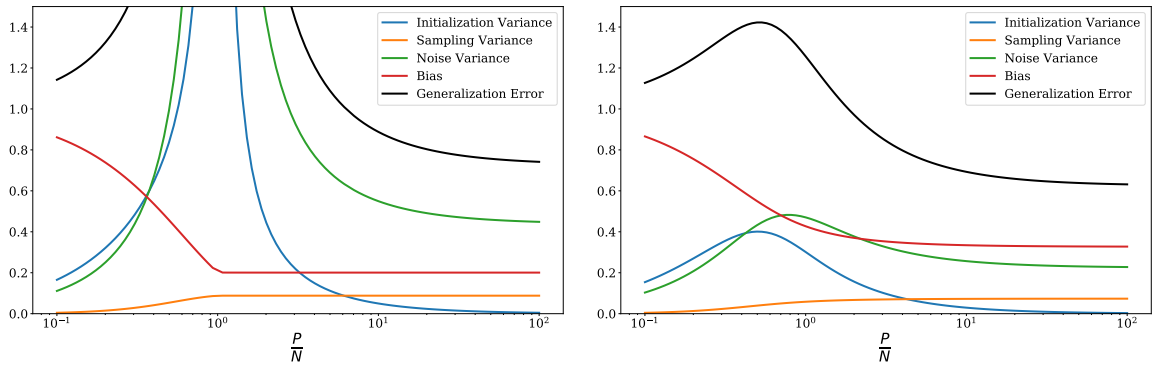
$$\mathbb{E}_{\boldsymbol{x}} \left[\mathbb{E}_{\boldsymbol{\varepsilon},\boldsymbol{\Theta},\boldsymbol{X}} \left[\hat{f}(\boldsymbol{x}) \right]^2 \right] = F^2 \Psi_2^b, \tag{3.15}$$

where the full expression of the terms $\{\Psi_1, \Psi_2^v, \Psi_3^v, \Psi_2^e, \Psi_3^e, \Psi_2^b\}$ is deferred to App. B.1. The derivation, which is presented in App. B.2, follows methods developed in Statistical Physics, and is summarized in the following steps:

1. *Mapping to a random matrix theory problem.* The first step is to express the right-hand sides of Equations 3.10-3.15 as traces over random matrices. This is achieved by replacing our model with its asymptotically equivalent Gaussian covariate model [82], in which the non-linearity of the activation function is encoded as an extra noise term. This enables to take the expectation value with respect to the test sample $\boldsymbol{x}$.
2. *Mapping to a statistical physics model.* The random matrix theory problem resulting from the solution of ridge regression (3.4) involves inverse random matrices. In order to evaluate their expection value, we use the formula:
$$M_{ij}^{-1} = \lim_{n \to 0} \int \prod_{\alpha=1}^{n} \prod_{i=1}^{D} d\eta_i^\alpha \eta_i^1 \eta_j^1 \, \mathrm{e}^{-\frac{1}{2} \eta_i^\alpha M_{ij} \eta_j^\alpha},$$
which is based on the Replica Trick [109, 129]. The Gaussian integrals over $\boldsymbol{\varepsilon}, \boldsymbol{\Theta}, \boldsymbol{X}$ can then be straightforwardly performed and lead to a Statistical Physics model for the auxiliary variables η_i^α.
3. *Mean-Field Theory.* The model for the η_i^α variables can then be solved by introducing as order parameters the $n \times n$ overlap matrices $Q^{\alpha\beta} = \frac{1}{P} \sum_{i=1}^{P} \eta_i^\alpha \eta_i^\beta$ and using replica theory [109], see the supplemental material (SM) for the detailed computation[4].

The Ψ's may also be estimated numerically at finite size by evaluating the traces of the random matrices appearing in the Gaussian covariate model at the end of step 1. Figure 3.4 shows that results thus obtained are in excellent agreement with the asymptotic expressions even at moderate sizes, e.g.

[4] In order to obtain the asymptotic formulas for the Ψ's we need to compute (what are called in the Statistical Physics jargon) fluctuations around mean-field theory.



(a) Small regularization $\lambda = 10^{-5}$

(b) Large regularization $\lambda = 10^{-1}$

Figure 3.2: **The benefit of overparametrization comes from reducing noise and initialization variance.** Decomposition of the test error into the bias and the various sources of variance as function of the overparametrization ratio P/N for $N/D = 1, \mathrm{SNR} = F/\tau = 1$. Notice the contrasting behaviors at the IT: the noise and initialization variances diverge then decrease monotonically whereas the sampling variance and the bias display a kink followed by a plateau. These singular behaviors are smoothed out by regularization.

$D = 200$, demonstrating the robustness of steps 2 and 3, which differ from the approach presented in [82].

The indices v, e, d in $\{\Psi_1, \Psi_2^v, \Psi_3^v, \Psi_2^e, \Psi_3^e, \Psi_2^b\}$ stand for *vanilla*, *ensemble* and *bagging*. The *vanilla* terms, which amount to a bias-variance decomposition with respect to noise in the labels, are sufficient to obtain the test error of a single RF model and were computed in [82]. The *ensemble* terms and *bagging* terms, which are new, respectively allow to study initialization and sampling variance, and hence obtain the test error given by averaging the predictions of several different learners trained on the same dataset (ensembling) and on different splits of the dataset (bagging), see section 3.3.2.

3.3 Bias-variance decomposition

3.3.1 Expression of bias and variances

The results of the previous section allow to rewrite the decomposition of the test error as follows:

$$\mathcal{E}_{\mathrm{Noise}} = \tau^2 \Psi_3^v, \tag{3.16}$$

$$\mathcal{E}_{\mathrm{Init}} = F^2(\Psi_2^v - \Psi_2^e), \tag{3.17}$$

$$\mathcal{E}_{\mathrm{Samp}} = F^2\left(\Psi_2^e - \Psi_2^b\right), \tag{3.18}$$

$$\mathcal{E}_{\mathrm{Bias}} = F^2\left(1 - 2\Psi_1 + \Psi_2^b\right). \tag{3.19}$$

These contributions, together with the test error, are shown in Fig. 3.2a for small regularization and Fig. 3.2b for large regularization.

Interpolation threshold The peak at the IT is completely due to noise and initialization variance, which both diverge at vanishing regularization. In contrast, the sampling variance and the bias remain

finite and exhibit a phase transition at $P = N$, which is revealed by a kink at vanishing regularization. Adding regularization smooths out these singular behaviours: it removes the divergence and irons out the kink.

Overparametrized regime In the overparametrized regime, the sampling variance and the bias do not vary substantially (they remain constant for vanishing regularization). The decrease of the test error is entirely due to the decrease of the noise and initalization variances for $P > N$. In the limit $P/N \to \infty$, the initialization variance vanishes, whereas there remains an irreducible noise variance. Hence, we conclude that the *origin of the double descent curve lies in the behavior of noise and initialization variances*. The benefit of overparametrizing stems only from reducing these two contributions.

These results are qualitatively similar to the empirical decomposition of [38] for real neural networks. Our results differ however from those of [82] where the authors relate the over-fitting peak occurring at $P = N$ to a divergence in both the variance and the bias terms. This is due to the fact the bias term, as defined in that paper, also includes the initialization variance[5]. When the two are disentangled, it becomes clear that it is only the latter which is responsible for the divergence: the bias is, in fact, well-behaved at $P = N$.

Discussion The phenomenology described above can be understood by noting that the RF model essentially performs linear regression, learning a vector $\boldsymbol{a} \in \mathbb{R}^P$ on a dataset $\boldsymbol{Z} = \sigma(\boldsymbol{\Theta}\boldsymbol{X}/\sqrt{D}) \in \mathbb{R}^{N \times P}$ projected from the original $\boldsymbol{X} \in \mathbb{R}^{N \times D}$. Since $\boldsymbol{a}$ is constrained to lie in the space spanned by $\boldsymbol{Z}$, which is of dimension $\min(N, P)$, the model gains expressivity when P increases while staying smaller than N.

At $P = N$, the problem becomes fully determined: the data is perfectly interpolated for vanishing λ. Two types of noise are overfit: (i) the *stochastic noise* corrupting the labels, yielding the divergence in noise variance, and (ii) the *deterministic noise* [119, 130] stemming from the non-linearity of the activation function which cannot be captured, yielding the divergence in initialization variance. However, by further increasing P, the noise is spread over more and more random features and is effectively averaged out. Consequently, the test error decreases again as P increases.

When we make the problem deterministic by averaging out all sources of randomness, i.e. by considering the bias, we see that increasing P beyond N has no effect whatsoever. Indeed, the extra degrees of freedom, which lie in the null space of $\boldsymbol{Z}$, do not provide any extra expressivity: at vanishing regularization, they are killed by the pseudo-inverse to reach the minimum norm solution. For non-vanishing λ, a similar phenomenology is observed but the IT is reached slightly after $P = N$ since the expressivity of the learner is lowered by regularization.

Is it always better to be overparametrized ? A common thought is that the double descent curve always reaches its minimum in the over-parametrized regime, leading to the idea that the corresponding model "cannot overfit". In this section, we show that this is not always the case. Three factors tend to shift the optimal generalization to the underparametrized regime: (i) increasing the numbers of learners from which we average the predictions, K, (ii) decreasing the signal-to-noise ratio (SNR), F/τ, and (iii) decreasing the size of the dataset, N/D. In other words, when ensembling on a small, noisy dataset, one is better off using an underparametrized model.

[5] For a given set of random features this is legitimate, but from the perspective of lazy learning the randomness in the features corresponds to the one due to initialization, which is an additional source of variance.

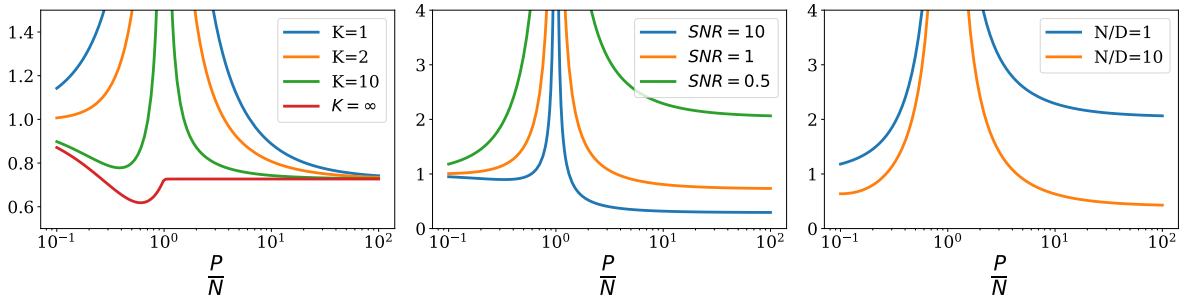


Figure 3.3: Generalisation error as a function of P/N: depending on the values of K, F/τ and N/D, optimal generalization can be reached in the underparametrized regime or the overparametrized regime. **Left:** $F/\tau = 1$, $N/D = 1$ and we vary K. This is the same as figure 5 in the main text, except that the higher noise causes the ensembling curve at $K \to \infty$ to exhibit a *dip* in the underparametrized regime. **Center:** $K = 2$, $N/D = 1$ and we vary F/τ. **Right:** $F/\tau = 1$, $K = 2$ and we vary N/D.

These three effects are shown in Fig. 3.3. In the left panel, we see that as we increase K, the minimum of test error jumps to the underparametrized regime $P < N$ for a high enough value of K. In the central/right panels, a similar effect occurs when decreasing the SNR or decreasing N/D.

3.3.2 Impact of ensembling

We now study in detail the impact of ensembling to underpin the empirical observations of the previous chapter. In the lazy regime of deep neural networks, the initial values of the weights only affect the gradient at initialization, which corresponds to the vector of random features. Hence, we can study the effect of ensembling in the lazy regime by averaging the predictions of RF models with independently drawn random feature vectors.

Expression of the test error Consider a set of K identical RF networks whose first layer weights are drawn independently; in the analogy outlined above, they correspond to K independent inizializations of the neural network. . These networks learn from the *same* training set, and provide K estimators $\{\hat{f}_{\boldsymbol{\Theta}^{(k)}}\}$ $(k = 1, ..., K)$. When a new sample $\boldsymbol{x}$ is presented to the system, the output is taken to be the average over the outputs of the K networks. By expanding the square and taking the expectation with respect to the random initalizations, the test error can then be written as:

$$\mathbb{E}_{\{\boldsymbol{\Theta}^{(k)}\}}\left[\epsilon_g\right] = \mathbb{E}_{\boldsymbol{x},\{\boldsymbol{\Theta}^{(k)}\}}\left[\left(\langle\boldsymbol{\beta},\boldsymbol{x}\rangle - \frac{1}{K}\sum_k \hat{f}_{\boldsymbol{\Theta}^{(k)}}(\boldsymbol{x})\right)^2\right]$$
$$= \mathbb{E}_{\boldsymbol{x}}\left[\langle\boldsymbol{\beta},\boldsymbol{x}\rangle^2\right] - \frac{2}{K}\sum_{i=1}^{K}\mathbb{E}_{\boldsymbol{x},\boldsymbol{\Theta}^{(i)}}\left[\langle\boldsymbol{\beta},\boldsymbol{x}\rangle\hat{f}_{\boldsymbol{\Theta}^{(i)}}(\boldsymbol{x})\right] + \frac{1}{K^2}\sum_{i,j=1}^{K}\mathbb{E}_{\boldsymbol{x},\boldsymbol{\Theta}^{(i)},\boldsymbol{\Theta}^{(j)}}\left[\hat{f}_{\boldsymbol{\Theta}^{(i)}}(\boldsymbol{x})\hat{f}_{\boldsymbol{\Theta}^{(j)}}(\boldsymbol{x})\right].$$

The key here is to isolate in the double sum the $K(K-1)$ *ensemble* terms $i \neq j$, which involve two different initalizations and yield $\mathbb{E}_{\boldsymbol{x}}\left[\mathbb{E}_{\boldsymbol{\Theta}}\left[\hat{f}_{\boldsymbol{\Theta}}(\boldsymbol{x})\right]^2\right]$, from the K *vanilla* terms which give $\mathbb{E}_{\boldsymbol{x},\boldsymbol{\Theta}}\left[\hat{f}_{\boldsymbol{\Theta}}(\boldsymbol{x})^2\right]$. This allows to express the test error in terms of the quantities defined in (3.10) to (3.15) and leads to the

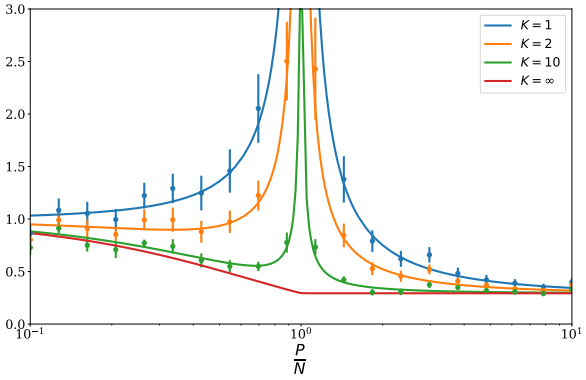


Figure 3.4: **Ensembling suppresses the overfitting peak at the interpolation threshold.** We plotted the test error when ensembling $K = 1, 2, 10$ differently initialized RF models as function of the overparametrization ratio P/N. We fixed $\lambda = 10^{-5}$, $N/D = 1$, SNR $= 10$. For comparison, we show the results of numerical simulations at finite $D = 200$: the vertical bars depict the standard deviation over 10 runs. Note that our analytic expression 3.20 gives us access to the limit $N \to \infty$, where the divergence at $P = N$ is entirely suppressed.

analytic formula for the test error valid for any $K \in \mathbb{N}$:

$$\mathbb{E}_{\{\boldsymbol{\Theta}^{(k)}\},\boldsymbol{X},\boldsymbol{\varepsilon}}\left[\epsilon_g\right] = F^2\left(1 - 2\Psi_1^v\right) + \frac{1}{K}\left(F^2\Psi_2^v + \tau^2\Psi_3^v\right) + \left(1 - \frac{1}{K}\right)\left(F^2\Psi_2^e + \tau^2\Psi_3^e\right). \tag{3.20}$$

We see that ensembling amounts to a linear interpolation between the *vanilla* terms Ψ_2^v, Ψ_3^v, for $K = 1$, and the *ensemble* terms Ψ_2^e, Ψ_3^e for $K \to \infty$.

The effect of ensembling on the double descent curve is shown in Fig. 3.4. As K increases, the overfitting peak at the IT is diminished. This observation is very similar to the empirical findings of [94] in the context of real neural networks. Our analytic expression agrees with the numerical results obtained by training RF models, even at moderate size $D = 200$.

Note that a related procedure is the *bagging* approach, where the dataset is partitioned into K splits of equal size and each one of the K differently initialized learners is trained on a distinct split. This approach and was studied for kernel learning in [131], and is analyzed within our framework in the SM.

Ensembling reduces the double trouble The bias-variance decomposition of the test error makes the suppression of the divergence explicit. The ensembled bias and variance are given by:

$$\mathcal{E}_{\text{Noise}} = \tau^2\left(\Psi_3^e + \frac{1}{K}\left(\Psi_3^v - \Psi_3^e\right)\right), \tag{3.21}$$

$$\mathcal{E}_{\text{Init}} = \frac{F^2}{K}\left(\Psi_2^v - \Psi_2^e\right), \tag{3.22}$$

$$\mathcal{E}_{\text{Samp}} = F^2\left(\Psi_2^e - \Psi_2^b\right), \tag{3.23}$$

$$\mathcal{E}_{\text{Bias}} = F^2\left(1 - 2\Psi_1 + \Psi_2^b\right). \tag{3.24}$$

These equations show that ensembling only affects the noise and initialization variances. In both cases, their divergence at the IT (due to Ψ_2^v, Ψ_3^v) is suppressed as $1/K$, as can be seen in 3.4.At $P > N$,

ensembling and overparametrizing have a very similar effect: they wipe out these two troubling sources of randomness by averaging them out over more random features. Indeed, we see in Fig. 3.4 that in this overparametrized regime, sending $K \to \infty$ has the same effect as sending $P/N \to \infty$: in both cases the system approaches the kernel limit. At $P < N$, this is not true: as shown in [132], the $K \to \infty$ predictor still operates in the kernel limit, but with an effective regularization parameter $\tilde{\lambda} > \lambda$ which diverges as $P/N \to 0$. This (detrimental) implicit regularization increases the test error.

3.3.3 Is ensembling optimal?

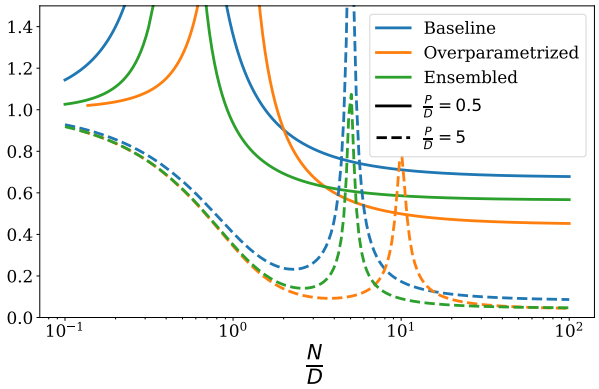


Figure 3.5: **Overparametrization is generally better than ensembling as equal number of features.** Comparison of the test error of a RF model (blue) with that obtained by doubling the number features (orange) or ensembling over two initializations of the features (green), as function of N/D. The parameters are $\lambda = 10^{-5}$, SNR $= 10$, $P/D = 0.5$ (solid lines) and $P/D = 5$ (dashed lines).

Ensembling vs. overparametrization As we have shown, ensembling and overparametrizing have similar effects in the lazy regime. But which is more powerful: ensembling K models, or using a single model with K times more features? The answer is given in Fig. 3.5 for $K = 2$ where we plot our analytical results while varying the number of data points, N. Two observations are particularly interesting. First, overparametrization shifts the IT, opening up a region where ensembling outperforms overparametrizing. Second, overparametrization yields a higher asymptotic improvement in the large dataset limit $N/D \to \infty$, but the gap between overparametrizing and ensembling is reduced as P/D increases. At $P \gg D$, where we are already close to the kernel limit, both methods yield a similar improvement. Note that since ridge regression involves the inversion of a $P \times P$ matrix, ensembling is significantly more efficient computation-wise.

Ensembling vs. optimal regularization In all the results presented above, we keep the regularization constant λ fixed. However, by appropriately choosing the value of λ at each value of P/N, the performance is improved. As Fig. 3.6 (left) reveals, the optimal value of λ decreases with K since the minimum of the test error shifts to the left when increasing K. In other words, ensembling is best when the predictors one ensembles upon are individually under-regularized, as was observed previously for kernel learning in [133].

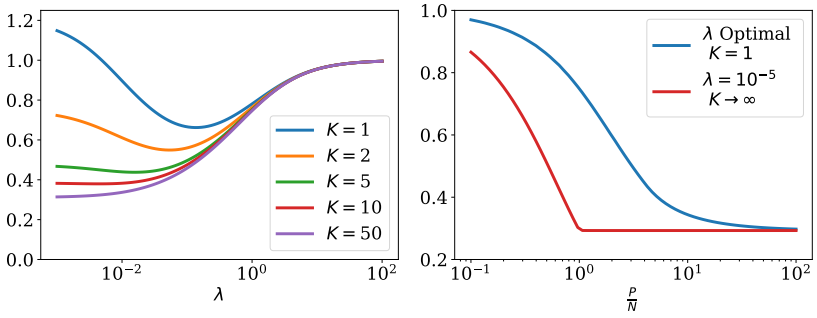


Figure 3.6: **Ensembled predictors are better off under-regularized. Left:** Test error as a function of λ for various values of K and parameters $P/D = 2$, $N/D = 1$, SNR $= 10$. **Right:** Comparison of test error for an optimal regularized system with $K = 1$ and the system with $K \to \infty$ with $\lambda = 10^{-5}$. Optimization performed over 50 values of λ from 10^{-5} to 10^{2}. Parameters are $N/D = 1$, SNR $= 10$.

Through a comparison between the performance, in test error perspective, of the ensembled system with $K \to \infty$ and of a single RF model ($K = 1$) optimally regularized, Fig. 3.6 (right) shows that the ensembled system always performs better than the optimally regularized one.

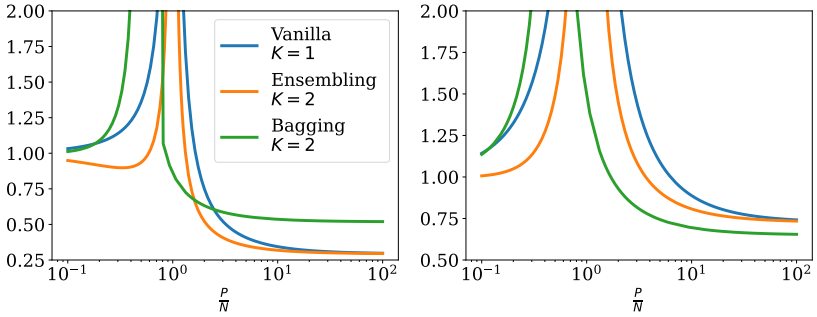


Figure 3.7: **Ensembling is better for noiseless data, bagging is better for noisy data.** Comparison between performances of ensembling and bagging for $K = 2$ at different SNR. **Left:** SNR$=\frac{F}{\tau} = 10$. **Right:** SNR$=\frac{F}{\tau} = 1$ Computations performed with fixed $N/D = 1$, $F = 1$ and $\lambda = 10^{-5}$.

Ensembling vs. bagging Another way to average the predictions of differently initialized learners is the *bagging* approach [131]. In this framework, the data set is divided into K splits of size $N_{\text{eff}} = N/K$. Each of the K differently initalized learner is trained on a distinct split. This approach is extremely useful for kernel learning [133], where the computational burden is in the inversion of the Gram matrix which is of size $N \times N$. In the random projection approach considered here, it does not offer any computational gain, however it is interesting how it affects the test error.

Within our framework, the test error can easily be calculated as:

$$\mathbb{E}_{\{\boldsymbol{\Theta}^{(k)}\},\boldsymbol{X},\boldsymbol{\varepsilon}}\left[\epsilon_g\right] = F^2\left(1 - 2\Psi_1^v\right) + \frac{1}{K}\left(F^2\Psi_2^v + \tau^2\Psi_3^v\right) + \left(1 - \frac{1}{K}\right)F^2\Psi_2^b, \tag{3.25}$$

where the effective number of data points which enters this formula is $N_{\text{eff}} = N/K$ due to the splitting of the training set.

Comparing the previous expression with that obtained for ensembling (3.20) is instructive: here, increasing K replaces the *vanilla* terms Ψ_2^v, Ψ_3^v by the *bagging* term Ψ_2^b. This shows that bagging has a *denoising* effect: at $K \to \infty$, the effect of the additive noise on the labels is completely suppressed. This was not the case for ensembling. The price to pay is that N_{eff} decreases, hence one is shifted to the underparametrized regime.

In Figure 3.7, we see that the kernel limit error of the bagging approach, i.e. the asymptotic value of the error at $P/N \to \infty$, is different from the usual kernel limit error, since the effective dataset is two times smaller at $K = 2$. The denoising effect of the bagging approach is illustrated by the fact that its kernel limit error is higher at high SNR, but lower at low SNR. This is of practical relevance, and is much related to the beneficial effect of *bagging* in noisy scenarios [134].

3.4 Conclusion

In this chapter, we performed a bias-variance decomposition of the test error of random feature models, aiming to reconcile the double descent curve with the U-shaped curve excepted from the classic bias-variance tradeoff. We show that although the bias decreases monotonically, as expected, the variance displays a very different behavior: it increases until the IT, as expected, then decreases beyond. This explains not only the double decent curve, but also the observation made in the previous chapter that ensembling achieves optimal performance near the IT; indeed, the benefit of overparametrizing beyond the IT can equivalently be obtained either by ensembling or overparametrizing.

Chapter 4

Triple Descent and the Two Kinds of Overfitting: When and Why do they Occur?

In the "double descent" curve presented in the previous chapters, the test error peaks when the number of training examples N is of the same order as the number of parameters P. In earlier works [85], a similar phenomenon was shown to exist in simpler models such as linear regression, but the peak instead occurs when N is equal to the input dimension D. Since both peaks coincide with the IT, they are often conflated in the litterature.

In this chapter, we show that despite their apparent similarity, these two scenarios are inherently different. In fact, both peaks can co-exist when neural networks are applied to noisy regression tasks. The relative size of the peaks is then governed by the degree of nonlinearity of the activation function. Building on recent developments in the analysis of random feature models, we provide a theoretical ground for this sample-wise *triple descent*. As shown previously, the *nonlinear peak* at $N = P$ is a true divergence caused by the extreme sensitivity of the output function to both the noise corrupting the labels and the initialization of the random features (or the weights in neural networks). This peak survives in the absence of noise, but can be suppressed by regularization. In contrast, the *linear peak* at $N = D$ is solely due to overfitting the noise in the labels, and forms earlier during training. We show that this peak is implicitly regularized by the nonlinearity, which is why it only becomes salient at high noise and is weakly affected by explicit regularization. Throughout this chapter, we compare analytical results obtained in the random feature model with the outcomes of numerical experiments involving deep neural networks.

4.1 Introduction

So far, we have presented the following paradox: classical learning theory predicts that test error should follow a U-shaped curve as the number of parameters P increases, and a monotonous decrease as the number of training examples N increases. Instead, for deep neural networks, increasing P and N respectively yields parameter-wise and sample-wise double descent curves [32, 33, 82, 118, 135, 136], whereby the test error first decreases, then peaks at the IT, then decreases monotonically again.

Although double descent has only recently gained interest in the context of deep learning, a seemingly

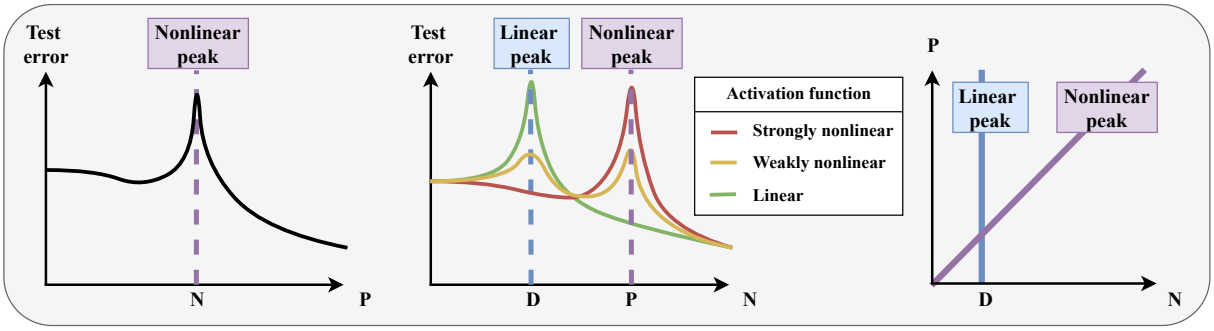


Figure 4.1: **Left:** The parameter-wise profile of the test error exhibits double descent, with a peak at $P = N$. **Middle:** The sample-wise profile can, at high noise, exhibit a single peak at $N = P$, a single peak at $N = D$, or a combination of the two (triple descent[1]) depending on the degree of nonlinearity of the activation function. **Right:** Color-coded location of the peaks in the (P, N) phase space.

similar phenomenon has been well-known for several decades for simpler models such as least squares regression [72–76], and has recently been studied in more detail in an attempt to shed light on the double descent curve observed in deep learning [79, 137–139]. However, in the context of linear models, the number of parameters P is not a free parameter: it is necessarily equal to the input dimension D. The IT occurs at $N = D$, and coincides with a peak in the test error which we refer to as the *linear peak*. As shown in Chap. 2, for neural networks with nonlinear activations, the IT surprisingly becomes independent of D and is instead observed when the number of training examples is of the same order as the total number of training parameters, i.e. $N \sim P$: we refer to the corresponding peak as the *nonlinear peak*.

Somewhere in between these two scenarios lies the case of neural networks with *linear* activations. They have $P > D$ parameters, but only D of them are independent: the IT occurs at $N = D$. However, their dynamical behaviour shares some similarities with that of deep nonlinear networks, and their analytical tractability has given them significant attention [37, 140, 141]. A natural question is the following: what would happen for a "quasi-linear" network, e.g. one that uses a sigmoidal activation function with a high saturation plateau? Would the overfitting peak be observed both at $N = D$ and $N = P$, or would it somehow lie in between?

In this chapter, we unveil the similarities and the differences between the linear and nonlinear peaks. In particular, we address the following questions:

- Are the linear and nonlinear peaks two different phenomena?
- If so, can both be observed simultaneously, and can we differentiate their sources?
- How are they affected by the activation function? Can they both be suppressed by regularizing or ensembling? Do they appear at the same time during training?

4.1.1 Contribution

In modern neural networks, the double descent phenomenon is mostly studied by increasing the number of parameters P (Fig. 4.1, left), and more rarely, by increasing the number of training examples N

[1]The name "triple descent" refers to the presence of two peaks instead of just one in the famous "double descent" curve, but in most cases the test error does not actually descend before the first peak.

(Fig. 4.1, middle) [118]. The analysis of linear models is instead performed by varying the ratio P/N. By studying the full (P, N) phase space (Fig. 4.1, right), we disentangle the role of the linear and the nonlinear peaks in modern neural networks, and elucidate the role of the input dimension D.

In Sec.4.2, we demonstrate that the linear and nonlinear peaks are two different phenomena by showing that they can co-exist in the (P, N) phase space in noisy regression tasks. This leads to a sample-wise *triple descent*, as sketched in Fig. 4.1. We consider both the random feature (RF) model presented in the previous chapter and a more realistic task involving neural networks.

In Sec. 4.3, we provide a theoretical analysis of this phenomenon for the RF model. We examine the eigenspectrum of random feature Gram matrices and show that whereas the nonlinear peak is caused by the presence of small eigenvalues [37], the small eigenvalues causing the linear peak gradually disappear when the activation function becomes nonlinear: the linear peak is implicitly regularized by the nonlinearity. Reusing the bias-variance decomposition of the test error presented in the previous chapter, we reveal that the linear peak is solely caused by overfitting the noise corrupting the labels, whereas the nonlinear peak is also caused by the variance due to the initialization of the random feature vectors (which plays the role of the initialization of the weights in neural networks).

Finally, in Sec. 4.4, we present the phenomenological differences which follow from the theoretical analysis. Increasing the degree of nonlinearity of the activation function weakens the linear peak and strengthens the nonlinear peak. We also find that the nonlinear peak can be suppressed by regularizing or ensembling, whereas the linear peak cannot since it is already implicitly regularized. Finally, we note that the nonlinear peak appears much later under gradient descent dynamics than the linear peak, since it is caused by small eigenmodes which are slow to learn.

Reproducibility We release the code necessary to reproduce the data and figures in this chapter publicly at `https://github.com/sdascoli/triple-descent-paper`.

4.1.2 Related work

Various sources of sample-wise non-monotonicity have been observed since the 1990s, from linear regression [75] to simple classification tasks [142, 143]. In the context of adversarial training, [144] shows that increasing N can help or hurt generalization depending on the strength of the adversary. In the non-parametric setting of [145], an upper bound on the test error is shown to exhibit *multiple descent*, with peaks at each $N = D^i, i \in \mathbb{N}$.

Two concurrent papers also discuss the existence of a triple descent curve, albeit of different nature to ours. On one hand, [146] observes a sample-wise triple descent in a non-isotropic linear regression task. In their setup, the two peaks stem from the block structure of the covariance of the input data, which presents two eigenspaces of different variance; both peaks boil down to what we call "linear peaks". [147] pushed this idea to the extreme by designing the covariance matrix in such a way to make an arbitrary number of linear peaks appear.

On the other hand, [148] presents a parameter-wise triple descent curve in a regression task using the Neural Tangent Kernel of a two-layer network. Here the two peaks stem from the block structure of the covariance of the random feature Gram matrix, which contains a block of linear size in input dimension (features of the second layer, i.e. the ones studied here), and a block of quadratic size (features of the first layer). In this case, both peaks are "nonlinear peaks".

The triple descent curve presented here is of different nature: it stems from the general properties of nonlinear projections, rather than the particular structure chosen for the data [146] or regression

kernel [148]. To the best of our knowledge, the disentanglement of linear and nonlinear peaks presented here is novel, and its importance is highlighted by the abundance of papers discussing both kinds of peaks.

On the analytical side, our work directly uses the results for high-dimensional random features models derived in Chap. 3 (for the test error and its bias-variance decomposition) and [149] (for the spectral analysis).

4.2 Triple descent in the test error phase space

We analyze the (P, N) phase space of the test error in noisy regression tasks to demonstrate the triple descent phenomenon, in two separate setups: on the analytical side, the RF model studied in the previous chapter, and on the numerical side, a teacher-student task involving neural networks trained with gradient descent.

Dataset For both setups, the input data $\boldsymbol{X} \in \mathbb{R}^{N\times D}$ consists of N vectors in D dimensions whose elements are drawn i.i.d. from $\mathcal{N}(0,1)$[2]. For each model, there is an associated label generator $f^\star$ corrupted by additive Gaussian noise: $y = f^\star(\boldsymbol{x}) + \epsilon$, where the noise variance is inversely related to the signal to noise ratio (SNR), $\epsilon \sim \mathcal{N}(0, 1/\mathrm{SNR})$.

4.2.1 Random features regression (RF model)

Model We consider the RF model introduced in the previous chapter, and use the same notations as before (see Fig. 3.1):

$$f(\boldsymbol{x}) = \sum_{i=1}^{P} \boldsymbol{a}_i \sigma\left(\frac{\langle \boldsymbol{\Theta}_i, \boldsymbol{x}\rangle}{\sqrt{D}}\right). \tag{4.1}$$

σ is a pointwise activation function, the choice of which will be of prime importance in the study. The ground truth is a linear model given by $f^\star(\boldsymbol{x}) = \langle \boldsymbol{\beta}, \boldsymbol{x}\rangle/\sqrt{D}$. Elements of $\boldsymbol{\Theta}$ and $\boldsymbol{\beta}$ are drawn i.i.d from $\mathcal{N}(0,1)$.

Training The second layer weights, i.e. the elements of $\boldsymbol{a}$, are calculated via ridge regression with a regularization parameter λ:

$$\hat{\boldsymbol{a}} = \underset{\boldsymbol{a}\in\mathbb{R}^P}{\arg\min}\left[\frac{1}{N}\left(\boldsymbol{y} - \boldsymbol{a}\boldsymbol{Z}^\top\right)^2 + \frac{P\lambda}{D}\|\boldsymbol{a}\|_2^2\right] = \frac{1}{N}\boldsymbol{y}^\top\boldsymbol{Z}\left(\boldsymbol{\Sigma} + \frac{P\lambda}{D}\mathbb{I}_P\right)^{-1} \tag{4.2}$$

$$\boldsymbol{Z}_i^\mu = \sigma\left(\frac{\langle \boldsymbol{\Theta}_i, \boldsymbol{X}_\mu\rangle}{\sqrt{D}}\right) \in \mathbb{R}^{N\times P}, \quad \boldsymbol{\Sigma} = \frac{1}{N}\boldsymbol{Z}^\top\boldsymbol{Z} \in \mathbb{R}^{P\times P} \tag{4.3}$$

4.2.2 Teacher-student regression with neural networks (NN model)

Model We consider a teacher-student neural network (NN) framework where a *student* network learns to reproduce the labels of a *teacher* network. The teacher $f^\star$ is taken to be an untrained ReLU fully-connected network with 3 layers of weights and 100 nodes per layer. The student f is a fully-connected network with 3 layers of weights and nonlinearity σ. Both are initialized with the default PyTorch initialization.

[2]The impact of data structure will be studied in the next chapter

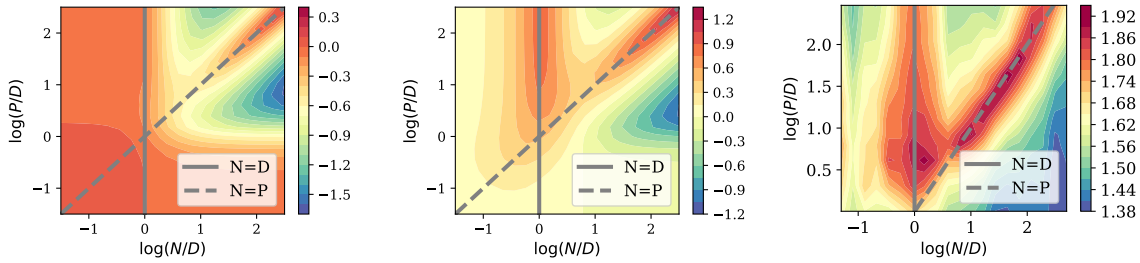
Training We train the student with mean-square loss using full-batch gradient descent for 1000 epochs with a learning rate of 0.01 and momentum 0.9[3]. We examine the effect of regularization by adding weight decay with parameter 0.05, and the effect of ensembling by averaging over 10 initialization seeds for the weights. All results are averaged over these 10 runs.

4.2.3 Test error phase space

In both models, the key quantity of interest is the *test error*, defined as the mean-square loss evaluated on unseen examples:

$$\epsilon_g = \mathbb{E}_{\boldsymbol{x}\sim\mathcal{N}(0,1)}\left[\left(f(\boldsymbol{x}) - f^\star(\boldsymbol{x})\right)^2\right]. \tag{4.4}$$

For the RF model, this quantity was first derived rigorously in [82], in the high-dimensional limit where N, P, D are sent to infinity with their ratios finite. More recently, a different approach based on the Replica Method from Statistic Physics was proposed in [122]; we use this method to compute the analytical phase space. As for the NN model, which operates at finite size $D = 196$, the test error is computed over a test set of 10^4 examples.



(a) RF, $SNR = 2$ (b) RF, $SNR = 0.2$ (c) NN, $SNR = 0.2$

Figure 4.2: **Evidence of triple descent at low signal-to-noise ratio.** Logarithmic plot of the test error in the (P, N) phase space. **(a)**: RF model with SNR $= 2$, $\lambda = 10^{-1}$. **(b)**: RF model with SNR $= 0.2$, $\lambda = 10^{-1}$. The solid arrows emphasize the sample-wise profile, and the dashed lines emphasize the parameter-wise profile. **(c)**: NN model. In all cases, $\sigma = \mathrm{Tanh}$. Analogous results for different activation functions and values of the SNR are shown in App. C.1.

In Fig.4.2, we plot the test error as a function of two intensive ratios of interest: the number of parameters per dimension P/D and the number of training examples per dimension N/D. In the left panel, at high SNR, we observe an overfitting line at $N = P$, yielding a parameter-wise and sample-wise double descent. However when the SNR becomes smaller than unity (middle panel), the sample-wise profile undergoes triple descent, with a second overfitting line appearing at $N = D$. A qualitatively identical situation is shown for the NN model in the right panel[4].

The case of structured data The case of structured datasets such as CIFAR10 is discussed in App. C.3. The main differences are (i) the presence of multiple linear peaks at $N < D$ due to the complex covariance

[3] We use full batch gradient descent with small learning rate to reduce the noise coming from the optimization as much as possible. After 1000 epochs, all observables appear to have converged.

[4] Note that for NNs, we necessarily have $P/D > 1$.

structure of the data, as observed in [146, 147], and (ii) the fact that the nonlinear peak is located sligthly above the line $N = P$ since the data is easier to fit, as observed in Chap. 2.

As we will see in the next chapter, triple descent can also be observed in classification tasks, at moderate signal-to-noise ratios.

4.3 Theory for the RF model

The qualitative similarity between the central and right panels of Fig. 3 indicates that a full understanding can be gained by a theoretical analysis of the RF model, which we present in this section.

4.3.1 High-dimensional setup

As before, we consider the following high-dimensional limit:

$$N, D, P \to \infty, \quad \frac{D}{P} = \psi = \mathcal{O}(1), \quad \frac{D}{N} = \phi = \mathcal{O}(1) \tag{4.5}$$

Then the key quantities governing the behavior of the system are related to the properties of the nonlinearity around the origin:

$$\eta = \int dz \frac{e^{-z^2/2}}{\sqrt{2\pi}} \sigma^2(z), \quad \zeta = \left[\int dz \frac{e^{-z^2/2}}{\sqrt{2\pi}} \sigma'(z) \right]^2 \quad \text{and} \quad r = \frac{\zeta}{\eta} \tag{4.6}$$

As explained in [150], the Gaussian Equivalence Theorem [82, 150, 151] which applies in this high dimensional setting establishes an equivalence to a *Gaussian covariate model* where the nonlinear activation function is replaced by a linear term and a nonlinear term acting as noise:

$$\boldsymbol{Z} = \sigma\left(\frac{\boldsymbol{X}\boldsymbol{\Theta}^\top}{\sqrt{D}}\right) \to \sqrt{\zeta}\frac{\boldsymbol{X}\boldsymbol{\Theta}^\top}{\sqrt{D}} + \sqrt{\eta - \zeta}\boldsymbol{W}, \quad \boldsymbol{W} \sim \mathcal{N}(0, 1) \tag{4.7}$$

Of prime importance is the *degree of linearity* $r = \zeta/\eta \in [0, 1]$, which indicates the relative magnitudes of the linear and the nonlinear terms[5].

4.3.2 Spectral analysis

As expressed by Eq. 4.3, RF regression is equivalent to linear regression on a structured dataset $\boldsymbol{Z} \in \mathbb{R}^{N \times P}$, which is projected from the original i.i.d dataset $\boldsymbol{X} \in \mathbb{R}^{N \times D}$. In [37], it was shown that the peak which occurs in unregularized linear regression on i.i.d. data is linked to vanishingly small (but non-zero) eigenvalues in the covariance of the input data. Indeed, the norm of the interpolator needs to become very large to fit small eigenvalues according to Eq.4.3, yielding high variance.

Following this line, we examine the eigenspectrum of $\boldsymbol{\Sigma} = \frac{1}{N}\boldsymbol{Z}^\top\boldsymbol{Z}$, which was derived in a series of recent papers. The spectral density $\rho(\mu)$ can be obtained from the resolvent $G(z)$ [149, 152–154]:

$$\begin{aligned} \rho(\mu) &= \frac{1}{\pi} \lim_{\epsilon \to 0^+} \mathrm{Im} G(\mu - i\epsilon), \qquad G(z) = \frac{\psi}{z} A\left(\frac{1}{z\psi}\right) + \frac{1-\psi}{z} \\ A(t) &= 1 + (\eta - \zeta) t A_\phi(t) A_\psi(t) + \frac{A_\phi(t) A_\psi(t) t\zeta}{1 - A_\phi(t) A_\psi(t) t\zeta} \end{aligned} \tag{4.8}$$

[5]Note from Eq. 4.6 that for non-homogeneous functions such as Tanh, r also depends on the variance of the inputs and fixed weights, both set to unity here: intuitively, smaller variance will yield smaller preactivations which will lie in the linear region of the Tanh, increasing the effective value of r.

where $A_\phi(t) = 1 + (A(t) - 1)\phi$ and $A_\psi(t) = 1 + (A(t) - 1)\psi$. We solve the implicit equation for $A(t)$ numerically, see for example Eq. 11 of [149].

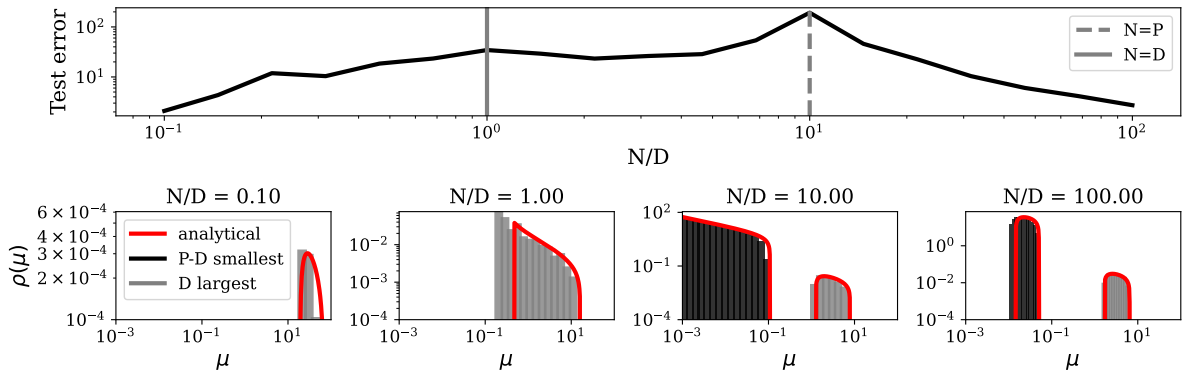
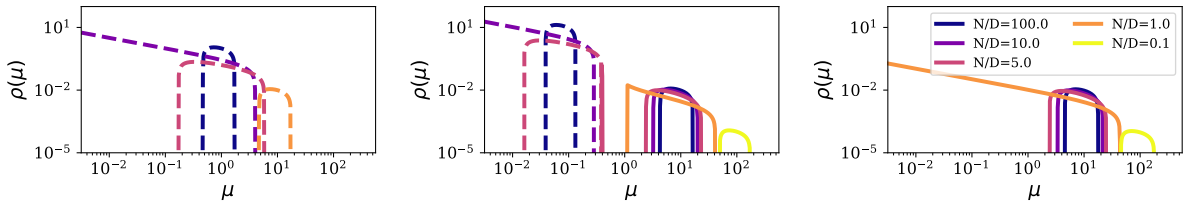


Figure 4.3: **The covariance of the random features splits into two components.** Empirical eigenspectrum of the covariance of the projected features $\mathbf{\Sigma} = \frac{1}{N}\boldsymbol{Z}^\top \boldsymbol{Z}$ at various values of N/D, with the corresponding test error curve shown above. Analytics match the numerics even at $D = 100$. We color the top D eigenvalues in gray, which allows to separate the linear and nonlinear components at $N > D$. We set $\sigma = \mathrm{Tanh}$, $P/D = 10$, $\mathrm{SNR} = 0.2$, $\lambda = 10^{-5}$.



(a) Absolute value ($r = 0$) (b) Tanh ($r \simeq 0.92$) (c) Linear ($r = 1$)

Figure 4.4: **The spectral gap closes at $N = P$, but not at $N = D$.** Analytical eigenspectrum of $\mathbf{\Sigma}$ for $\eta = 1$, $P/D = 10$, and $\zeta = 0, 0.92, 1$ **(a,b,c)**. We distinguish linear and nonlinear components by using respectively solid and dashed lines. **(a)** (purely nonlinear): the spectral gap vanishes at $N = P$ (i.e. $N/D = 10$). **(c)** (purely linear): the spectral gap vanishes at $N = D$. **(b)** (intermediate): the spectral gap of the nonlinear component vanishes at $N = P$, but the gap of the linear component does not vanish at $N = D$.

In the bottom row of Fig. 4.3 (see also middle panel of Fig. 4.4), we show the numerical spectrum obtained for various values of N/D with $\sigma = \mathrm{Tanh}$, and we superimpose the analytical prediction obtained from Eq. 4.8. At $N > D$, the spectrum separates into two components: one with D large eigenvalues, and the other with $P - D$ smaller eigenvalues. The spectral gap (distance of the left edge of the spectrum to zero) closes at $N = P$, causing the nonlinear peak [111], but remains finite at $N = D$.

Fig. 4.4 shows the effect of varying r on the spectrum. We can interpret the results from Eq. 4.7:

- **"Purely nonlinear"** ($r = 0$): this is the case of even activation functions such as $x \mapsto |x|$, which verify $\zeta = 0$ according to Eq. 4.6. The spectrum of $\mathbf{\Sigma}_{nl} = \frac{1}{N}\boldsymbol{W}^\top \boldsymbol{W}$ follows a Marcenko-Pastur

distribution of parameter $c = P/N$, concentrating around $\mu = 1$ at $N/D \to \infty$. The spectral gap closes at $N = P$.

- **"Purely linear"** ($r = 1$): this is the maximal value for r, and is achieved only for linear networks. The spectrum of $\boldsymbol{\Sigma}_l = \frac{1}{ND}(\boldsymbol{X}\boldsymbol{\Theta}^\top)^\top \boldsymbol{X}\boldsymbol{\Theta}^\top$ follows a product Wishart distribution [155, 156], concentrating around $\mu = P/D = 10$ at $N/D \to \infty$. The spectral gap closes at $N = D$.

- **Intermediate** ($0 < r < 1$): this case encompasses all commonly used activation functions such as ReLU and Tanh. We recognize the linear and nonlinear components, which behave almost independently (they are simply shifted to the left by a factor of r and $1-r$ respectively), except at $N = D$ where they interact nontrivially, leading to implicit regularization (see below).

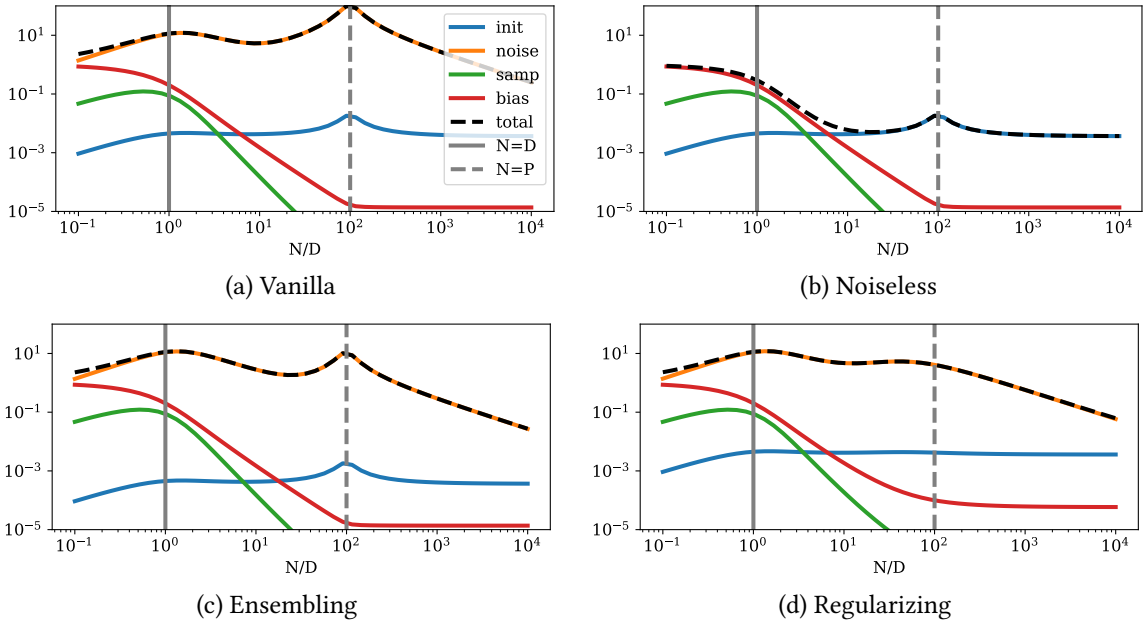
The linear peak is implicitly regularized As stated previously, one can expect to observe overfitting peaks when $\boldsymbol{\Sigma}$ is badly conditioned, i.e. its when its spectral gap vanishes. This is indeed observed in the purely linear setup at $N = D$, and in the purely nonlinear setup at $N = P$. However, in the everyday case where $0 < r < 1$, the spectral gap only vanishes at $N = P$, and not at $N = D$. The reason for this is that a vanishing gap is symptomatic of a random matrix reaching its maximal rank. Since $\mathrm{rk}\,(\boldsymbol{\Sigma}_{nl}) = \min(N, P)$ and $\mathrm{rk}\,(\boldsymbol{\Sigma}_l) = \min(N, P, D)$, we have $\mathrm{rk}\,(\boldsymbol{\Sigma}_{nl}) \geq \mathrm{rk}\,(\boldsymbol{\Sigma}_l)$ at $P > D$. Therefore, the rank of $\boldsymbol{\Sigma}$ is imposed by the nonlinear component, which only reaches its maximal rank at $N = P$. At $N = D$, the nonlinear component acts as an *implicit regularization*, by compensating the small eigenvalues of the linear component. This causes the linear peak to be implicitly regularized be the presence of the nonlinearity.

What is the linear peak caused by? At $0 < r < 1$, the spectral gap vanishes at $N = P$, causing the norm of the estimator $\|\boldsymbol{a}\|$ to peak, but it does not vanish at $N = D$ due to the implicit regularization; in fact, the lowest eigenvalue of the full spectrum does not even reach a local minimum at $N = D$. Nonetheless, a soft *linear peak* remains as a vestige of what happens at $r = 1$. What is this peak caused by? A closer look at the spectrum of Fig. 4.4.b clarifies this question. Although the left edge of the full spectrum is not minimal at $N = D$, the left edge of the *linear component*, in solid lines, reaches a minimum at $N = D$. This causes a peak in $\|\boldsymbol{\Theta}\boldsymbol{a}\|$, the norm of the "linearized network", as shown in App. C.2. This, in turn, entails a different kind of overfitting as we explain in the next section.

4.3.3 Bias-variance decomposition

The previous spectral analysis suggests that both peaks are related to some kind of overfitting. To address this issue, we make use of the bias-variance decomposition presented in the previous chapter.

Only the nonlinear peak is affected by initialization variance In Fig. 4.5.a, we plot the various terms in the decomposition. As observed in the previous chapter, the nonlinear peak is caused by an interplay between initialization and noise variance. This peak appears starkly at $N = P$ in the high noise setup, where noise variance dominates the test error, but also in the noiseless setup (Fig. 4.5.b), where the residual initialization variance dominates: nonlinear networks can overfit even in absence of noise.



(a) Vanilla (b) Noiseless

(c) Ensembling (d) Regularizing

Figure 4.5: **The linear peak is only caused by noise variance, whereas the nonlinear peak is caused by the interplay between noise and initialization varince.** Bias-variance decompostion of the test error in the RF model for $\sigma = \mathrm{ReLU}$ and $P/D = 100$. Regularizing (increasing λ) and ensembling (increasing the number K of initialization seeds we average over) mitigates the nonlinear peak but does not affect the linear peak. **(a)** $K=1, \lambda=10^{-5}, SNR=0.2$. **(b)** Same but $SNR=\infty$. **(c)** Same but $K = 10$. **(d)** Same but $\lambda = 10^{-3}$.

The linear peak vanishes in the noiseless setup In stark contrast, the linear peak which appears clearly at $N = D$ in Fig. 4.5.a is caused solely by a peak in noise variance, in agreement with [37]. Therefore, it vanishes in the noiseless setup of Fig. 4.5.b. This is expected, as for linear networks the solution to the minimization problem is independent of the initialization of the weights.

4.4 Phenomenology of triple descent

4.4.1 The nonlinearity determines the relative height of the peaks

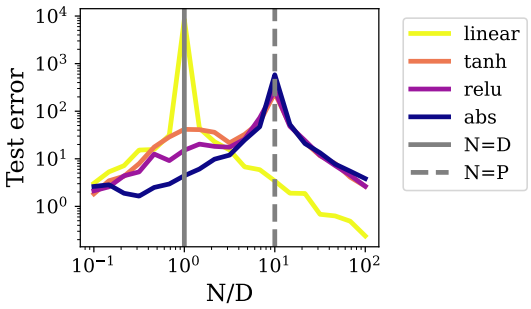
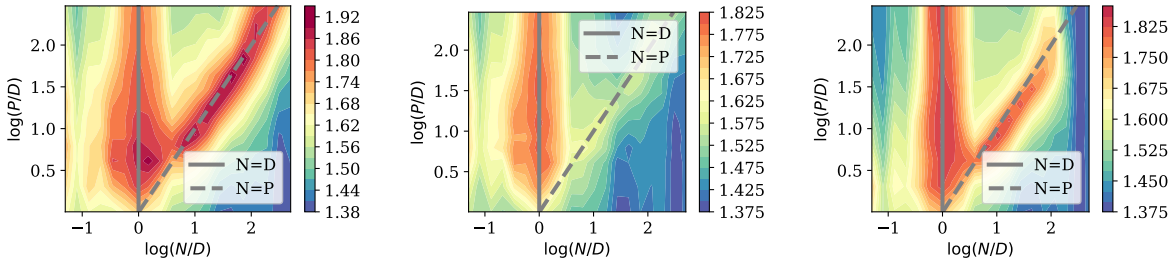


Figure 4.6: **The more linear the activation function, the stronger the linear peak.** Numerical test error of RF models at finite size ($D = 100$), averaged over 10 runs. We set $P/D = 10$, SNR $= 0.2$ and $\lambda = 10^{-3}$.

In Fig. 4.6, we consider RF models with four different activation functions: absolute value ($r = 0$), ReLU ($r = 0.5$), Tanh ($r \sim 0.92$) and linear ($r = 1$). Increasing r strengthens the nonlinear peak (by increasing initialization variance) and weakens the linear peak (by increasing the implicit regularization). In App. C.1, we present additional results where the degree of linearity r is varied systematically in the RF model, and show that replacing Tanh by ReLU in the NN setup produces a similar effect. Note that the behavior changes abruptly near $r = 1$, marking the transition to the linear regime.



(a) Vanilla (b) Ensembling (c) Regularizing

Figure 4.7: **Regularization and ensembling attenuates the nonlinear peak but not the linear peak.** Test error phase space for the NN model with $\sigma = \mathrm{Tanh}$. Weight decay with parameter λ and ensembling over K seeds weakens the nonlinear peak but leaves the linear peak untouched. **(a)** $K=1, \lambda=0, SNR=0.2$. **(b)** Same but $K = 10$. **(c)** Same but $\lambda = 0.05$.

4.4.2 Ensembling and regularization only affects the nonlinear peak

As we have seen in previous chapters, regularization and ensembling can mitigate the nonlinear peak. This is shown in panel (c) and (d) of Fig. 4.5 for the RF model, where ensembling is performed by averaging the predictions of 10 RF models with independently sampled random feature vectors. However, we see that these procedures only weakly affect the linear peak. This can be understood by the fact that the linear peak is already implicitly regularized by the nonlinearity for $r < 1$, as explained in Sec. 4.3.

In the NN model, we perform a similar experiment by using weight decay as a proxy for the regularization procedure, see Fig. 4.7. Similarly as in the RF model, both ensembling and regularizing attenuates the nonlinear peak much more than the linear peak.

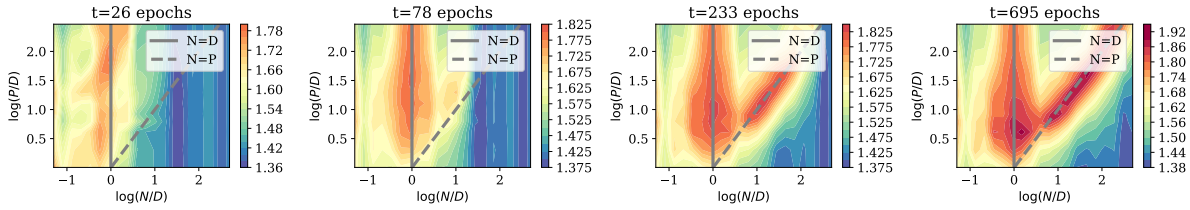


Figure 4.8: **The nonlinear peak appears later than the linear peak.** Test error phase space for the NN model with $\sigma = \mathrm{Tanh}$, plotted at various times during training. The linear peak grows first, followed by the nonlinear peak.

4.4.3 The nonlinear peak forms later during training

To study the evolution of the phase space during training dynamics, we focus on the NN model (there are no dynamics involved in the RF model we considered, where the second layer weights were learnt via ridge regression). In Fig. 4.8, we see that the linear peak appears early during training and maintains throughout, whereas the nonlinear peak only forms at late times.

This can be understood qualitatively as follows [37]: for linear regression the time required to learn a mode of eigenvalue μ in the covariance matrix is proportional to $1/\mu$. Since the nonlinear peak is

due to vanishingly small eigenvalues, which is not the case of the linear peak, the nonlinear peak takes more time to form completely.

4.5 Conclusion

One of the key challenges in solving tasks with network-like architectures lies in choosing an appropriate number of parameters P given the properties of the training dataset, namely its size N and dimension D. By elucidating the structure of the (P, N) phase space, its dependency on D, and distinguishing the two different types of overfitting which it can exhibit, we believe our results can be of interest to practitioners.

Our results leave room for several interesting follow-up questions, among which the impact of (1) various architectural choices, (2) the optimization algorithm, and (3) the structure of the dataset. For future work, we will consider extensions along those lines with particular attention to the structure of the dataset. We believe it will provide a deeper insight into data-model matching.

Chapter 5

On the Interplay between Loss Function and Data Structure

In this work, we extend the results of the previous chapters on the random feature model to classification problems on structured data. Using a different analytical approach, also inspired by statistical physics, we obtain a precise asymptotic expression for the test error of random features models trained on *strong and weak features* - a model of data with input data covariance built from independent blocks allowing us to tune the saliency of low-dimensional structures and their alignment with respect to the target function. Leveraging our analytical results, we explore how properties of data distributions impact generalization in the over-parametrized regime and compare results for the logistic and square loss. Our results show in particular how the logistic loss better captures the structure of the data than the squared loss. Numerical experiments on MNIST and CIFAR10 confirm this insight.

5.1 Introduction

Most existing analytical studies of generalization (including those of the previous chapters) consider synthetic tasks where the data is built from structureless features. Yet, the structure of data plays the center role in generalization. The first aspect of data structure is the distribution of inputs: MNIST and CIFAR10 have the same number of classes and images, yet generalization is harder for CIFAR10 because the images are more complex than handwritten numbers. The second aspect is the rule between inputs and outputs: a random labelling of CIFAR10 can be learned by a neural network but offers no possibility of generalization [5]. Characterizing the structure of real-world data involves studying the interplay between these two aspects, an endeavour which has only been tackled by a few works [157–160].

5.1.1 Contribution

In this chapter we formulate a simple model of structured data, which simultaneously enables us to: (i) study the effect of overparametrization; (ii) analyze classification tasks; (iii) control the data structure in an interpretable manner. Our main analytical contribution, is the expression of the test error of a random feature model trained on this task. Our analysis is valid in the high-dimensional limit both for regression and classification tasks, although we focus here on the latter.

As a first application of our result, we study how the data structure and the loss function interplay to shape the test error, and in particular how they affect the double descent curve. We highlight behavioral

differences between square loss and logistic loss, in particular the fact that logistic loss generalizes better for easy tasks. We validate this insight via controlled experiments on the MNIST and CIFAR10 datasets.

Reproducibility The code used to produce the results presented in this chapter are available at `https://github.com/sdascoli/strong-weak`.

5.1.2 Related work

Replica method As in Chap. 3, the calculation performed in this chapter uses the replica method from statistical physics [109]. Although the latter is non-rigorous, several of its conjectures have been proven exact in the last decade [151, 161–163], including in settings very close to the model considered here [164, 165]. In particular, our result is very related to [165] which establishes rigorously the replica prediction in related learning problems. A rigorous proof of our replica results is within reach: it requires a small extension of [164] (to prove the anisotropic GET derived in Section D.2.2 of the Appendix) combined with the recent results of [165].

Data structure A few recent works investigate the roles of the loss and the structure in data in realistic setups where theoretically robust results are harder to obtain. Several works have studied the low intrinsic dimensionality of real-world data distributions and how it impacts sample complexity for supervised tasks [157–160]. The impact of the loss function is also an active research area: [166, 167] show that square loss can perform equally or better than the ubiquitous cross-entropy loss in realistic multi-class classification problems, if one rescales the weight of the correct class to emphasize its importance.

Although most theoretical studies of generalization focus on structureless data, a few exceptions exist. In the special case of random Fourier Feature regression, [168] derived the train and test error for a general input distribution. [169] achieved a similar result in the non-parametric setting of kernel regression, which can be viewed as the limiting case of random feature regression when the number of random features P goes to infinity.

Strong and weak features The strong and weak features model has recently been studied in the context of least-squares regression, both empirically [146] and theoretically [170–172]. Several intriguing observations emerge in strongly anisotropic setup: (i) several overfitting peaks can be seen [146, 147]; (ii) the optimal ridge regularization parameter γ can become negative [171, 173], as it becomes helpful to encourage the weights to have very different magnitudes; (iii) extra features acting as pure noise can play a beneficial role by inducing some implicit regularization [172].

The strong and weak features model also includes the setup studied in [174], called the spiked covariates model. The latter involves a small block of size D^η ($\eta < 1$) and a large "junk" block with no correlation with the labels. The question is then: can kernel methods learn to discard the junk features, hence "beat the curse of dimensionality" in the way neural networks do? The answer was shown to depend on the strength of the junk features: when the variance of these features is small, they are not problematic, and the kernel method ignores them, effectively learning a task of effective dimensionality $D^\eta \ll D$.

Clasification tasks The few works studying classification analytically have mostly focused on linear models, trained on linearly separable data [175–178] or gaussian mixtures [120, 179]. One of the

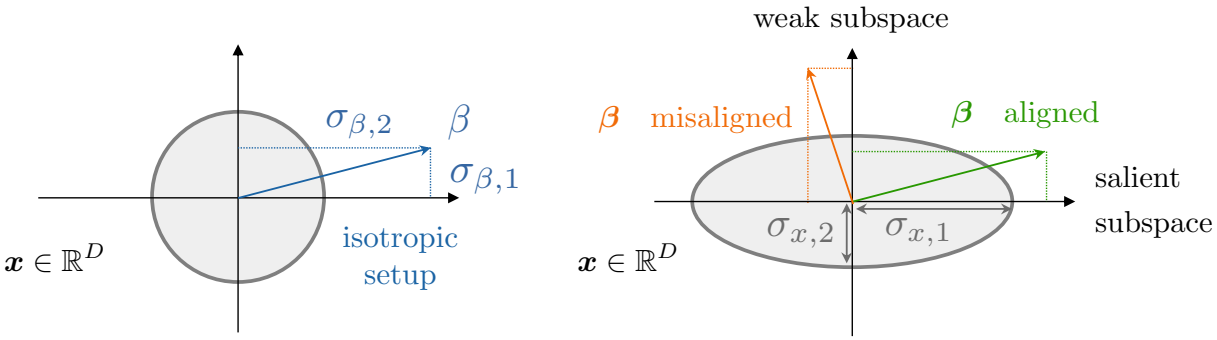


Figure 5.1: **Strong and weak features model.** Strong and weak features model considered here. Input space is decomposed into two subspaces with different variance in the anisotropic setting, with $\sigma_{x,1} > \sigma_{x,2}$. The *saliency* is given by the ratio $r_x = \sigma_{x,1}/\sigma_{x,2} > 1$. The labels are given by a linear teacher $y = \text{sign}(\boldsymbol{\beta} \cdot \boldsymbol{x}/\sqrt{D})$ and flipped with a certain probability Δ. We can adjust the *relevance* of the salient features by varying the ratio $r_\beta = \sigma_{\beta,1}/\sigma_{\beta,2}$. The *aligned* scenario ($r_\beta > 1$) is easier than the *misaligned* scenario ($r_\beta < 1$).

challenges in classification (in comparison to regression) is the large set of available loss functions [121, 180]. In the context of random feature models, [122] uses tools from statistical mechanics to derive the generalization loss of random features model for *any* loss function with i.i.d. Gaussian input.

More recently, [165] shows that this framework could extend to more complex data distributions and learned feature maps provided that key population covariances are estimated by Monte Carlo methods. Our work crucially builds on these contributions, by deriving a fully analytical analysis for a simple interpretable model of data structure, while also analyzing the effect of label flipping.

5.2 A solvable model of data structure

5.2.1 Setup

We focus again on the random features model described in the last two chapters:

$$\hat{y}_\mu = \sum_{i=1}^{P} a_i \sigma\left(\frac{\boldsymbol{\Theta}_i \cdot \boldsymbol{x}_\mu}{\sqrt{D}}\right), \tag{5.1}$$

where $\sigma(\cdot)$ is a pointwise activation function and $a_i \in \mathbb{R}$ are the second layer weights which trained by minimizing an ℓ_2-regularized loss on N training examples $\{\boldsymbol{x}_\mu \in \mathbb{R}^D\}_{\mu=1\dots N}$:

$$\hat{\boldsymbol{a}} = \underset{\boldsymbol{a}}{\text{argmin}} \left[\mathcal{L}(\boldsymbol{a}) + \frac{\lambda}{2}\|\boldsymbol{a}\|_2^2\right], \qquad \mathcal{L}(\boldsymbol{a}) = \sum_{\mu=1}^{N} \ell\left(y_\mu, \hat{y}_\mu\right). \tag{5.2}$$

Classification task This time, the target labels are given by a probabilistic teacher $y \sim \mathcal{P}_t(y|\boldsymbol{\beta} \cdot \boldsymbol{x})$ corresponding to the sign of a linear function possibly corrupted by label flipping:

$$y_\mu = \eta_\mu \, \text{sign}\left(\frac{\boldsymbol{\beta} \cdot \boldsymbol{x}_\mu}{\sqrt{D}}\right), \tag{5.3}$$

with $\eta_\mu = 1$ with probability $1-\Delta$ and $\eta_\mu = -1$ otherwise. We study two cases for the loss function: (i) square loss, i.e. $\ell(y,\hat{y}) = \frac{1}{2}(\hat{y}-y)^2$, and (ii) logistic loss, i.e. $\ell(y,\hat{y}) = \log(1+e^{-y\hat{y}})$[1]. The test error is computed as the 0-1 loss,

$$\epsilon_g = \mathbb{E}_{\boldsymbol{x},y}\left[\mathbb{1}_{\text{sign}(\hat{y}(\boldsymbol{x})),y(\boldsymbol{x})}\right]. \tag{5.4}$$

Strong and weak features To impose structure on the input space, we assume a Gaussian distribution of the inputs $\boldsymbol{x} \in \mathbb{R}^D$ with a block-structured covariance matrix:

$$\boldsymbol{x} \sim \mathcal{N}(0,\Sigma_x), \qquad \Sigma_x = \begin{bmatrix} \sigma_{x,1}\mathbb{I}_{\phi_1 D} & 0 & 0 \\ 0 & \sigma_{x,2}\mathbb{I}_{\phi_2 D} & 0 \\ 0 & 0 & \ddots \end{bmatrix},$$

$$\boldsymbol{\beta} \sim \mathcal{N}(0,\Sigma_\beta), \qquad \Sigma_\beta = \begin{bmatrix} \sigma_{\beta,1}\mathbb{I}_{\phi_1 D} & 0 & 0 \\ 0 & \sigma_{\beta,2}\mathbb{I}_{\phi_2 D} & 0 \\ 0 & 0 & \ddots \end{bmatrix}.$$

Our result presented in the rest of Section 5.2 is valid for an arbitrary number of blocks. In Section 5.3 we will focus for interpretability on the special case where we only have two blocks of sizes $\phi_1 D$ and $\phi_2 D$, with $\phi_1 + \phi_2 = 1$. We will typically be interested in the strongly anisotropic setup where the first subspace is much smaller ($\phi_1 \ll 1$), but potentially has higher *saliency* $r_x = \sigma_{x,1}/\sigma_{x,2} \gg 1$ or *relevance* $r_\beta = \sigma_{\beta,1}/\sigma_{\beta,2} \gg 1$ (see Fig. 5.1).

5.2.2 Main result

Using the replica method from statistical physics [109] and the Gaussian Equivalence Theorem (GET) [82, 151, 164, 181], we derive the test error in the high-dimensional limit where D, N and $P \to \infty$ with fixed ratios:

$$\lim_{N\to\infty} \epsilon_g = \frac{1}{\pi}\cos^{-1}\left(\frac{M}{\sqrt{\rho Q}}\right), \tag{5.5}$$

with

$$\rho = \sum_i \phi_i \sigma_{\beta,i}\sigma_{x,i}, \quad Q = \mu_1^2 \sum_i \sigma_{x,i} q_{s,i} + \mu_\star^2 q_w, \quad M = \mu_1 \sum_i \sigma_{x,i} m_{s,i}.$$

The parameters $\mu_1, \mu_\star$ are related to the activation function: denoting $\psi = \sum_i \phi_i \sigma_{x,i}$ and $z \sim \mathcal{N}(0,\psi)$, one has

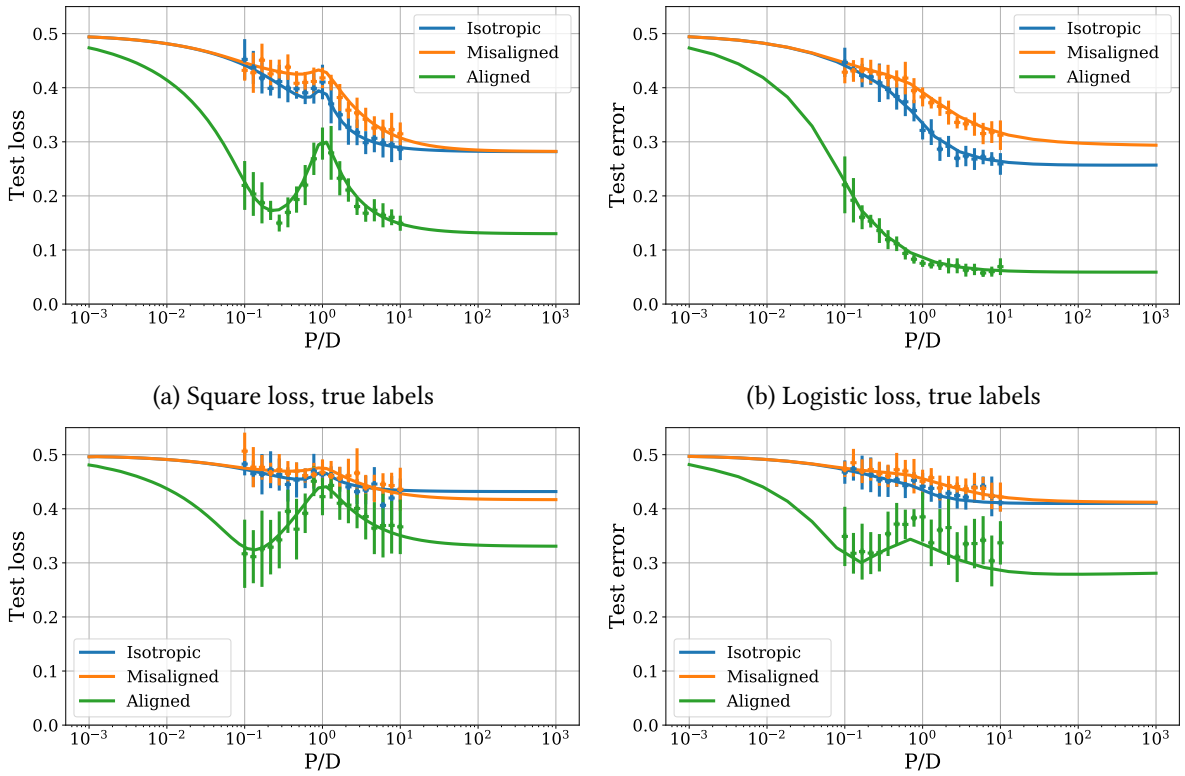
$$\mu_1 = \frac{1}{\psi}\mathbb{E}_z[z\sigma(z)]], \quad \mu_\star = \sqrt{\mathbb{E}_z\left[\sigma(z)^2\right] - \psi\mu_1^2}. \tag{5.6}$$

As in the previous chapter, these parameters quantify the degree of nonlinearity of the activation function, μ_1 encoding the linear part, and $\mu_\star$ quantifying the nonlinear part. Besides these constants and the ones defining the data structure ($\phi_i, \sigma_{\beta,i}, \sigma_{x,i}$), the key ingredients to obtain asymptotic errors are the so-called *order parameters* m_s, q_s, q_w. They correspond to the high-dimensional limit of the following expectations:

$$q_{s,i} = \lim_{D\to\infty}\frac{1}{D}\mathbb{E}_{\mathcal{P}}\left[\boldsymbol{s}_i \cdot \boldsymbol{s}_i\right], \quad q_w = \lim_{P\to\infty}\frac{1}{P}\mathbb{E}_{\mathcal{P}}\left[\hat{\boldsymbol{w}} \cdot \hat{\boldsymbol{w}}\right], m_{s,i} = \lim_{D\to\infty}\frac{1}{D}\mathbb{E}_{\mathcal{P}}\left[\boldsymbol{s}_i \cdot \boldsymbol{\beta}_i\right], \tag{5.7}$$

where $\boldsymbol{s} = \frac{1}{\sqrt{P}}\boldsymbol{\Theta}\hat{\boldsymbol{w}} \in \mathbb{R}^D$ and $\boldsymbol{s}_i, \boldsymbol{\beta}_i \in \mathbb{R}^{\phi_i D}$ denote the orthogonal projections of $\boldsymbol{s}$ and $\boldsymbol{\beta}$ onto subspace $i \in \{1,2\}$ and $\mathcal{P}$ denotes the joint distribution of all random quantities in the problem (the teacher weights, the random features and the training data).

[1] Our framework is valid for any convex loss function.



(a) Square loss, true labels

(b) Logistic loss, true labels

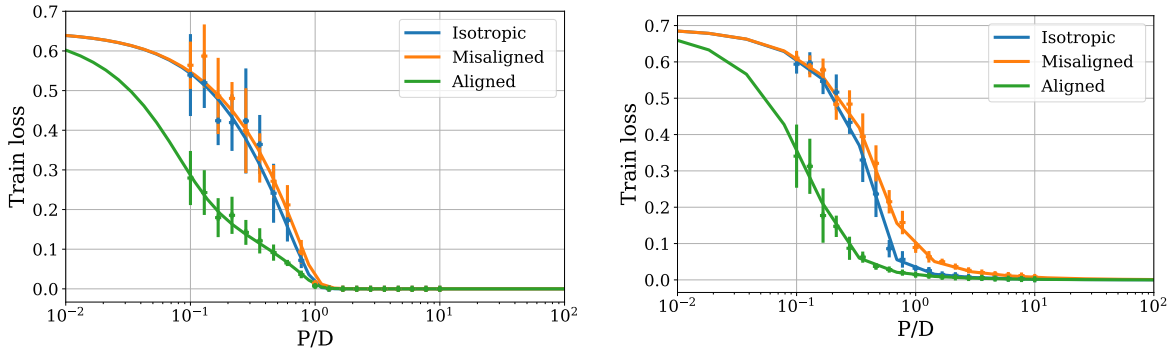
(c) Square loss, label flip probability $\Delta = 0.3$

(d) Logistic loss, label flip probability $\Delta = 0.3$

Figure 5.2: **Anisotropic data strongly affects the double descent curves.** Theoretical results (solid curves) and numerical results (dots with vertical bars denoting standard deviation over 10 runs) agree even at moderate size $D = 100$. We set $\sigma = \mathrm{Tanh}$, $\lambda = 10^{-3}$ and $N/D = 1$.

Intuition Intuitively, ρ is the variance of the outputs of the teacher, Q is the variance of the outputs of the student, and M is the covariance between the outputs of the student and those of the teacher. The test error is given by the "angle" between the teacher and the student, as expressed by Eq. 5.5. The order parameters of Eq. 5.7 are one of the outputs of the replica computation; they are obtained by solving a set of non-linear saddle-point equations (see App. D.2.5).

Steps and validity of the replica analysis The necessary steps of the derivation are detailed in App. D.2. In particular, (i) we obtain an anistropic extension of the GET, (ii) conduct random matrix analysis for block matrices and finally (iii) derive the analytical saddle-point equations which yield the values of the order parameters. Our result generalizes the strategy of [122] from isotropic to anisotropic data and additionally covers the effect of label flipping. In App. D.2.6, we also derive an expression for the training error.



(a) Square loss

(b) Logistic loss

Figure 5.3: **Structure of data affects the position of the interpolation threshold for logistic loss.** We depicted the train loss curves in the noiseless setup of Fig. 5.2, where $\sigma = \text{Tanh}$, $\lambda = 10^{-3}$, $N/D = 1$.

5.3 Effect of data structure and loss function on double descent

In this section, we investigate how the interplay between the data structure and the loss function shapes the train and test error curves. In the main text, we only examine parameter-wise curves, where we increase the number of parameters P at fixed number of data N. In App. D.1, we also examine the sample-wise dependency by plotting the train and test error in the entire (N, P) phase space.

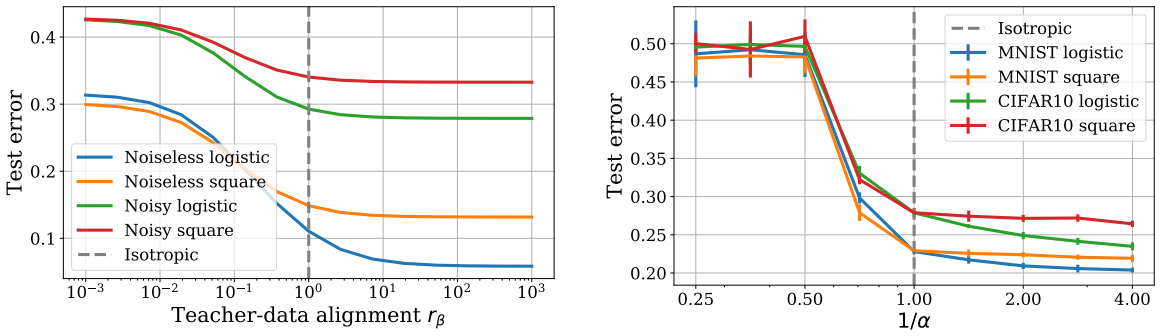
5.3.1 Synthetic data

Modulating the teacher-data alignment We compare three cases illustrated on Fig. 5.1. The first is the **isotropic** setup where $r_x = 1$ (blue curves in Fig. 5.2). In the two next setups, the data is anisotropic with a small subspace ($\phi_1 = 0.1$) of large variance and a large subspace ($\phi_2 = 0.9$) of small variance. The ratio of the variances $r_x = \sigma_{x,1}/\sigma_{x,2}$ is set to 10, and their values are chosen to keep the total variance of the inputs unchanged: $\frac{1}{D}\mathbb{E}[\|\boldsymbol{x}\|^2] = \phi_1\sigma_{x,1} + \phi_2\sigma_{x,2} = 1$.

We study two cases for the outputs: (i) the **aligned** scenario where the strong features are highly correlated with the labels ($r_\beta = \sigma_{\beta,1}/\sigma_{\beta,2} = 100$, green curves in Fig. 5.2); (ii) the **misaligned** scenario, where the strong features have low correlation with the labels ($r_\beta = 0.01$, orange curves in Fig. 5.2). In both cases, we choose the $\sigma_{\beta,i}$ such that the total variance of the teacher scores is unchanged: $\frac{1}{D}\mathbb{E}[(\boldsymbol{\beta}\cdot\boldsymbol{x})^2] = \phi_1\sigma_{x,1}\sigma_{\beta,1} + \phi_2\sigma_{x,2}\sigma_{\beta,2} = 1$.

Validity at finite size We begin by comparing our analytical predictions with the outcomes of numerical experiments, both for test loss (Fig. 5.2) and train loss (Fig. 5.3). The agreement is excellent even for moderately large dimensions $D = 100$. Note that the replica method, which relies on solving a set of scalar fixed point equations, is also computationally efficient. It allows here to probe ratios of P/D and N/D far beyond what is tractable by the numerics (P, D and N only appear in the replica equations through the values of the ratios N/P and P/D).

Effect of data structure on generalization Looking at Fig. 5.2, a first immediate observation is that strong teacher-data alignment makes the task easier: as number of parameters P increases the test loss



(a) Strong and weak features model

(b) Real data

Figure 5.4: **The easier the task, the wider the gap between logistic and square loss.** (a) Strong and weak feature model in the noiseless ($\Delta = 0$) and noisy ($\Delta = 0.3$) setups, where we make the task easier from left to right by increasing the alignment between the data and the teacher. (b) Real data (MNIST parity and CIFAR10 airplanes vs cars), where we make the task easier by decreasing the exponent α controlling the saliency of the top PCA components (see Sec. 5.3.2). In both cases, we considered an over-parametrized RF model ($P/D = 100$) learning from a moderate amount of data ($N/D = 1$), with $\sigma = \mathrm{Tanh}$ and $\lambda = 10^{-4}$.

drops earlier and eventually reaches a lower asymptotic value, both for square loss and logistic loss. In App. D.1, we show that the same phenomenon occurs when varying the number of samples N instead of the number of parameters P. These observations are in line with the results of [174] and show that relevant salient features make the problem low-dimensional with an effective dimension close to $\phi_1 D$.

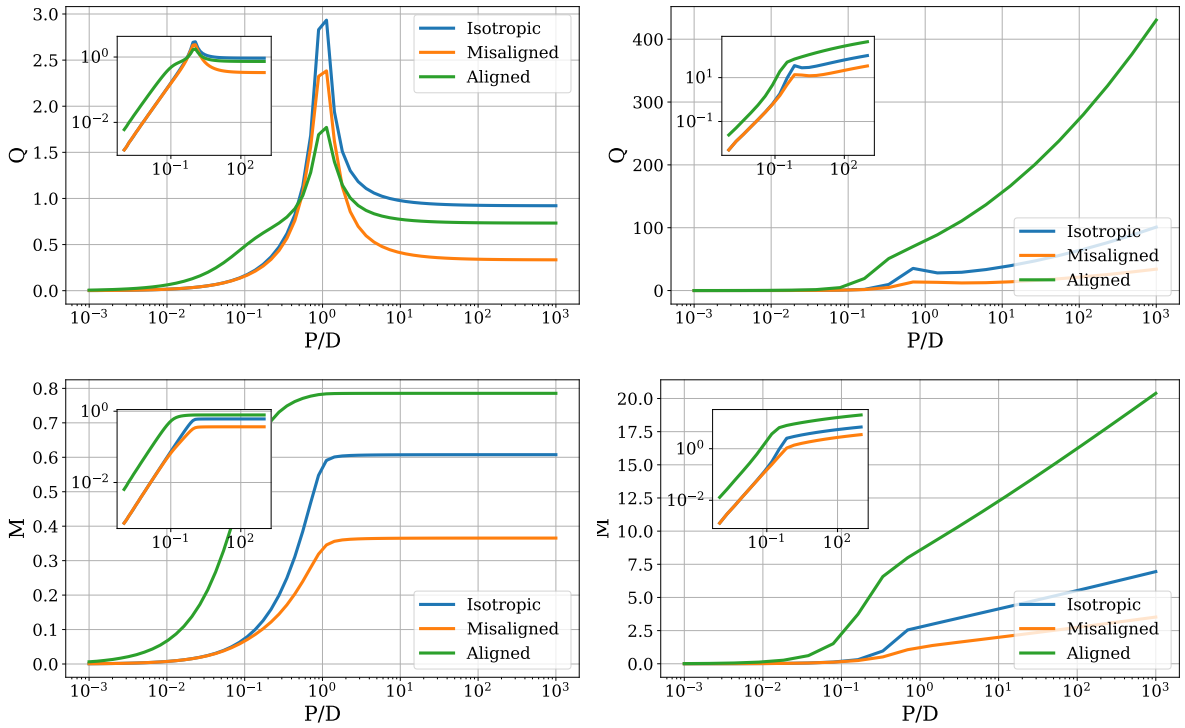
This aligned setup is the most akin to real-world tasks in which the most salient features of an image are often the most relevant to its recognition. In this sense, the impressive performance of kernel methods such as the Convolutional NTK on real-world datasets [98] can be associated with the anisotropy of the data: feature learning is not indispensable to beat the curse of dimensionality if the irrelevant features are weakly salient to begin with [160].

Conversely, misalignment generally makes the task harder and increases the value of the test loss. Note however that for square loss, an interesting crossover occurs in presence of noise (panel c): the irrelevant features are detrimental from small P, but become helpful at large P, as can be seen from the orange curve reaching a lower asymptotic value than the blue curve. We associate this to the phenomenon discovered for linear regression in [172], whereby adding noisy features acts as a form of implicit regularization.

Logistic is better than square loss Comparing the two loss functions, we observe two beneficial effects of using logistic loss rather than square loss.

First, we observe that the overfitting peak characteristic of the double descent curve which appears at $P = D$ for square loss is absent for logistic loss in the noiseless setting (Fig. 5.2), and vastly reduced in presence of noise (we use in both cases the same small amount of regularization for square and logistic loss). This suggests that the logistic loss exerts some form of implicit regularization, reducing the amount of overfitting.

Second, in the aligned and isotropic setups, the asymptotic test loss reached in the "kernel" regime



(a) Square loss

(b) Logistic loss

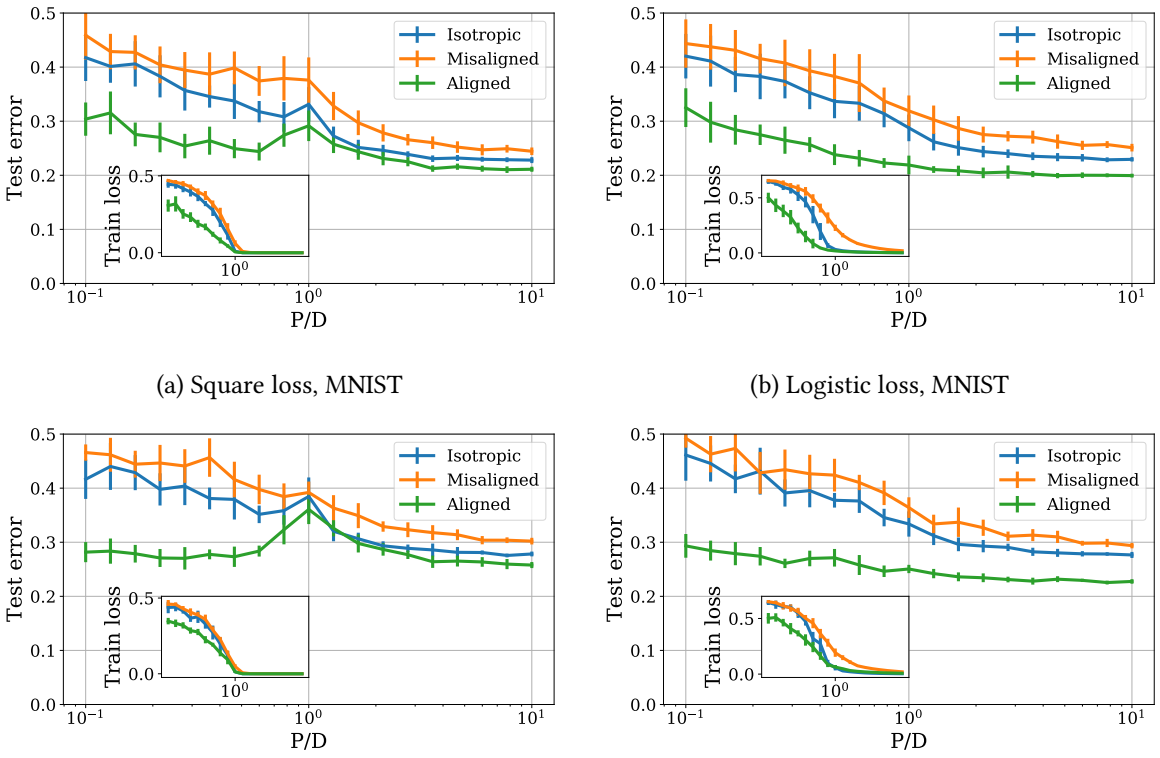
Figure 5.5: **Overparametrization causes the weights to diverge for logistic loss.** We depicted the order parameters Q and M, quantifying the variance of the outputs of the student and their covariance with the outputs of the teacher, in the noiseless setup of Fig. 5.2. *Insets:* log-log plot, showing the power-law asymptotic behaviors.

$P/D \to \infty$ is lower with logistic loss, especially in the aligned setup. To better highlight this phenomenon, we continuously vary the teacher-data alignment in Fig. 5.4(a) for an overparametrized model. Logistic loss performs similarly or worse than square loss at very small alignment, but outperforms square loss as soon as the alignment is sufficient. The gap between the two then grows as we increase alignment. In other words, logistic loss is particularly powerful on tasks made easy by the structure in the data.

The better ability of logistic loss to detect structure in the data is also reflected in the train loss curves of Fig. 5.3. For square loss, the IT, i.e. the point when the the train loss vanishes, occurs at $P = N$. For logistic loss, there is no IT strictly speaking since the train loss cannot be zero. However, one can define an effective threshold as the point where the training loss reaches the near-zero plateau. Notably, this effective threshold depends on the data structure: it is lower for the aligned setup, where the data is easier to fit, and higher for the misaligned setup, where the data is harder to fit.

Behavioral differences between losses Further understanding of the differences between logistic and square loss can be gained thanks to the replica approach which gives access to the order parameter Q and M defined in Eq. 5.6, see Fig. 5.5. For recall, Q corresponds to the variance of the outputs of the student $\hat{y}$ and M to their covariance with the linear scores of the teacher.

For square loss, Q and M increase and reach a finite value (with a peak in Q at the IT), reflecting



(a) Square loss, MNIST

(b) Logistic loss, MNIST

(c) Square loss, CIFAR10

(d) Logistic loss, CIFAR10

Figure 5.6: **Test error and train loss (inset) on realistic tasks.** *Top:* MNIST dataset with labels given by the digit parity. *Bottom:* CIFAR10 dataset with airplanes and cars. We synthetically reproduce the isotropic, aligned and misaligned scenarios by applying a PCA transformation to the inputs and tuning how salient the largest PCA features are compared to the smallest PCA features (see Sec. 5.3.2). We set $\sigma = \text{Tanh}$, $\lambda = 10^{-3}$ and $N/D = 1$.

the fact that the linearized estimator $\boldsymbol{\Theta}\boldsymbol{w}$ converges towards a fixed norm vector more or less correlated with the teacher vector $\boldsymbol{\beta}$ depending on the depending on the data structure. For logistic loss, M increases linearly with overparametrization and Q increases as a power law (see logarithmic insets), reflecting the fact the the estimator endlessly grows in the direction of the teacher vector [175] as the number of parameters increases. This growth appears to shield the peak observed in Q for the squared loss explaining the very mild double descent observed in Fig. 5.2. Interestingly, these quantities grow much faster in the aligned setup, where the estimator is more "confident" in its predictions, which also hints at the better performance of the logistic loss when the structure of data is favorable.

5.3.2 Realistic data

To examine the applicability of our results, we consider two realistic binary classification tasks: parity of digits in the MNIST dataset and airplanes vs cars in the CIFAR10 dataset. In both cases, we learn with an RF model in the same setup as described above. To control the alignment, we apply a PCA transformation to the inputs (keeping the top $D = 100$ components and discarding the rest), then

apply the following component-wise rescaling: $\boldsymbol{x}_i \to \boldsymbol{x}_i / \operatorname{std}(\boldsymbol{x}_i)^\alpha$, where $\operatorname{std}(\boldsymbol{x}_i)$ denotes the standard deviation of feature $\boldsymbol{x}_i$ over the whole training dataset, and the exponent α allows us to tune the saliency of the features:

- $\alpha < 1$ yields an **aligned** scenario, since the top PCA features are naturally the most relevant;
- $\alpha = 1$ yields the **isotropic** scenario, where all features have same variance;
- $\alpha > 1$ yields a **misaligned** scenario, since the strong features will become weak and the weak features become strong.

The corresponding train and test error curves are shown in Fig. 5.6 (we set $\alpha = 0$ for the aligned scenario and $\alpha = 1.5$ for the misaligned scenario). Remarkably, we recover many of the phenomenological features described previously. The test error drops earlier and reaches a lower asymptotic value in the aligned setup, and conversely reaches a higher value in the misaligned setup. We observe a double descent curve for square loss, but the peak is suppressed for logistic loss. The location of the IT depends on the teacher-data alignment for logistic loss, whereas it does not for square loss. Finally, logistic loss has a lower asymptotic error than square loss in the aligned setup, signalling that it is favorable for "easy" data distributions.

To strengthen the latter observation, we vary continuously the difficulty of the task by adjusting the exponent α and show the results in Fig. 5.4(b) (increasing α makes the task harder). As observed analytically in Fig. 5.4(a), the gap between square loss and logistic loss increases as we decrease α.

5.4 Conclusion

In this chapter, we studied how the loss function interplays with the data structure to shape the generalization curve of random feature models. Our results show strong behavioral differences between quadratic and logistic loss, the latter performing particularly well for easy datasets where most of the information comes from low-dimension projections of the inputs.

As a possible direction of future work, we conclude with the observation that our results, which apply to random feature (or lazy learning) tasks, appear in contrast with those of [166], which suggest that cross-entropy losses can be traded at no cost for quadratic losses in modern deep learning tasks, which are known to have low intrinsic dimensionality [159]. This opens up an interesting direction for future work: does feature learning help quadratic losses by better capturing the low-dimensional structure of the inputs, as suggested by [160]? Does the key difference reside in the multi-class nature of practical classification problems [167]?

Part II

From Architectural Constraints to Inductive Biases

Chapter 6

Finding the Needle in the Haystack: When do Convolutional Constraints help?

In the first part of this thesis, we studied the role of overparametrization in efficient generalization. Yet, our analysis was restricted to Fully-Connected Networks (FCNs), and ignored a key aspect of the learning model: its architecture. Convolutional Neural Networks (CNNs) are known to perform much better than FCNs on spatially structured data, as their architecture provides them with appropriate inductive biases (translation equivariance or invariance). Yet, it is unclear why FCNs fail to find convolutional solutions, which they could in principle represent. In this chapter, we attempt to elucidate this question by introducing a method to map a CNN to its equivalent FCN (denoted as eFCN), using the representation of convolutions as Toeplitz matrices. Such an embedding enables the comparison of CNN and FCN training dynamics directly in the FCN function space.

We use this method to test a new training protocol, which consists in training a CNN, embedding it to FCN space at a certain "relax time", then resuming the training in FCN space. We observe that for all relax times, the deviation from the CNN subspace is small, and the final performance reached by the eFCN is higher than that reachable by a standard FCN of same architecture. More surprisingly, for intermediate relax times, the eFCN outperforms the CNN it stemmed from, by combining the prior information of the CNN and the expressivity of the FCN in a complementary way. Although the practical interest of our protocol is limited by the very large size of the highly sparse eFCN, it demonstrates that inductive biases are mainly useful in the early stages of optimization, and can become restrictive beyond; this insight will be core to the practical applications presented in the next chapters.

6.1 Introduction

Architectural bias prevails as a major factor to explain good generalization in many tasks, such as visual classification – empirically, it is well-known that CNNs generalize well, but fully-connected models do not. The reasons underlying their success is rather clear: their architectural constraints incorporate prior information on images – namely, locality and translation equivariance (without pooling layers) or invariance (with pooling layers) of important features [182–184]. This inductive bias makes CNNs more sample-efficient and parameter-efficient [185, 186]

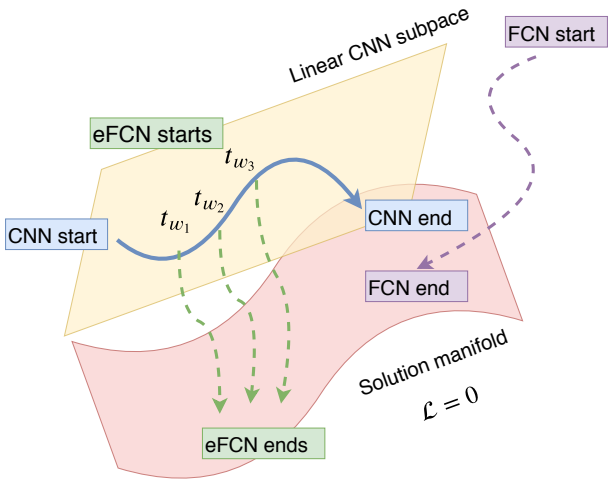


Figure 6.1: **Illustration of our experiment. White background:** ambient P-dimensional fully-connected space. **Yellow subspace:** linear, p-dimensional convolutional subspace. We have $p \ll P$. **Red manifold:** (near-) zero loss valued, (approximate-) solution set for a given training data. Note that it is a nontrivial manifold due to continuous symmetries (also, see the related work section on mode connectivity) and it intersects with the CNN subspace. **Blue path:** a CNN initialized and trained with the convolutional constraints. **Purple path:** a FCN model initialized and trained without the constraints. **Green paths:** Snapshots taken along the CNN training that are lifted to the ambient FCN space, and trained in the FCN space without the constraints.

Geometrically, these constraints define a low-dimensional subspace within the space of parameters of generic FCNs (this subspace is linear since the CNN constraints of weight sharing and locality are linear, see Figure 6.1.1 for a sketch of the core idea). Even though the optimization scheme is able to minimize the training loss with or without the constraints (for sufficiently over-parametrized models [5, 187]), the CNN subspace somehow a "better route" that navigates the loss landscape to basins with better generalization performance.

Why are such basins so difficult to reach in absence of constraints ? The objective of this chapter is to answer this question, for which many hypotheses are imaginable: (i) they are unreachable (e.g. surrounded by loss barriers), (ii) they occupy narrow basins which are missed by the optimization algorithm, (iii) they are too rare to be found.

Our results offer a new perspective on the success of the convolutional architecture: within FCN loss landscapes there exist rare basins associated to very good generalization, characterised not only by their width but rather by their distance to the CNN subspace. These can be accessed thanks to the CNN prior, and are otherwise missed in the usual training of FCNs.

6.1.1 Contribution

In this chapter, we show that the CNN prior plays a favorable role mostly at the *beginning* of optimization. Surprisingly, leaving this subspace at an appropriate time can result in a FCN with an equivalent or even better generalization than a CNN. Our numerical experiments suggest that the CNN subspace *as well as* its vicinity are good candidates for high-performance solutions. Furthermore, we observe a threshold distance from the CNN space beyond which the performance drops back down to the vanilla FCN accuracy level.

Reproducibility The code to reproduce the figures in this chapter is available at `https://github.com/sdascoli/anarchitectural-search`.

6.1.2 Related Work

Compression and expansion The relationship between CNNs and FCNs is an instance of trading-off prior information with expressivity within neural networks. There is abundant literature that explored the relationship between different neural architectures, for different purposes. One can roughly classify these works on whether they attempt to map a large model into a smaller one, or vice-versa.

In the first category, one of the earliest efforts to introduce structure within FCNs with the goal of improving generalization was the work of [188], in which the weights are regularized via a Mixture of Gaussians. Another highly popular line of work attempts to *distill* the "knowledge" of a large model (or an ensemble of models) into a smaller one [189–191], with the goal of improving both computational efficiency and generalization performance. Network pruning [192] and the recent "Lottery Ticket Hypothesis" [193] are other remarkable instances of the benefits of model compression.

In the second category, which is more directly related to our work, authors have attempted to build larger models by embedding small architectures into larger ones, such as the Net2Net model [194] or more evolved follow-ups [195]. In these works, however, the motivation is to accelerate learning by some form of knowledge transfer between the small model and the large one, whereas our motivation is to understand the specific role of architectural bias in generalization.

Landscape analysis Several works explored the link between sharpness of local minima and generalization [52, 196, 197], and [198] argued in terms of the volume of basins of attraction. [199, 200] introduce entropic SGD, a way of biasing dynamics towards wide minima.

The characterization of the loss landscape along paths connecting different models have been studied recently, e.g. in [201], [202], and [203]. The existence of rare basins leading to better generalization was found and highlighted in simple models in [204, 205].

Theoretical works In the infinite-width context, [206] study the role of translation equivariance of CNNs compared to FCNs. They find that in this limit, weight sharing does not play any role in the Bayesian treatment of CNNs, despite providing significant improvment in the finite-channel setup.

The links between generalization and the geometry and topology of the optimization landscape have been also extensively studied in recent times. [207] compare generalisation bounds between CNNs and FCNs, establishing a sample complexity advantage in the case of linear activations. [208, 209] obtain specific generalisation bounds for CNN architectures.

Follow-ups of this work Since its publication, this work has sparked several interesting follow-ups in the literature. Notably, [210] uses strong regularization and augmentation to bias fully-connected networks towards convolutional solutions (without using the convolutional initialization considered here). Another approach presented in [211] is to prune the weights of a pre-trained FCN in order to achieve the locality of a CNN. In a more theoretical perspective, [212] shows under which conditions convolutional configurations can be learnt in a data-driven manner.

The convolutional mapping described here also served as the core idea behind the ConViT, a state-of-the-art vision transformer which will be presented in the following chapter.

6.2 Methods

In this section, we present the method to map CNNs to FCNs and the experimental protocol we will follow for the rest of the chapter.

6.2.1 CNN to FCN embedding

In both FCNs and CNNs, each feature of a layer is calculated by applying a non-linearity to a weighted sum over the features of the previous layer (or over all the pixels of the image, for the first layer). CNNs are a particular type of FCNs, which make use of two key ingredients to reduce drastically the number of fitting parameters: locality and weight sharing.

- **Locality:** In FCNs, the sum is taken over all the features of the previous layer. In locally connected networks (LCNs), locality is imposed by restricting the sum to a small receptive field (a box of adjacent features of the previous layer). The set of weights of this restricted sum is called a filter. This procedure exploits the fact that the most relevant correlations in an image are general short-ranged.
- **Weight sharing:** CNNs are a particular type of LCNs where all the filters of a given channel use the same set of weights. This procedure ensures the translation equivariance of the features extracted (or translation invariance, when pooling layers are used).

When mapping a CNN to its equivalent FCN (eFCN), one obtains very sparse (due to locality) and redundant (due to weight sharing) weight matrices called Toeplitz matrices (see App. E.1 for some intuition on the mapping). This typically results in a large memory overhead as the eFCN of a simple CNN can take several orders of magnitude more space in the memory. Therefore, we present the core ideas on a simple 3-layer CNN on CIFAR-10 [1], and show similar results for AlexNet on CIFAR-100 in App. E.2.

In the mapping, all layers apart form the convolutional layers (ReLU, Dropout, MaxPool and fully-connected) are left unchanged except for proper reshaping. Each convolutional layer is mapped to a fully-connected layer. As a result, for a given CNN, we obtain its eFCN counterpart with an end-to-end fully-connected architecture which is functionally identical to the original CNN.

6.2.2 Experimental details

We are given input-label pairs for a supervised classification task, (x, y), with $x \in \mathbb{R}^D$ and y the index of the correct class for a given image x. The network, parametrized by $\boldsymbol{W}$, outputs $\hat{y} = f_{\boldsymbol{W}}(x)$. To distinguish between different architectures we denote the CNN weights by $\boldsymbol{W}^{CNN} \in \mathbb{R}^p$ and the eFCNs weights by $\boldsymbol{W}^{eFCN} \in \mathbb{R}^P$. Let's denote the embedding function described in Sec. 6.2 by $\Phi : \mathbb{R}^P \mapsto \mathbb{R}^p$ where $p \ll P$. Dropping the explicit input dependency for simplicity we have:

$$f_{\boldsymbol{W}^{CNN}} = f_{\Phi(\boldsymbol{W}^{CNN})} = f_{\boldsymbol{W}^{eFCN}}.$$

For the experiments, we prepare the CIFAR-10 dataset for training without data augmentation. The optimizer is set to stochastic gradient descent with a constant learning rate of 0.1 and a minibatch size of 250. We turn off the momentum and weight decay to simply focus on the stochastic gradient dynamics and we do not adjust the learning rate throughout the training process. In the following, we focus on

a convolutional architecture with 3 layers, 64 channels at each layer that are followed by ReLU and MaxPooling operators, and a single fully connected layer that outputs prediction probabilities. In our experience, this VanillaCNN strikes a good balance of simplicity and performance in that its equivalent FCN version does not suffer from memory issues yet it significantly outperforms any FCN model trained from scratch. We study the following protocol:

1. Initialize the VanillaCNN at $\boldsymbol{W}_{init}^{CNN}$ and train for 150 epochs. At the end of training $\boldsymbol{W}_{final}^{CNN}$ reaches $\sim 72\%$ test accuracy.

2. Along the way, save k snapshots of the weights at logarithmically spaced epochs. This provides k CNN points denoted by $\{\boldsymbol{W}_{t_0}^{CNN} = \boldsymbol{W}_{init}^{CNN}, \boldsymbol{W}_{t_1}^{CNN}, \dots, \boldsymbol{W}_{t_{k-1}}^{CNN}\}$.

3. Lift each one to its eFCN: $\{\Phi(\boldsymbol{W}_{t_0}^{CNN}), \dots, \Phi(\boldsymbol{W}_{t_{k-1}}^{CNN})\} = \{\boldsymbol{W}_{t_0}^{eFCN}, \dots, \boldsymbol{W}_{t_{k-1}}^{eFCN}\}$ (so that only p among a total of P parameters are non-zero).

4. Train these k eFCNs in the FCN space for 100 epochs in the same conditions, except a smaller learning rate of 0.01. We obtain k solutions $\{\boldsymbol{W}_{t_0,final}^{eFCN}, \dots, \boldsymbol{W}_{t_{k-1},final}^{eFCN}\}$.

5. For comparison, train a standard FCN (with the same architecture as the eFCNs but with the default PyTorch initialization) for 100 epochs in the same conditions as the eFCNs, and denote the resulting weights by $\boldsymbol{W}_{final}^{FCN}$. The latter reaches $\sim 55\%$ test accuracy.

This process gives us one CNN solution, one FCN solution, and k eFCN solutions that are labeled as

$$\boldsymbol{W}_{final}^{CNN}, \boldsymbol{W}_{final}^{FCN}, \text{ and } \{\boldsymbol{W}_{t_0,final}^{eFCN}, \dots, \boldsymbol{W}_{t_{k-1},final}^{eFCN}\} \tag{6.1}$$

which we analyze in the following subsections. Note that due to the difference in size between the CNN and the eFCNs, it unclear what learning rate would give a fair comparison. One solution, shown in App. E.2, is to use an adaptive learning rate optimizer such as Adam.

6.3 Results

In this section, we present the results of the experiments described in the previous section.

6.3.1 Performance and training dynamics of eFCNs

Our first aim is to characterize the training dynamics of eFCNs and study how their training evolution depends on their *relax time* $t_w \in \{t_0 = 0, t_1, \dots, t_{k-2}, t_{k-1} = 150\}$ (in epochs). When the architectural constraint is relaxed, the loss decreases monotonically to zero (see the left panel of Fig. 6.2). The initial losses are smaller for larger t_ws, as expected since those t_ws correspond to CNNs trained for longer. In the right panel of Fig. 6.2, we show a more surprising result: test accuracy increases monotonously in time for all t_ws, thus showing that *relaxing the constraints does not lead to overfitting or catastrophic forgetting*. Hence, from the point of view of the FCN space, it is not as if CNN dynamics took place on an unstable region from which the constraints of locality and weight sharing prevented from falling off. It is quite the contrary instead: the CNN dynamics takes place in a stable valley, and when the constraints are relaxed, the system keeps going down on the training surface and up in test accuracy, as opposed to falling back to the standard FCN dynamics.

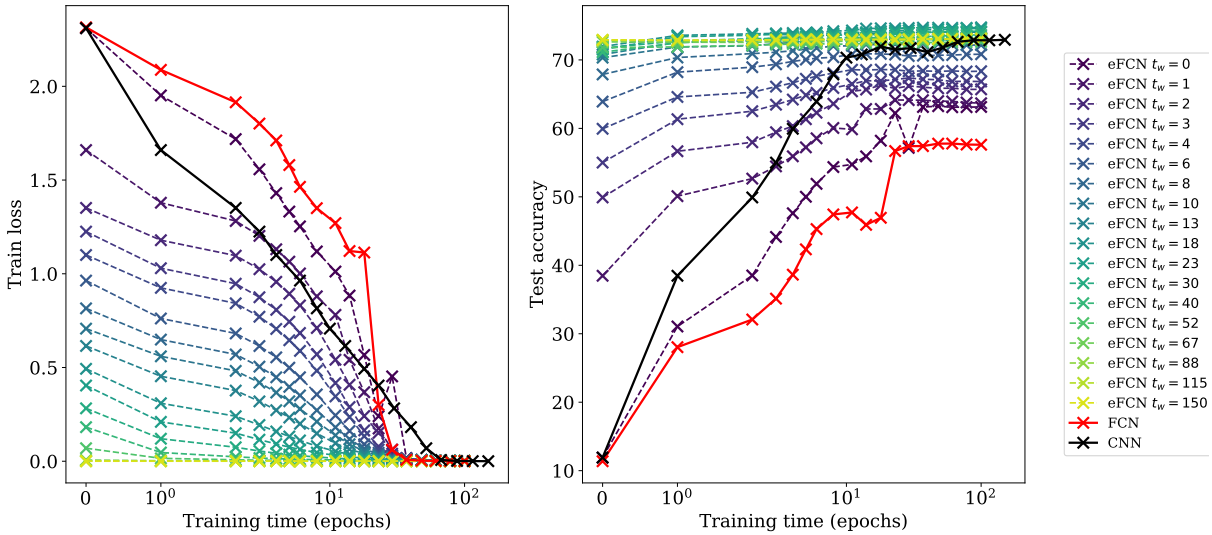


Figure 6.2: **The CNN subspace is stable within FCN space.** Training loss (**left**) and test accuracy (**right**) on CIFAR-100 vs. training time in logarithmic scale including the initial point. Different models are color coded as follows: the VanillaCNN is shown in black, standard FCN is in red, and the eFCNs with their relax time t_ws are indicated by the gradient ranging from purple to light green.

In Fig. 6.3 (left) we compare the final test accuracies reached by eFCN with the ones of the CNN and the standard FCN. We find two main results. First, the accuracy of the eFCN for $t_w = 0$ is approximately at 62.5%, well above the standard FCN result of 57.5%. This shows that imposing an *untrained* CNN prior is already enough to find a solution with much better performance than a standard FCN. Hence the CNN prior brings us to a good region of the landscape *to start with*. The second result, perhaps even more remarkable, is that at intermediate relax times ($t_w \sim 20$ epochs), the eFCN reaches—and exceeds—the final test accuracy reached by the CNN it stemmed from. This supports the idea that the constraints are mostly helpful for navigating the landscape *during the early stages of optimization*, and can be restrictive later on. Finally, at late relax times ($t_w > 50$ epochs), the eFCN is initialized close to the bottom of the landscape and has little room to move, hence the test accuracy falls back to that of the fully trained CNN.

Interestingly, very similar observations were made in a concurrent paper [213] which studies the effect of early relaxing of regularization procedures such as weight decay and data augmentation. They relate this phenomonelogy to an early "critical period" of learning during which regularization is most important.

6.3.2 A closer look at the landscape

A widespread idea in the deep learning literature is that the sharpness of the minima of the training loss is related to generalization performance [196, 214], the intuition being that flat minima reduce the impact of an offset between training loss and test loss. This motivates us to compare the first and second order properties of the landscape explored by the eFCNs and the CNNs they stem from. To do so, we investigate the norm of the gradient of the training loss, $|\nabla \mathcal{L}|$, and the top eigenvalue of the Hessian of the training loss, λ_{max}, in the central and right panels of Fig. 6.3 (we calculate the latter using a power

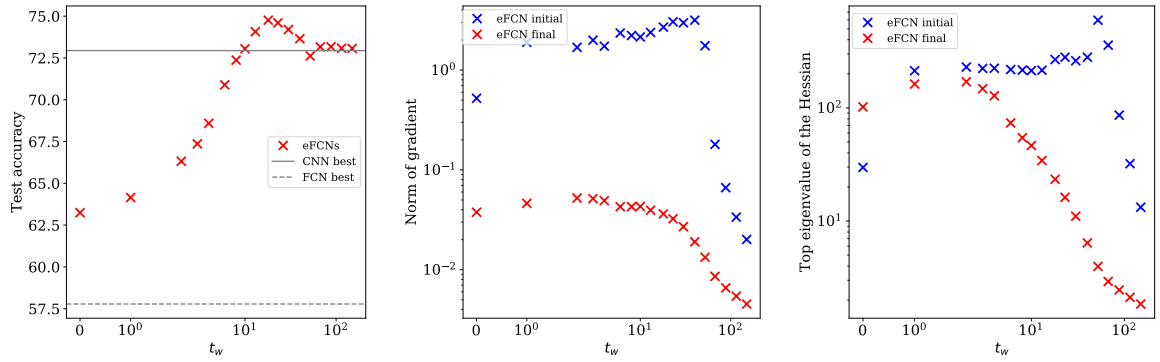


Figure 6.3: **Left**: Performance of eFCNs reached at the end of training (red crosses) compared to its counterpart for the best CNN accuracy (straight line) and the best FCN accuracy (dashed line). **Center**: Norm of the gradient for eFCNs at the beginning and at the end of training. **Right**: Largest eigenvalue of the Hessian for eFCNs at the beginning and at the end of training. In all figures the x-axis is the relax time t_w.

method).

We point out several interesting observations. First, the steepness ($|\nabla \mathcal{L}|$) and sharpness (λ_{max}) indicators increase then decrease during the training of the CNN (as analyzed in [215]), and display a maximum around $t_w \simeq 20$, which coincides with the relax time of best improvement for the eFCNs. Second, we see that after training the eFCNs, these indicators plummet by an order of magnitude, which is particularly surprising at very late relax time where it appeared in the left panel of Fig. 6.3 (see also 6.4) as if the eFCNs was hardly moving away from initialization. This supports the idea that when the constraints are relaxed, the extra degrees of freedom *lead us to wider basins*, possibly explaining the gain in performance.

6.3.3 How far does the eFCN escape from the CNN subspace?

A major question naturally arises: how far do the eFCNs move away from their initial condition? Do they stay close to the sparse configuration they were initialized in? To answer this question, we quantify how locality is violated once the constraints are relaxed (violation of weight sharing will be studied in Sec. 6.3.4). To this end, we consider a natural decomposition of the weights in the FCN space into two parts, $\boldsymbol{W} = (\boldsymbol{W}_{\text{local}}, \boldsymbol{W}_{\text{off-local}})$, where $\boldsymbol{W}_{\text{off-local}} = 0$ for an eFCN when it is initialized from a CNN. A visualization of these blocks may be found in Sec. A of the Appendix. We then study the ratio δ of the norm of the off-local weights to the total norm, $\delta(\boldsymbol{W}) = \frac{||\boldsymbol{W}_{\text{off-local}}||_2}{||\boldsymbol{W}||_2}$, which is a measure of the deviation of the model from the CNN subspace.

Fig. 6.4 (left) shows that the deviation δ at the end of eFCN training decreases monotonically with its relax time t_w. Indeed, the earlier we relax the constraints (and therefore the higher the initial loss of the eFCN) the further the eFCN escapes from the CNN subspace, as emphasized in Fig. 6.4 (middle). However, even at early relax times, the eFCNs stay rather close to the CNN subspace, since the ratio never exceeds 8%, whereas it is around 97% for a regular FCN (since the number of off-local weights is much larger than the number of local weights). This underlines the *persistence of the architectural bias under the stochastic gradient dynamics*.

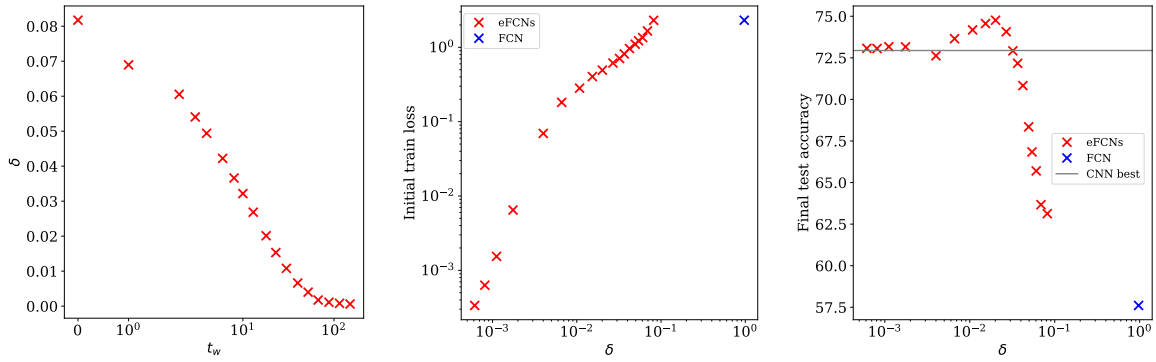


Figure 6.4: **Left:** relax time t_w of the eFCN vs. δ, the measure of deviation from the CNN subspace through the locality constraint, at the final point of eFCN training. **Centre:** δ vs. the initial loss value. **Right:** δ vs. final test accuracy of eFCN models. For reference, the blue point in the **middle** and **right** panels indicate the deviation measure for a standard FCN, where $\delta \sim 97\%$.

Fig. 6.4 (right) shows that when we move away from the CNN subspace, performance stays high then plummets down to FCN level. *This hints to a critical distance from the CNN subspace within which eFCNs behave like CNNs, and beyond which they fall back to the standard FCN regime.* We further explore this high performance vicinity of the CNN subspace using interpolations in weight space in App. E.3.

6.3.4 What role do the extra degrees of freedom play in learning?

How can the eFCN use the extra degrees of freedom to improve performance? From Fig. 6.5, we see that the off-local part of the eFCN is useless on its own (with the local part masked off), as its accuracy is equal to random guessing. However, when combined with the local part, it may greatly improve performance when the constraints are relaxed early enough. This hints to the fact that the local and off-local parts are performing complementary tasks.

To understand what tasks the two parts they are performing, we show in Fig. 6.6 a "filter" from the first layer of the eFCN (whose receptive field is of the size of the images since locality is relaxed). Note that each CNN filter gives rise to many eFCN filters: one for each position of the CNN filter on the image, since weight sharing is relaxed. Here we show the one obtained when the CNN filter (local block) is on the top left of the image. We see that off-local blocks stay orders of magnitude smaller than the local blocks, as expected from Sec. 6.3.3 where we saw that locality was almost conserved. We also see that local blocks hardly change during training, showing that weight sharing of the local blocks is also almost conserved.

More surprisingly, we see that for $t_w > 0$ distinctive shapes of the images are learned by the eFCN off-local blocks, which perform some kind of template-matching. Note that the silhouettes are particularly clear for the intermediate relax time (middle row), at which we know from Sec. 6.3.1 that the eFCN had the best improvement over the CNN. *Hence, the eFCN is combining template-matching with convolutional feature extraction in a complementary way.*

Note that by itself, template-matching is very inefficient for complicated and varied images such as those of the CIFAR-10 dataset. Hence it cannot be observed in standard FCNs, as shown in Fig. 6.7 where we reproduce the counterpart of Fig. 6.6 for the FCN in the left and middle images (they correspond to initial and final training times respectively). To reveal the silhouettes learned, we need to look at

the pixelwise difference between the two images, i.e. focus on the change due to training (this in unnecessary for the eFCN whose off-local weights started at zero). In the right panel of Fig. 6.7, we see that a loose texture emerges, however, it is not as sharp as that of the eFCN weights after training. Template-matching is only useful as a cherry-on-the-cake alongside more efficient learning procedures.

6.4 Conclusion

In this chapter, we examined the inductive bias of CNNs, and challenged the accepted view that FCNs are unable to generalize as well as CNNs on visual tasks. Specifically, we showed that the CNN prior is mainly useful during the early stages of training, to prevent the unconstrained FCN from falling prey of spurious solutions with poor generalization too early.

Our experimental results show that there exists a vicinity of the CNN subspace with high generalization properties, and one may even enhance the performance of CNNs by exploring it, if one relaxes the CNN constraints at an appropriate time during training. The extra degrees of freedom are used to perform complementary tasks which alone are unhelpful. This offers interesting theoretical perspectives, in relation to other high-dimensional estimation problems, such as in spiked tensor models [216], where a smart initialization, containing prior information on the problem, is used to provide an initial condition that bypasses the regions where the estimation landscape is "rough" and full of spurious minima.

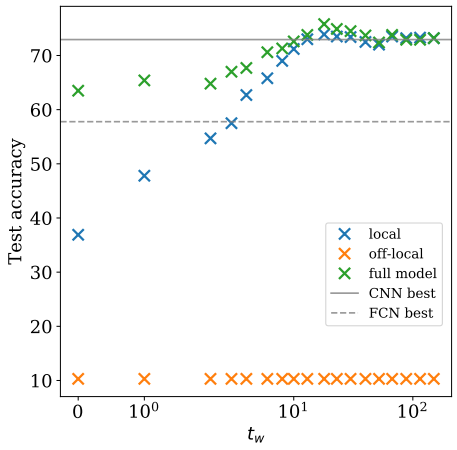


Figure 6.5: Contributions to the test accuracy of the local blocks (off-local blocks masked out), in orange, and off-local blocks (local blocks masked out), in blue. Combining them together yields a large gain in performance for the eFCN, in green.

On the practical front, despite the performance gains obtained, our algorithm remains highly impractical due to the large number of degrees of freedom required on our eFCNs. However, in the next chapter, we show that the same method can yield significant performance gains without any increase in number of parameters in a type of architecture which appeared after the publication of this work: vision transformers.

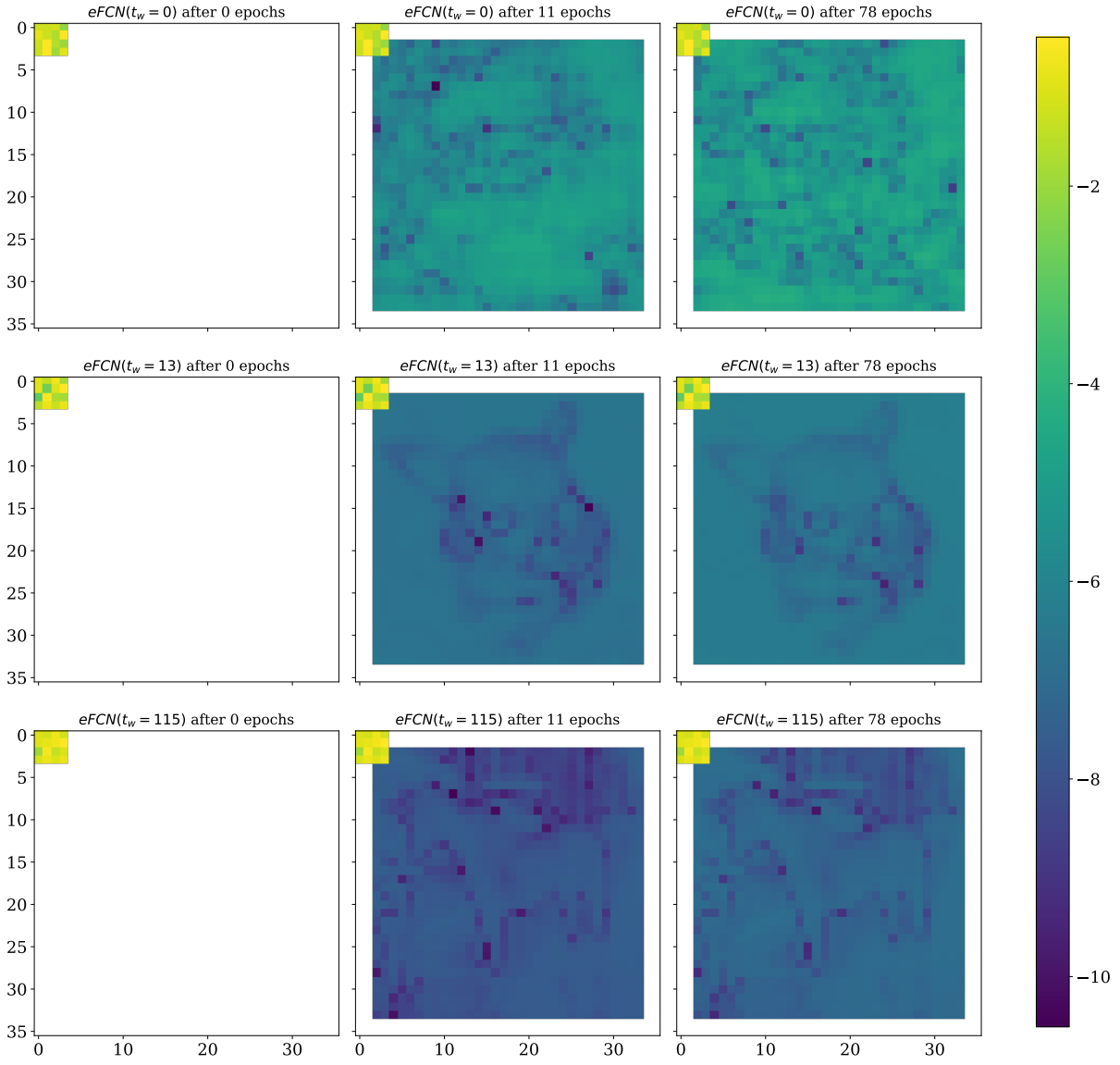


Figure 6.6: Heatmap of the weights of an eFCN "filter" from the first layer just at relax time (**left** column), after training for 11 epochs (**middle** column), and after training for 78 epochs (**right** column). The eFCNs were initialized at relax times $t_w = 0$ (**top** row), $t_w = 13$ (**middle** row), and $t_w = 115$ (**bottom** row). The colors indicate the natural logarithm of the absolute value of the weights. Note that the convolutional filters, in the top right, vary little and remain orders of magnitude larger than the off-local blocks, whereas the off-local blocks pick up strong signals from images as sharp silhouettes appear.

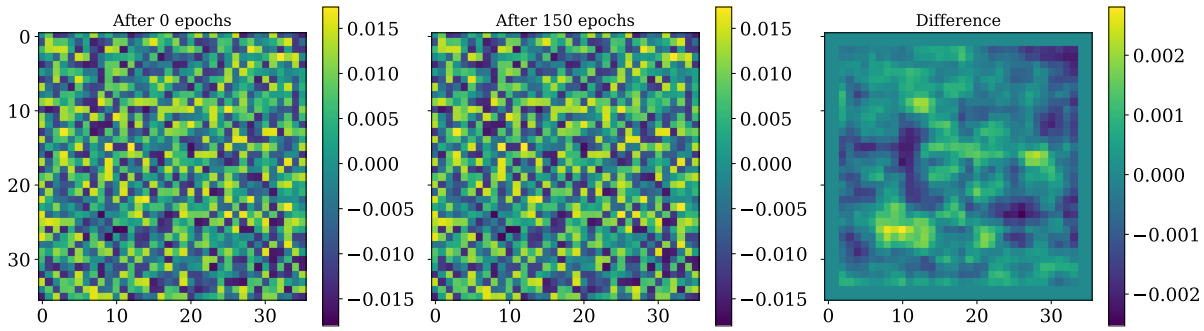


Figure 6.7: Same heatmap of weights as shown in Fig. 6.6 but for a standard FCN at a randomly initialized point (**left**) and after training for 150 epochs (**middle**). The pixelwise difference is shown on the **right** panel. A loose texture appears, but it is by no means as sharp as the silhouettes of the eFCNs.

Chapter 7

Improving Vision Transformers with Soft Convolutional Inductive Biases

Convolutional architectures have proven extremely successful for vision tasks. Their hard inductive biases enable sample-efficient learning, but can be restrictive as shown in the previous chapter. Vision Transformers (ViTs) rely on more flexible self-attention layers, and have recently outperformed CNNs for image classification. However, they require costly pre-training on large external datasets or distillation from pre-trained convolutional networks.

In this chapter, we ask the following question: is it possible to combine the strengths of these two architectures while avoiding their respective limitations? To this end, we introduce *gated positional self-attention* (GPSA), a form of positional self-attention which can be equipped with a "soft" convolutional inductive bias. Inspired by the results of the previous chapter, we initialize the GPSA layers to mimic the locality of convolutional layers, then give each attention head the freedom to escape locality by adjusting a *gating parameter* regulating the attention paid to position versus content information. The resulting convolutional-like ViT architecture, *ConViT*, outperforms the DeiT [217] on ImageNet, while offering a much improved sample efficiency. We further investigate the role of locality in learning by first quantifying how it is encouraged in vanilla self-attention layers, then analyzing how it is escaped in GPSA layers. We conclude by presenting various ablations to better understand the success of the ConViT.

7.1 Introduction

The success of deep learning over the last decade has largely been fueled by models with strong inductive biases, allowing efficient training across domains [184, 218]. The use of Convolutional Neural Networks (CNNs) [219, 220], which have become ubiquitous in computer vision since the success of AlexNet in 2012 [221], epitomizes this trend. As explained in the previous chapter, their inductive biases are hard-coded into their architecture in the form of two strong constraints on the weights: locality and weight sharing. Similarly, for sequence-based tasks, recurrent networks with hard-coded memory cells have been shown to simplify the learning of long-range dependencies (LSTMs) and outperform vanilla recurrent neural networks in a variety of settings [222–224].

However, the rise of models based purely on attention in recent years calls into question the necessity of hard-coded inductive biases. First introduced as an add-on to recurrent neural networks for Sequence-

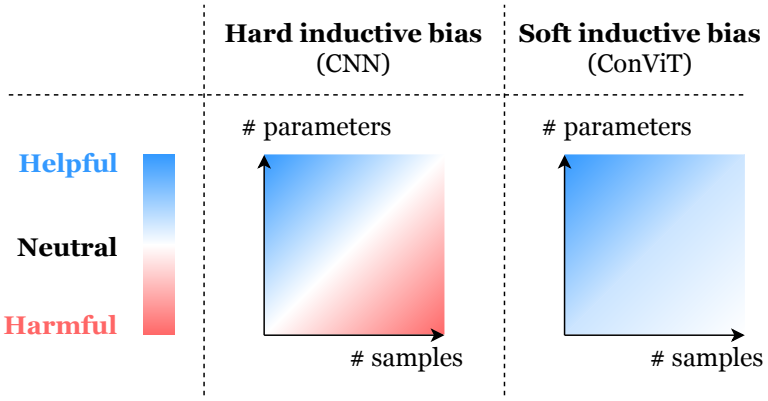


Figure 7.1: **Soft inductive biases can help models learn without being restrictive.** Hard inductive biases, such as the architectural constraints of CNNs, can greatly improve the sample-efficiency of learning, but can become constraining when the size of the dataset is not an issue. The soft inductive biases introduced by the ConViT avoid this limitation by vanishing away when not required.

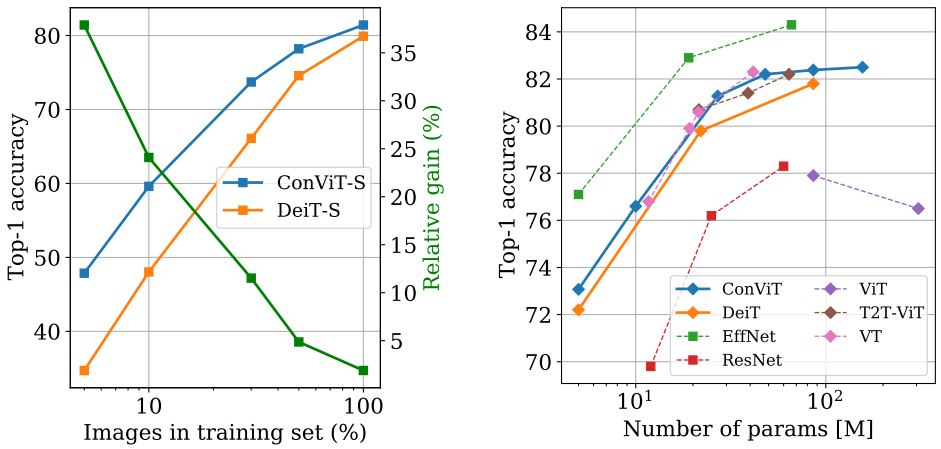
to-Sequence models [225], attention has led to a breakthrough in Natural Language Processing through the emergence of Transformer models, which rely solely on a particular kind of attention: Self-Attention (SA) [78]. The strong performance of these models when pre-trained on large datasets has quickly led to Transformer-based approaches becoming the default choice over recurrent models like LSTMs [226].

In vision tasks, the locality of CNNs impairs their ability to capture long-range dependencies, whereas attention does not suffer from this limitation. [227] and [228] leveraged this complementarity by augmenting convolutional layers with attention. More recently, [229] ran a series of experiments replacing some or all convolutional layers in ResNets with attention, and found the best performing models used convolutions in early layers and attention in later layers. The Vision Transformer (ViT), introduced by [14], entirely dispenses with the convolutional inductive bias by performing SA across embeddings of patches of pixels. The ViT is able to match or exceed the performance of CNNs but requires pre-training on vast amounts of data. More recently, the Data-efficient Vision Transformer (DeiT) [217] was able to reach similar performances without any pre-training on supplementary data, instead relying on Knowledge Distillation [190] from a convolutional teacher.

The success of the ViT demonstrates that while convolutional constraints can enable strongly sample-efficient training in the small-data regime, they can also become limiting as the dataset size is not an issue. In data-plentiful regimes, hard inductive biases can be overly restrictive and *learning* the most appropriate inductive bias can prove more effective. The practitioner is therefore confronted with a dilemma between using a convolutional model, which has a high performance floor but a potentially lower performance ceiling due to the hard inductive biases, or a self-attention based model, which has a lower floor but a higher ceiling. This dilemma leads to the following question: can one get the best of both worlds, and obtain the benefits of the convolutional inductive biases without suffering from its limitations (see Fig. 7.1)?

7.1.1 Contribution

In this chapter, we take a step towards bridging the gap between CNNs and Transformers, by presenting a new method to “softly" introduce a convolutional inductive bias into the ViT. The idea is to let each



(a) Sample efficiency

(b) Parameter efficiency

Figure 7.2: **The ConViT outperforms the DeiT both in sample and parameter efficiency**. *Left:* we compare the sample efficiency of our ConViT-S (see Tab. 7.1) with that of the DeiT-S by training them on restricted portions of ImageNet-1k, where we only keep a certain fraction of the images of each class. Both models are trained with the hyperparameters reported in [217]. We display the the relative improvement of the ConViT over the DeiT in green. *Right:* we compare the top-1 accuracies of our ConViT models with those of other ViTs (diamonds) and CNNs (squares) on ImageNet-1k. The performance of other models on ImageNet are taken from [77, 217, 230–232].

SA layer decide whether to behave as a convolutional layer or not, depending on the context. We make the following contributions:

1. We present a new form of SA layer, named *gated positional self-attention* (GPSA), which one can initialize as a convolutional layer. Each attention head then has the freedom to recover expressivity by adjusting a *gating parameter*.
2. We then perform experiments based on the DeiT [217], with a certain number of SA layers replaced by GPSA layers. The resulting Convolutional Vision Transformer (ConViT) outperforms the DeiT while boasting a much improved sample-efficiency (Fig. 7.2).
3. We analyze quantitatively how local attention is naturally encouraged in vanilla ViTs, then investigate the inner workings of the ConViT and perform ablations to investigate how it benefits from the convolution initialization.

Overall, our work demonstrates the effectiveness of "soft" inductive biases, especially in the low-data regime where the learning model is highly underspecified (see Fig. 7.1), and motivates the exploration of further methods to induce them.

Reproducibility We provide an open-source implementation of our method as well as pretrained models at the following address: `https://github.com/facebookresearch/convit`.

7.1.2 Related work

Hybrid models One successful approach in combining the advantages of CNNs and ViTs are “hybrid" models. These models, which interleave or combine convolutional and self-attention layers, have fueled successful results in a variety of tasks [229, 231, 233–238]. Another approach is that of Knowledge Distillation [190], which has recently been applied to transfer the inductive bias of a convolutional teacher to a student transformer [217]. While these two methods offer an interesting compromise, they forcefully induce convolutional inductive biases into the Transformers, potentially affecting the Transformer with their limitations.

Soft inductive biases The restrictiveness of hard locality constraints has been proven by [239]. A breadth of approaches have been taken to imbue CNN architectures with nonlocality [231, 240–242]. An opposite approach is to induce a convolutional inductive bias in non-local architectures. The previous chapter has shown a method to do so for fully-connected networks, and mentioned several concurrent approaches [210, 211].

The idea explored in this chapter is inspired by this approach, and the formalized relationship between SA and convolution. Indeed, [243] showed that a SA layer with N_h heads can express a convolution of kernel size $\sqrt{N}_h$, if each head focuses on one of the pixels in the kernel patch. By investigating the qualitative aspect of attention maps of models trained on CIFAR-10, it is shown that SA layers with relative positional encodings naturally converge towards convolutional-like configurations, suggesting that some degree of convolutional inductive bias is desirable.

Follow-ups of this work Since the publication of this work, a plethora of alternative methods have been proposed to combine the advantages of self-attention and convolution [244–251], see [252] for a recent review.

The ConViT has been integrated in the Timm [253] package for state-of-the-art computer vision architectures, and has been applied to many different tasks, ranging from volcano detection [30] to skin lesion classification [31].

7.2 Methods

In this section, we describe the methods used in the ConViT architecture.

7.2.1 A crash course on self-attention

We begin by introducing the basics of SA layers, and show how positional attention can allow SA layers to express convolutional layers.

Multi-head self-attention The attention mechanism is based on a trainable associative memory with (key, query) vector pairs. A sequence of L_1 “query" embeddings $\boldsymbol{Q} \in \mathbb{R}^{L_1 \times D_h}$ is matched against another sequence of L_2 “key" embeddings $\boldsymbol{K} \in \mathbb{R}^{L_2 \times D_h}$ using inner products. The result is an attention matrix whose entry (ij) quantifies how semantically “relevant" $\boldsymbol{Q}_i$ is to $\boldsymbol{K}_j$:

$$\boldsymbol{A} = \text{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{D_h}}\right) \in \mathbb{R}^{L_1 \times L_2}, \tag{7.1}$$

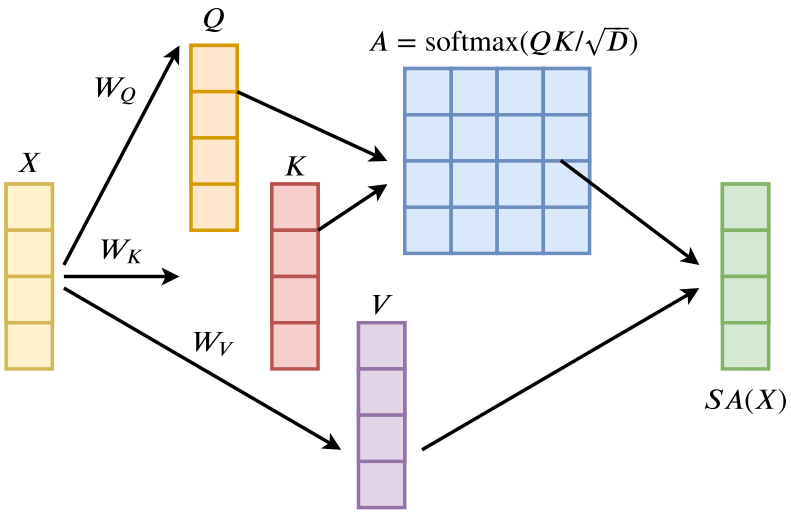


Figure 7.3: **The self-attention mechanism.**

where $(\text{softmax}\,[\boldsymbol{X}])_{ij} = \boldsymbol{e}^{\boldsymbol{X}_{ij}} / \sum_k e^{\boldsymbol{X}_{ik}}$.

Self-attention is a special case of attention where a sequence is matched to itself, to extract the semantic dependencies between its parts, and is illustrated in Fig. 7.3. In the ViT, the queries and keys are linear projections of patches of 16×16 pixels, $\boldsymbol{X} \in \mathbb{R}^{L \times D_{\text{emb}}}$. Hence, we have $\boldsymbol{Q} = \boldsymbol{W}_{qry}\boldsymbol{X}$ and $\boldsymbol{K} = \boldsymbol{W}_{key}\boldsymbol{X}$, where $\boldsymbol{W}_{key}, \boldsymbol{W}_{qry} \in \mathbb{R}^{D_{\text{emb}} \times D_h}$.

Multi-head SA layers use several self-attention heads in parallel to allow the learning of different kinds of interdependencies. They take as input a sequence of L embeddings of dimension $D_{\text{emb}} = N_h D_h$, and output a sequence of L embeddings of the same dimension through the following mechanism:

$$\text{MHSA}(\boldsymbol{X}) := \underset{h \in [N_h]}{\text{concat}} \left[\text{SA}_h(\boldsymbol{X})\right] \boldsymbol{W}_{out}, \tag{7.2}$$

where $\boldsymbol{W}_{out} \in \mathbb{R}^{D_{\text{emb}} \times D_{\text{emb}}}$. Each self-attention head h performs the following operation:

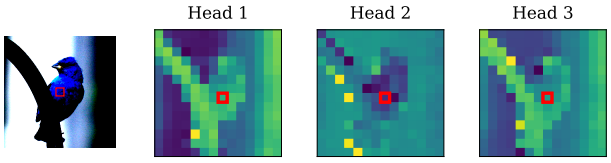
$$\text{SA}_h(\boldsymbol{X}) := \boldsymbol{A}^h \boldsymbol{X} \boldsymbol{W}^h_{val}, \tag{7.3}$$

where $\boldsymbol{W}^h_{val} \in R^{D_{\text{emb}} \times D_h}$ is the *value* matrix.

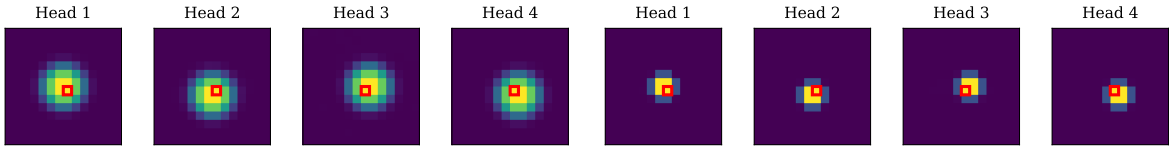
Positional self-attention In the vanilla form of Eq. 7.1, SA layers are permutation invariance hence position-agnostic: they do not know how the patches are located according to each other. To incorporate positional information, there are several options. One is to add some positional information to the input at embedding time, before propagating it through the SA layers: [14] use this approach in their ViT. Another possibility is to replace the vanilla SA with positional self-attention (PSA), using encodings $\boldsymbol{r}_{ij}$ of the relative position of patches i and j [229]:

$$\boldsymbol{A}^h_{ij} := \text{softmax}\left(\boldsymbol{Q}^h_i \boldsymbol{K}^{h\top}_j + \boldsymbol{v}^{h\top}_{pos} \boldsymbol{r}_{ij}\right) \tag{7.4}$$

Each attention head uses a trainable embedding $\boldsymbol{v}^h_{pos} \in \mathbb{R}^{D_{\text{pos}}}$, and the relative positional encodings $\boldsymbol{r}_{ij} \in \mathbb{R}^{D_{\text{pos}}}$ only depend on the distance between pixels i and j, denoted denoted by a two-dimensional vector $\boldsymbol{\delta}_{ij}$.



(a) Input (b) Standard initialization



(c) Convolutional initialization, strength $\alpha = 0.5$ (d) Convolutional initialization, strength $\alpha = 2$

Figure 7.4: **Positional self-attention layers can be initialized as convolutional layers.** (a): Input image from ImageNet, where the query patch is highlighted by a red box. (b),(c),(d): attention maps of an untrained SA layer (b) and those of a PSA layer using the convolutional-like initialization scheme of Eq. 7.5 with two different values of the locality strength parameter, α (c, d). Note that the shapes of the image can easily be distinguished in (b), but not in (c) or (d), when the attention is purely positional.

Positional self-attention as a generalized convolution In [243], it is shown that a multi-head PSA layer with N_h heads and learnable relative positional encodings (Eq. 7.4) of dimension $D_{\text{pos}} \geq 3$ can express any convolutional layer of filter size $\sqrt{N_h} \times \sqrt{N_h}$, by setting the following:

$$\begin{cases} \boldsymbol{v}_{pos}^h := -\alpha^h \left(1, -2\Delta_1^h, -2\Delta_2^h, 0, \dots 0\right) \\ \boldsymbol{r}_{\boldsymbol{\delta}} := \left(\|\boldsymbol{\delta}\|^2, \delta_1, \delta_2, 0, \dots 0\right) \\ \boldsymbol{W}_{qry} = \boldsymbol{W}_{key} := \boldsymbol{0}, \quad \boldsymbol{W}_{val} := \boldsymbol{I} \end{cases} \tag{7.5}$$

In the above,

- The *center of attention* $\boldsymbol{\Delta}^h \in \mathbb{R}^2$ is the position to which head h pays most attention to, relative to the query patch. For example, in Fig. 7.4(c), the four heads correspond, from left to right, to $\boldsymbol{\Delta}^1 = (-1, 1), \boldsymbol{\Delta}^2 = (-1, -1), \boldsymbol{\Delta}^3 = (1, 1), \boldsymbol{\Delta}^4 = (1, -1)$.
- The *locality strength* $\alpha^h > 0$ determines how focused the attention is around its center $\boldsymbol{\Delta}^h$ (it can also by understood as the "temperature" of the softmax in Eq. 7.1). When α^h is large, the attention is focused only on the patch(es) located at $\boldsymbol{\Delta}^h$, as in Fig. 7.4(d); when α^h is small, the attention is spread out into a larger area, as in Fig. 7.4(c).

Thus, the PSA layer can achieve a strictly convolutional attention map by setting the centers of attention $\boldsymbol{\Delta}^h$ to each of the possible positional offsets of a $\sqrt{N_h} \times \sqrt{N_h}$ convolutional kernel, and sending the locality strengths α^h to some large value.

7.2.2 Our approach

Building on the insight of [243], we introduce the ConViT, a variant of the ViT [14] obtained by replacing some of the SA layers by a new type of layer which we call *gated positional self-attention* (GPSA) layers.

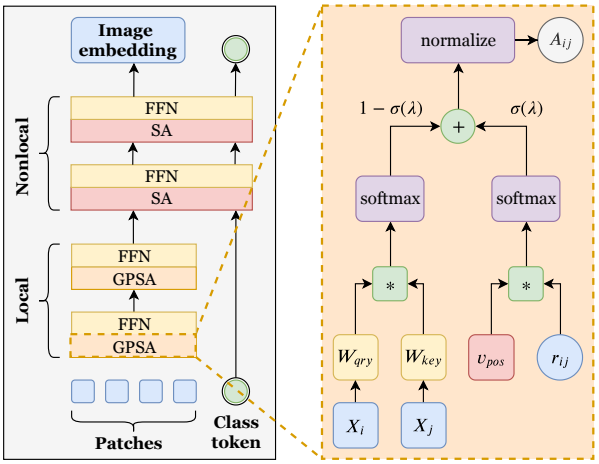


Figure 7.5: **Architecture of the ConViT.** The ConViT (left) is a version of the ViT in which some of the self-attention (SA) layers are replaced with gated positional self-attention layers (GPSA; right). Because GPSA layers involve positional information, the class token is concatenated with hidden representation after the last GPSA layer. In this chapter, we typically take 10 GPSA layers followed by 2 vanilla SA layers. FFN: feedforward network (2 linear layers separated by a GeLU activation); W_{qry}: query weights; W_{key}: key weights; v_{pos}: attention center and span embeddings (learned); r_{qk}: relative position encodings (fixed); λ: gating parameter (learned); σ: sigmoid function.

The core idea is to enforce the "informed" convolutional configuration of Eqs. 7.5 in the GPSA layers at initialization, then let them decide whether to stay convolutional or not. However, the standard parameterization of PSA layers (Eq. 7.4) suffers from two limitations, which lead us two introduce two modifications.

Adaptive attention span The first caveat in PSA is the vast number of trainable parameters involved, since the number of relative positional encodings $\boldsymbol{r}_\delta$ is quadratic in the number of patches. This led some authors to restrict the attention to a subset of patches around the query patch [229], at the cost of losing long-range information.

To avoid this, we leave the relative positional encodings $\boldsymbol{r}_\delta$ fixed, and train only the embeddings $\boldsymbol{v}_{pos}^h$ which determine the center and span of the attention heads; this approach relates to the *adaptive attention span* introduced in [254] for Language Transformers. The initial values of $\boldsymbol{r}_\delta$ and $\boldsymbol{v}_{pos}^h$ are given by Eq. 7.5, where we take $D_{\text{pos}} = 3$ to get rid of the useless zero components. Thanks to $D_{\text{pos}} \ll D_h$, the number of parameters involved in the positional attention is negligible compared to the number of parameters involved in the content attention. This makes sense, as content interactions are inherently much simpler to model than positional interactions.

Positional gating The second issue with standard PSA is the fact that the content and positional terms in Eq. 7.4 are potentially of different magnitudes, in which case the softmax will ignore the smallest of the two. In particular, the convolutional initialization scheme discussed above involves highly concentrated attention scores, i.e. high-magnitude values in the softmax. In practice, we observed that

using a convolutional initialization scheme on vanilla PSA layers gives a boost in early epochs, but degrades late-time performance as the attention mechanism lazily ignores the content information (see App. F.1).

To avoid this, GPSA layers sum the content and positional terms *after* the softmax, with their relative importances governed by a learnable *gating* parameter λ_h (one for each attention head). Finally, we normalize the resulting sum of matrices (whose terms are positive) to ensure that the resulting attention scores define a probability distribution. The resulting GPSA layer is therefore parametrized as follows (see also Fig. 7.5):

$$\text{GPSA}_h(\boldsymbol{X}) := \text{normalize}\left[\boldsymbol{A}^h\right]\boldsymbol{X}\boldsymbol{W}_{val}^h \tag{7.6}$$

$$\begin{aligned}\boldsymbol{A}_{ij}^h := &(1-\sigma(\lambda_h))\,\text{softmax}\left(\boldsymbol{Q}_i^h\boldsymbol{K}_j^{h\top}\right)\\ &+\sigma(\lambda_h)\,\text{softmax}\left(\boldsymbol{v}_{pos}^{h\top}\boldsymbol{r}_{ij}\right),\end{aligned} \tag{7.7}$$

where $(\text{normalize}\,[\boldsymbol{A}])_{ij} = \boldsymbol{A}_{ij}/\sum_k \boldsymbol{A}_{ik}$ and $\sigma : x \mapsto 1/(1+e^{-x})$ is the sigmoid function. By setting the gating parameter λ_h to a large positive value at initialization, one has $\sigma(\lambda_h) \simeq 1$: the GPSA bases its attention purely on position, dispensing with the need of setting $\boldsymbol{W}_{qry}$ and $\boldsymbol{W}_{key}$ to zero as in Eq. 7.5. However, to avoid the ConViT staying stuck at $\lambda_h \gg 1$, we initialize $\lambda_h = 1$ for all layers and all heads.

7.2.3 Experimental details

Architecture The ViT slices input images of size 224 into 16×16 non-overlapping patches of 14×14 pixels and embeds them into vectors of dimension $D_{\text{emb}} = 64N_h$ using a convolutional stem. It then propagates the patches through 12 blocks which keep their dimensionality constant. Each block consists in a SA layer followed by a 2-layer Feed-Forward Network (FFN) with GeLU activation, both equipped with residual connections. The ConViT is simply a ViT where the first 10 blocks replace the SA layers by GPSA layers with a convolutional initialization.

Similar to language Transformers like BERT [226], the ViT uses an extra "class token", appended to the sequence of patches to predict the class of the input. Since this class token does not carry any positional information, the SA layers of the ViT do not use positional attention: the positional information is instead injected to each patch before the first layer, by adding a learnable positional embedding of dimension D_{emb}. As GPSA layers involve positional attention, they are not well suited for the class token approach. We solve this problem by appending the class token to the patches after the last GPSA layer, similarly to what is done in [255] (see Fig. 7.5)[1].

For fairness, and since they are computationally cheap, we keep the absolute positional embeddings of the ViT active in the ConViT. However, as shown in App. F.6, the ConViT relies much less on them, since the GPSA layers already use relative positional encodings. Hence, the absolute positional embeddings could easily be removed, dispensing with the need to interpolate the embeddings when changing the input resolution (the relative positional encodings simply need to be resampled according to Eq. 7.5, as performed automatically in our open-source implementation).

Training We based our ConViT on the DeiT [217], a hyperparameter-optimized version of the ViT which has been open-sourced[2]. Thanks to its ability to achieve competitive results without using any

[1] We also experimented incorporating the class token as an extra patch of the image to which all heads pay attention to at initialization, but results were worse than concatenating the class token after the GPSA layers (not shown).

[2] https://github.com/facebookresearch/deit

Name	Model	N_h	D_{emb}	Size	Flops	Speed	Top-1	Top-5
Ti	DeiT	3	192	6M	1G	1442	72.2	-
	ConViT	4	192	6M	1G	734	**73.1**	**91.7**
Ti+	DeiT	4	256	10M	2G	1036	75.9	93.2
	ConViT	4	256	10M	2G	625	**76.7**	**93.6**
S	DeiT	6	384	22M	4.3G	587	79.8	-
	ConViT	9	432	27M	5.4G	305	**81.3**	**95.7**
S+	DeiT	9	576	48M	10G	480	79.0	94.4
	ConViT	9	576	48M	10G	382	**82.2**	**95.9**
B	DeiT	12	768	86M	17G	187	81.8	-
	ConViT	16	768	86M	17G	141	**82.4**	**95.9**
B+	DeiT	16	1024	152M	30G	114	77.5	93.5
	ConViT	16	1024	152M	30G	96	**82.5**	**95.9**

Table 7.1: **Performance of the models considered, trained from scratch on ImageNet.** Speed is the number of images processed per second on a Nvidia Quadro GP100 GPU at batch size 128. Top-1 accuracy is measured on ImageNet-1k test set without distillation (see App. F.2 for distillation). The results for DeiT-Ti, DeiT-S and DeiT-B are reported from [217].

external data, the DeiT both an excellent baseline and relatively easy to train: the largest model (DeiT-B) only requires a few days of training on 8 GPUs.

To mimic 2×2, 3×3 and 4×4 convolutional filters, we consider three different ConViT models with 4, 9 and 16 attention heads (see Tab. 7.1). Their number of heads are slightly larger than the DeiT-Ti, ConViT-S and ConViT-B of [217], which respectively use 3, 6 and 12 attention heads. To obtain models of similar sizes, we use two methods of comparison.

- To establish a direct comparison with [217], we lower the embedding dimension of the ConViTs to $D_{\text{emb}}/N_h = 48$ instead of 64 used for the DeiTs. Importantly, *we leave all hyperparameters (scheduling, data-augmentation, regularization) presented in [217] unchanged* in order to achieve a fair comparison. The resulting models are named ConViT-Ti, ConViT-S and ConViT-B.

- We also trained DeiTs and ConViTs using the same number of heads and $D_{\text{emb}}/N_h = 64$, to ensure that the improvement due to ConViT is not simply due to the larger number of heads [255]. This leads to slightly larger models denoted with a "+" in Tab. 7.1. To maintain stable training while fitting these models on 8 GPUs, we lowered the learning rate from 0.0005 to 0.0004 and the batch size from 1024 to 512. These minimal hyperparameter changes lead the DeiT-B+ to perform less well than the DeiT-S+, which is not the case for the ConViT, suggesting a higher stability to hyperparameter changes.

7.3 Results

In this section, we present the results of the ConViT on image classification tasks.

Train size	Top-1			Top-5		
	DeiT	ConViT	Gap	DeiT	ConViT	Gap
5%	34.8	**47.8**	37%	57.8	**70.7**	22%
10%	48.0	**59.6**	24%	71.5	**80.3**	12%
30%	66.1	**73.7**	12%	86.0	**90.7**	5%
50%	74.6	**78.2**	5%	91.8	**93.8**	2%
100%	79.9	**81.4**	2%	95.0	**95.8**	1%

Table 7.2: **The convolutional inductive bias strongly improves sample efficiency.** We compare the top-1 and top-5 accuracy of our ConViT-S with that of the DeiT-S, both trained using the original hyperparameters of the DeiT [217], as well as the relative improvement of the ConViT over the DeiT. Both models are trained on a subsampled version of ImageNet-1k, where we only keep a variable fraction (leftmost column) of the images of each class for training.

7.3.1 Performance of the ConViT

In Tab. 7.1, we display the top-1 accuracy achieved by these models evaluated on the ImageNet test set after 300 epochs of training, alongside their number of parameters, number of flops and throughput. Each ConViT outperforms its DeiT of same size and same number of flops by a margin. Importantly, although the positional self-attention does slow down the throughput of the ConViTs, they also outperform the DeiTs at equal throughput. For example, The ConViT-S+ reaches a top-1 of 82.2%, outperforming the original DeiT-B with less parameters and higher throughput. Without any tuning, the ConViT also reaches high performance on CIFAR100, see App. F.3 where we also report learning curves.

Note that our ConViT is compatible with the distillation methods introduced in [217] at no extra cost. As shown in App. F.2, hard distillation improves performance, enabling the hard-distilled ConViT-S+ to reach 82.9% top-1 accuracy, on the same footing as the hard-distilled DeiT-B with half the number of parameters. However, while distillation requires an additional forward pass through a pre-trained CNN at each step of training, ConViT has no such requirement, providing similar benefits to distillation without additonal computational requirements.

7.3.2 Sample efficiency of the ConViT

In Tab. 7.2, we investigate the sample-efficiency of the ConViT in a systematic way, by subsampling each class of the ImageNet-1k dataset by a fraction $f = \{0.05, 0.1, 0.3, 0.5, 1\}$ while multiplying the number of epochs by $1/f$ so that the total number images presented to the model remains constant. As one might expect, the top-1 accuracy of both the DeiT-S and its ConViT-S counterpart drops as f decreases. However, the ConViT suffers much less: while training on only 10% of the data, the ConViT reaches 59.5% top-1 accuracy, compared to 46.5% for its DeiT counterpart.

This result can be directly compared to [256], which after testing several thousand convolutional models reaches a top-1 accuracy of 56.4%; the ConViT is therefore highly competitive in terms of sample efficiency. These findings confirm our hypothesis that the convolutional inductive bias is most helpful on small datasets, as depicted in Fig. 7.1.

7.3.3 Investigating the role of locality

In this section, we demonstrate that locality is naturally encouraged in standard SA layers, and examine how the ConViT benefits from locality being imposed at initialization.

SA layers are pulled towards locality We begin by investigating whether the hypothesis that PSA layers are naturally encouraged to become "local" over the course of training [243] holds for the vanilla SA layers used in ViTs, which do not benefit from positional attention. To quantify this, we define a measure of "nonlocality" by summing, for each query patch i, the distances $\|\boldsymbol{\delta}_{ij}\|$ to all the key patches j weighted by their attention score $\boldsymbol{A}_{ij}$. We average the number obtained over the query patch to obtain the nonlocality metric of head h, which can then be averaged over the attention heads to obtain the nonlocality of the whole layer ℓ:

$$
\begin{aligned}
D_{loc}^{\ell,h} &:= \frac{1}{L}\sum_{ij} \boldsymbol{A}_{ij}^{h,\ell}\|\boldsymbol{\delta}_{ij}\|, \\
D_{loc}^{\ell} &:= \frac{1}{N_h}\sum_{h} D_{loc}^{\ell,h}
\end{aligned}
\tag{7.8}
$$

Intuitively, D_{loc} is the number of patches between the center of attention and the query patch: the further the attention heads look from the query patch, the higher the nonlocality.

In Fig. 7.6 (left panel), we show how the nonlocality metric evolves during training across the 12 layers of a DeiT-S trained for 300 epochs on ImageNet. During the first few epochs, the nonlocality falls from its initial value in all layers, confirming that the DeiT becomes more "convolutional". During the later stages of training, the nonlocality metric stays low for lower layers, and gradually climbs back up for upper layers, revealing that the latter capture long range dependencies, as observed for language Transformers [254].

These observations are particularly clear when examining the attention maps (Fig. F.7 of the Appendix), and point to the beneficial effect of locality in lower layers. In Fig. F.2 of the Appendix., we also show that the nonlocality metric is lower when training with distillation from a convolutional network as in [217], suggesting that the locality of the teacher is partly transferred to the student [257].

GPSA layers escape locality In the ConViT, strong locality is imposed at the beginning of training in the GPSA layers thanks to the convolutional initialization. In Fig. 7.6 (right panel), we see that this local configuration is escaped throughout training, as the nonlocality metric grows in all the GPSA layers. However, the nonlocality at the end of training is lower than that reached by the DeiT, showing that some information about the initialization is preserved throughout training. Interestingly, the final nonlocality does not increase monotonically throughout the layers as for the DeiT. The first layer and the final layers strongly escape locality, whereas the intermediate layers (particularly the second layer) stay more local.

To gain more understanding, we examine the dynamics of the gating parameters in Fig. 7.7. We see that in all layers, the average gating parameter $\mathbb{E}_h\sigma(\lambda_h)$ (in black), which reflects the average amount of attention paid to positional information versus content, decreases throughout training. This quantity reaches 0 in layers 6-10, meaning that positional information is practically ignored. However, in layers 1-5, some of the attention heads keep a high value of $\sigma(\lambda_h)$, hence take advantage of positional information. Interestingly, the ConViT-Ti only uses positional information up to layer 4, whereas the

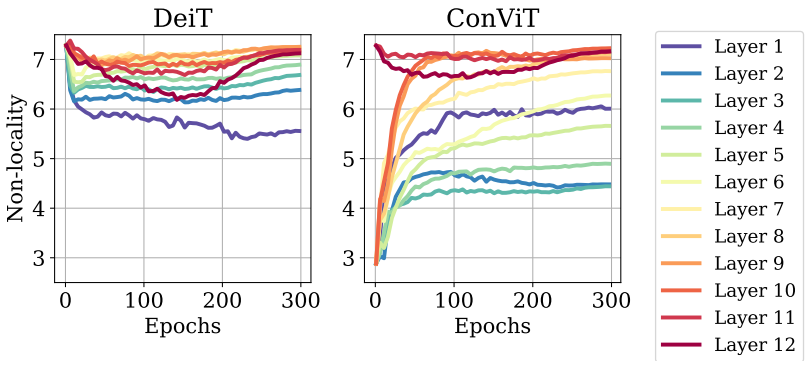


Figure 7.6: **SA layers try to become local, GPSA layers escape locality.** We plot the nonlocality metric defined in Eq. 7.8, averaged over a batch of 1024 images: the higher, the further the attention heads look from the query pixel. We trained the DeiT-S and ConViT-S for 300 epochs on ImageNet. Similar results for DeiT-Ti/ConViT-Ti and DeiT-B/ConViT-B are shown in App. F.4.

Ref.	Train gating	Conv init	Train GPSA	Use GPSA	Full data	10% data
a (ConViT)	✓	✓	✓	✓	**82.2**	**59.7**
b	✗	✓	✓	✓	82.0	57.4
c	✓	✗	✓	✓	81.4	56.9
d	✗	✗	✓	✓	81.6	54.6
e (DeiT)	✗	✗	✗	✗	79.1	47.8
f	✗	✓	✗	✓	78.6	54.3
g	✗	✗	✗	✓	73.7	44.8

Table 7.3: **Gating and convolutional initialization play nicely together.** We ran an ablation study on the ConViT-S+ trained for 300 epochs on the full ImageNet training set and on 10% of the training data. From the left column to right column, we experimented freezing the gating parameters to 0, removing the convolutional initialization, freezing the GPSA layers and removing them altogether.

ConViT-B uses it up to layer 6 (see App. F.4), suggesting that larger models - which are more under-specified - benefit more from the convolutional prior. These observations highlight the usefulness of the gating parameter in terms of interpretability.

The inner workings of the ConViT are further revealed by the attention maps of Fig. 7.8, which are obtained by propagating an embedded input image through the layers and selecting a query patch at the center of the image[3]. In layer 10, (bottom row), the attention maps of DeiT and ConViT look qualitatively similar: they both perform content-based attention. In layer 2 however (top row), the attention maps of the ConViT are more varied: some heads pay attention to content (heads 1 and 2) whereas other focus mainly on position (heads 3 and 4). Among the heads which focus on position, some stay highly localized (head 4) whereas others broaden their attention span (head 3). The interested reader can find more attention maps in App. F.5.

[3]We do not show the attention paid to the class token in the SA layers.

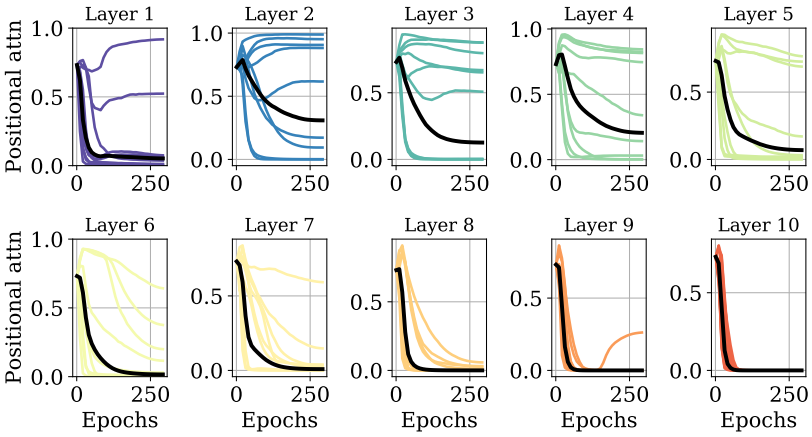


Figure 7.7: **The gating parameters reveal the inner workings of the ConViT.** For each layer, the colored lines (one for each of the 9 attention heads) quantify how much attention head h pays to positional information versus content, i.e. the value of $\sigma(\lambda_h)$, see Eq. 7.7. The black line represents the value averaged over all heads. We trained the ConViT-S for 300 epochs on ImageNet. Similar results for ConViT-Ti and ConViT-B are shown in App. F.4.

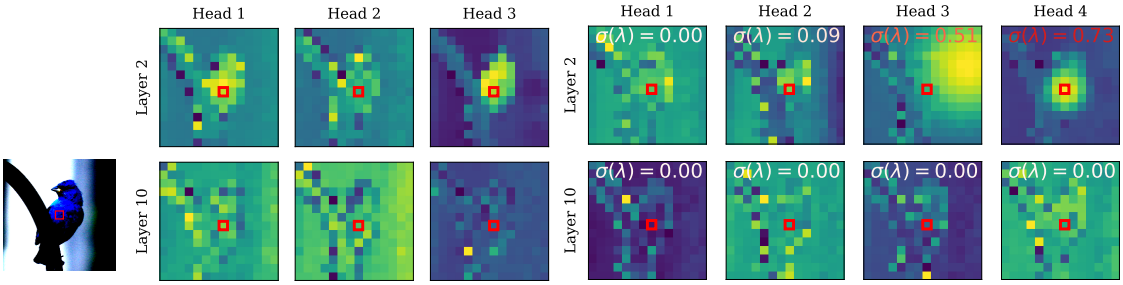
Strong locality is desirable We next investigate how the performance of the ConViT is affected by two important hyperparameters of the ConViT: the *locality strength*, α, which determines how focused the heads are around their center of attention, and the number of SA layers replaced by GPSA layers. We examined the effects of these hyperparameters on ConViT-S, trained on the first 100 classes of ImageNet. As shown in Fig. 7.9(a), final test accuracy increases both with the locality strength and with the number of GPSA layers; in other words, the more convolutional, the better.

In Fig. 7.9(b), we show how performance at various stages of training is impacted by the presence of GPSA layers. We see that the boost due to GPSA is particularly strong during the early stages of training: after 20 epochs, using 9 GPSA layers leads the test-accuracy to almost double, suggesting that the convolution initialization gives the model a substantial "head start". This speedup is of practical interest in itself, on top of the boost in final performance.

Ablation study In Tab. 7.3, we present an ablation on the ConViT, denoted as **[a]**. We experiment removing the positional gating **[b]**[4], the convolutional initialization **[c]**, both gating and the convolutional initialization **[d]**, and the GPSA altogether (**[e]**, which leaves us with a plain DeiT).

Surprisingly, on full ImageNet, GPSA without gating **[d]** already brings a substantial benefit over the DeiT (+2.5), which is mildly increased by the convolutional initialization (**[b]**, +2.9). As for gating, it helps a little in presence of the convolutional initialization (**[a]**, +3.1), and is unhelpful otherwise. These mild improvements due to gating and convolutional initialization (likely due to performance saturation above 80% top-1) become much clearer in the low data regime. Here, GPSA alone brings +6.8, with an extra +2.3 coming from gating, +2.8 from convolution initialization and +5.1 with the two together, illustrating their complementarity.

[4]To remove gating, we freeze all gating parameters to $\lambda = 0$ so that the same amount of attention is paid to content and position.



(a) Input (b) DeiT (c) ConViT

Figure 7.8: **The ConViT learns more diverse attention maps**. *Left:* input image which is embedded then fed into the models. The query patch is highlighted by a red box and the colormap is logarithmic to better reveal details. *Center:* attention maps obtained by a DeiT-Ti after 300 epochs of training on ImageNet. *Right:* Same for ConViT-Ti. In each map, we indicated the value of the gating parameter in a color varying from white (for heads paying attention to content) to red (for heads paying attention to position). Attention maps for more images and heads are shown in App. F.5.

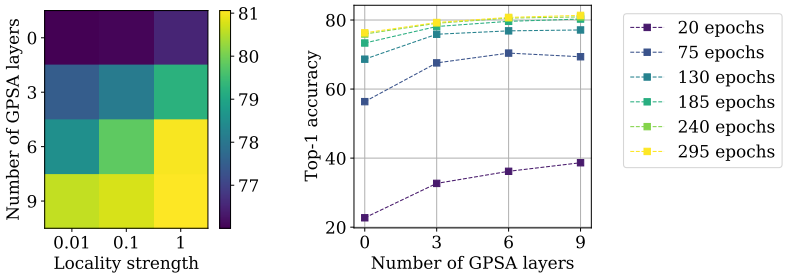


Figure 7.9: **The beneficial effect of locality**. *Left:* As we increase the locality strength (i.e. how focused each attention head is its associated patch) and the number of GPSA layers of a ConViT-S+, the final top-1 accuracy increases significantly. *Right:* The beneficial effect of locality is particularly strong in the early epochs.

We also investigated the performance of the ConViT with all GPSA layers frozen, leaving only the FFNs to be trained in the first 10 layers. As one could expect, performance is strongly degraded in the full data regime if we initialize the GPSA layers randomly (**[f]**, -5.4 compared to the DeiT). However, the convolutional initialization remarkably enables the frozen ConViT to reach a very decent performance, almost equalling that of the DeiT (**[e]**, -0.5). In other words, replacing SA layers by random "convolutions" hardly impacts performance. In the low data regime, the frozen ConViT even outperforms the DeiT by a margin (+6.5). This naturally begs the question: is attention really key to the success of ViTs [15, 258, 259]?

7.4 Conclusion

The present work investigates the importance of initialization and inductive biases in learning with vision transformers. By showing that one can take advantage of convolutional constraints in a soft way,

we merge the benefits of architectural priors and expressive power. The result is a simple recipe that improves trainability and sample efficiency, without increasing model size or requiring any tuning.

Our approach can be summarized as follows: instead of interleaving convolutional layers with SA layers as done in hybrid models, let the layers decide whether to be convolutional or not by adjusting a set of gating parameters. More generally, combining the biases of varied architectures and letting the model choose which ones are best for a given task could become a promising direction, reducing the need for greedy architectural search while offering higher interpretability.

Another direction which will be explored in the next chapter is the following: if SA layers benefit from being initialized as random convolutions, could one reduce even more drastically their sample complexity by initializing them as pre-trained convolutions, just like we initialized FCNs from trained CNNs in the previous chapter?

Chapter 8

Transformed CNNs: Recasting Pre-trained Convolutional Networks as Transformers

One aspect we did not mention in the previous chapter is training time. Although the ConViT provides us with a flexible architecture, able to combine the sample efficiency of CNNs with the expressivity of ViTs, it remains self-attention based, which imposes a bottleneck in computational speed. In this chapter, we push the same idea further by introducing a method to initialize self-attention layers as pre-trained (rather than random) convolutional layers. This enables us to transition smoothly from any pre-trained CNN to its functionally identical hybrid model, called Transformed CNN (T-CNN). With only 50 epochs of fine-tuning, the resulting T-CNNs demonstrate significant performance gains over the CNN as well as substantially improved robustness. We analyze the representations learnt by the T-CNN, providing deeper insights into the fruitful interplay between convolutions and self-attention.

8.1 Introduction

As we have seen in the previous chapter, Vision Transformers have emerged as powerful alternatives to CNNs. Yet, capturing long-range dependencies necessarily comes at the cost of quadratic complexity in input size, a computational burden which many recent directions have tried to alleviate [260–263]. Additionally, ViTs are generally harder to train [264, 265], and require vast amounts of pre-training [14] or distillation from a convolutional teacher [190, 266, 267] to match the performance of CNNs.

One common answer to both issues are hybrid models, which append SA layers onto convolutional backbones [227, 228, 267–269], and have already fueled successful results in a variety of tasks [233–237]. However, modelling long-range dependencies at low computational cost remains a challenge for practitioners.

8.1.1 Contributions

At a time when pre-training on vast datasets has become common practice, we ask the following question: does one need to train the SA layers during the whole learning process? Could one instead learn cheap components such as convolutions first, leaving the SA layers to be learnt at the end? In this chapter, we leverage the *Gated Positional Self-Attention* (GPSA) layers introduced in the previous chapter to fully

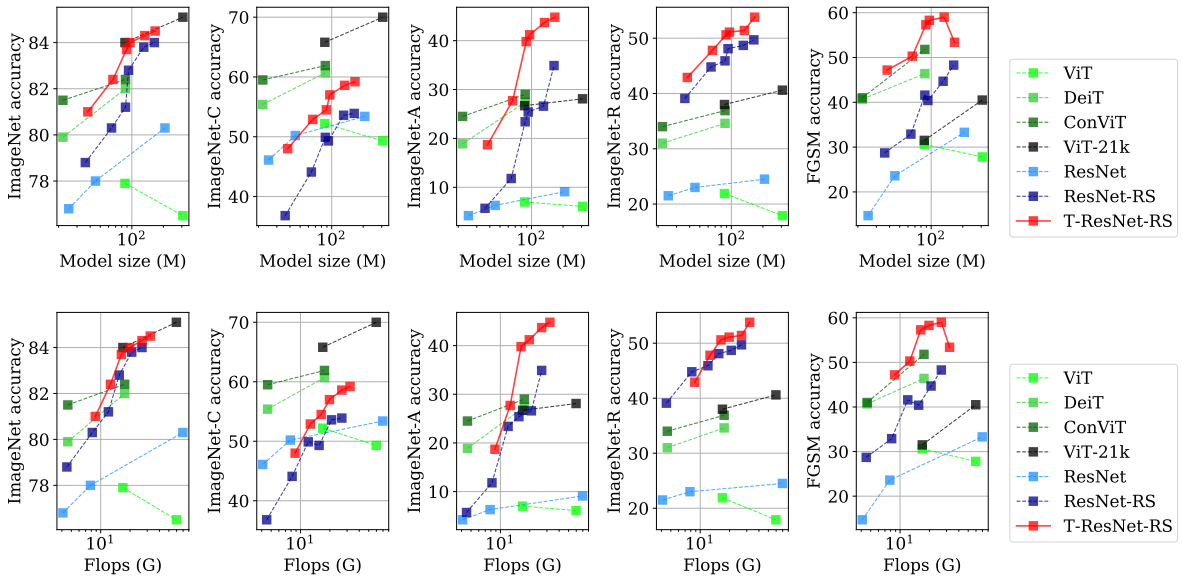


Figure 8.1: **Transformed ResNets strike a strong accuracy-robustness balance.** Our models (red) significantly outperform the original ResNet-RS models (dark blue) they were initialized from when evaluated on ImageNet-1k. They also exhibit solid performances on various robustness benchmarks (ImageNet-C, A and R, and FGSM adversarial attacks from left to right).

reparameterize any pre-trained convolutional network as a hybrid model (and not a Vision Transformer, as for the ConViT!). The latter is initialized to reproduce the mapping of the convolutional layer, but is then encouraged to learn more general mappings which are not accessible to the CNN by adjusting positional gating parameters.

After only 50 epochs of fine-tuning, the resulting Transformed CNNs (T-CNNs) boast significant performance and robustness improvements as shown in Fig. 8.1, demonstrating the practical relevance of our method. We analyze the inner workings of the T-CNNs, and show that they learn more robust representations by combining convolutional heads and attentional heads in a complementary way.

8.1.2 Related work

This chapter mainly builds on two pillars. First, the idea that SA layers can express any convolution, introduced by [243] and leveraged in the previous chapter to imbue self-attetion layers with a local inductive bias favorable to sample efficiency. Our approach leverages the opposite idea: giving an end-to-end CNN the freedom to escape locality by learning self-attention at late times.

Second, we exploit the following learning paradigm: train a simple and fast model, then reparameterize it as a more complex model for the final stages of learning. This approach was used in Chap. 6, but its practical relevance was limited by the vast increase in number of parameters required by the FCN to functionally represent the CNN. In contrast, our reparameterization hardly increases the parameter count of the CNN, making it easily applicable to any state-of-the-art CNN. Note that this reparameterization methods can be viewed an informed version of dynamic architecture growing algorithms such as AutoGrow [270].

In the context of hybrid models, various works have studied the performance gains obtained by introducing MHSA layers in ResNets with minimal architectural changes [267–269]. However, the MHSA layers used in these works are initialized randomly and need to be trained from scratch. Our approach is different, as it makes use of GPSA layers, which can be initialized to represent the same function as the convolutional layer it replaces. We emphasize that the novelty in this work is not in the architectures used, but in the unusual way they are blended together.

8.2 Methods

In this section, we introduce our method for mapping a convolutional layer to a functionally equivalent PSA layer with minimal increase in parameter count. To do this, we leverage the GPSA layers introduced in [271].

Loading the filters We want each head h of the PSA layer introduced Eq. 7.4 of the previous chapter to functionally mimic the pixel h of a convolutional filter $\boldsymbol{W}_{\text{filter}} \in \mathbb{R}^{N_h \times D_{in} \times D_{out}}$, where we typically have $D_{out} \geq D_{in}$. Rewriting the action of the MHSA operator (Eq. 7.2) in a more explicit form, we have

$$\text{MHSA}(\boldsymbol{X}) = \sum_{h=1}^{N_h} \boldsymbol{A}^h \boldsymbol{X} \underbrace{\boldsymbol{W}_{\text{val}}^h \boldsymbol{W}_{\text{out}}^h}_{\boldsymbol{W}^h \in \mathbb{R}^{D_{emb} \times D_h}} \tag{8.1}$$

In the convolutional configuration of Eq. 7.5, $\boldsymbol{A}^h \boldsymbol{X}$ selects pixel h of $\boldsymbol{X}$. Hence, we need to set $\boldsymbol{W}^h = \boldsymbol{W}_{\text{filter}}^h$. We do this by setting $D_{emb} = D_{out}$, $D_h = D_{in}$, and using set the following initialization:

$$\boldsymbol{W}_{\text{val}}^h = \boldsymbol{I}, \qquad \boldsymbol{W}_{\text{out}}^h = \boldsymbol{W}_{\text{filter}}^h. \tag{8.2}$$

Note that this differs from the usual choice made in SA layers, where $D_h = \lfloor D_{emb}/N_h \rfloor$. However, to keep the parameter count the same, we share the same $\boldsymbol{W}_{val}^h$ across different heads h, since it plays a symmetric role at initialization.

Note that this reparameterization introduces three additional matrices compared to the convolutional filter: $\boldsymbol{W}_{qry}, \boldsymbol{W}_{key}, \boldsymbol{W}_{val}$, each containing $D_{in} \times D_{in}$ parameters. However, since the convolutional filter contains $N_h \times D_{in} \times D_{out}$ parameters, where we typically have $N_h = 9$ and $D_{out} \in \{D_{in}, 2D_{in}\}$, these additional matrices are much smaller than the filters and hardly increase the parameter count. This can be seen from the model sizes in Tab. G.1.

How convolutional should the initialization be? The convolutional initialization of GPSA layers involves two parameters, determining how strictly convolutional the behavior is: the initial value of the *locality strength* α, which determines how focused each attention head is on its dedicated pixel, and the initial value of the *gating parameters* λ, which determines the importance of the positional information versus content. If $\lambda_h \gg 0$ and $\alpha \gg 1$, the T-CNN will perfectly reproduce the input-output function of the CNN, but may want to greedily stay in the convolutional configuration. Conversely, if $\lambda_h \ll 0$ and $\alpha \ll 1$, the T-CNN will forget about the input-output function of the CNN. Hence, we choose $\alpha = 1$ and $\lambda = 1$ to lie in between these two extremes, encouraging the T-CNNs to escape locality throughout training.

Architectural details To make our setup as canonical as possible, we focus on ResNet architectures [77], which contain 5 stages, with spatial resolution halfed and number of channels doubled at each stage. Our method involves reparameterizing 3×3 convolutions as GPSA layers with 9 attention heads. However, global SA is too costly in the first layers, where the spatial resolution is large. We therefore only reparameterize the last stage of the architecture, while replacing the first stride-2 convolution by a stride-1 convolution, exactly as in [269]. We also add explicit padding layers to account for the padding of the original convolutions.

8.3 Performance of the Transformed CNNs

In this section, we apply our reparametrization to state-of-the-art CNNs, then fine-tune the resulting T-CNNs to learn better representations. This method allows to fully disentangle the training of the SA layers from that of the convolutional backbone, which is of practical interest for two reasons. First, it minimizes the time spent training the SA layers, which typically have a slower throughput. Second, it separates the algorithmic choices of the CNN backbone from those of the SA layers, which are typically different; for example, CNNs are typically trained with SGD whereas SA layers perform much better with adaptive optimizers such as Adam [264], an incompatibility which may limit the performance of usual hybrid models.

Note that our reparametrization can also be applied halfway through training: this scenario is investigated in Sec. 8.4.2. Results suggest that reparametrizing at intermediate times is optimal in terms of speed-performance trade-offs. However, for easier reproducibility, we focus on reparametrizing the fully pre-trained CNNs available in the Timm package [253]: this avoids having to retrain the various models from scratch.

8.3.1 Training details

We applied our method to pre-trained ResNet-RS [272] models, using the weights provided by the Timm package [253]. These models are derived from the original ResNet [77], but use improved architectural features and training strategies, enabling them to reach better speed-accuracy trade-offs than EfficientNets.

To minimize computational cost, we restrict our fine-tuning to 50 epochs[1]. Following [264], we use the AdamW optimizer, with a batch size of 1024[2]. The learning rate is warmed up to 10^{-4} then annealed using a cosine schedule. To encourage the T-CNN to escape the convolutional configuration and learn content-based attention, we use a larger learning rate of 0.1 for the gating parameters of Eq. 7.7 (one could equivalently decrease the temperature of the sigmoid function).

We use the same data augmentation scheme as the DeiT [217], as well as rather large stochastic depth coefficients d_r reported in Tab. 8.1. Hoping that our method could be used as an alternative to the commonly used practice of fine-tuning models at higher resolution, we also increase the resolution during fine-tuning [273]. In this setting, a ResNet50 requires only 6 hours of fine-tuning on 16 V100 GPUs, compared to 33 hours for the original training. For our largest model (ResNet350-RS), the fine-tuning lasts 50 hours.

[1] We study how performance depends on the number of fine-tuning epochs in App. G.4.

[2] Confirming the results of [264], we obtained worse results with SGD.

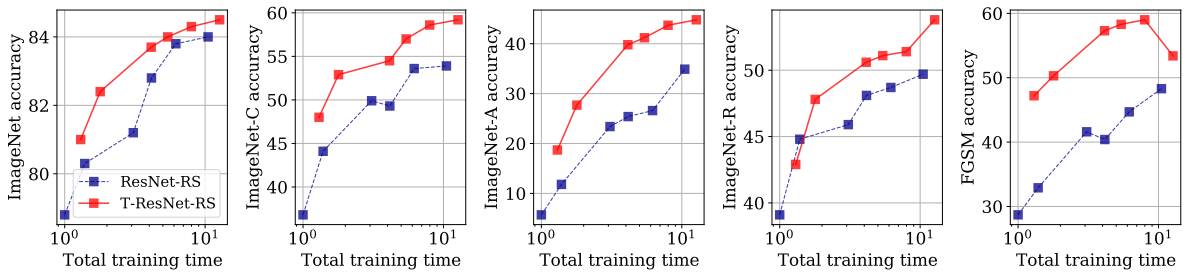


Figure 8.2: **T-CNNs reach much higher performance and robustness at equal training time.** Total training time (original training + finetuning) is normalized by the total training time of the ResNet50-RS.

Backbone	Training				Fine-tuning					
							Without SA		With SA	
	Res.	d_r	TTT	Top-1	Res.	d_r	TTT	Top-1	TTT	Top-1
ResNet50-RS	160	0.0	1 (ref.)	78.8	224	0.1	1.16	80.4	1.30	**81.0**
ResNet101-RS	192	0.0	1.39	80.3	224	0.1	1.65	81.9	1.79	**82.4**
ResNet152-RS	256	0.0	3.08	81.2	320	0.2	3.75	83.4	4.13	**83.7**
ResNet200-RS	256	0.1	4.15	82.8	320	0.2	5.04	83.7	5.42	**84.0**
ResNet270-RS	256	0.1	6.19	83.8	320	0.2	7.49	83.9	7.98	**84.3**
ResNet350-RS	288	0.1	10.49	84.0	320	0.2	12.17	84.1	12.69	**84.5**

Table 8.1: **Statistics of the models considered, trained from scratch on ImageNet.** Top-1 accuracy is measured on ImageNet-1k validation set. "TTT" stands for total training time (including fine-tuning), normalized by the total training time of the ResNet50-RS. d_r is the stochastic depth coefficient used for the various models.

8.3.2 Performance improvements

Results are presented in Tab. 8.1, where we also report the baseline improvement of fine-tuning in the same setting but without SA. In all cases, our fine-tuning improves top-1 accuracy, with a significant gap over the baseline. To demonstrate the wide applicability of our method, we report similar improvements for ResNet-D architectures in App. G.5.

Despite the extra fine-tuning epochs and their slower throughput, the resulting T-CNNs also outperforming the original CNNs on the ImageNet validation set at equal training budget, as shown in the leftmost panel of Fig. 8.2[3]. However, the major benefit of the reparametrization is in terms of robustness, as shown in Fig. 8.2(b) and explained below.

8.3.3 Robustness improvements

Recent work has shown that Transformer-based architectures offer better robustness and out-of-domain generalization than convolutional architectures [274–277]. To investigate whether our fine-tuning

[3]We estimated the training times of the original ResNet-RS models based on their throughput, for the same hardware as used for the T-ResNet-RS.

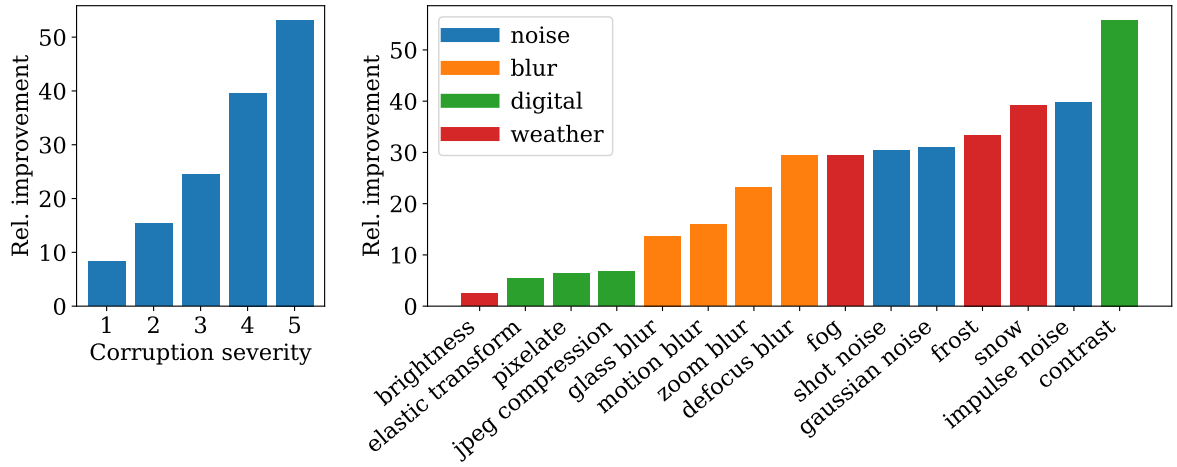


Figure 8.3: **Robustness is most improved for strong and blurry corruption categories.** We report the relative improvement between the top-1 accuracy of the T-ResNet50-RS and that of the ResNet50-RS on ImageNet-C, averaging over the different corruption categories (left) and corruption severities (right).

procedure is enough to imbue CNNs with these advantages, we evaluate our T-CNNs on various benchmarks:

- **Common corruptions**: we use ImageNet-C [278], a dataset containing 15 sets of randomly generated corruptions, grouped into 4 categories: 'noise', 'blur', 'weather', and 'digital'. Each corruption type has five levels of severity, resulting in 75 distinct corruptions. Note that to avoid distorting the corruptions, which are often pixel-based, we keep a resolution of 224 at inference, which disadvantages the large models trained at higher resolutions.
- **Adversarial robustness**: following [275], we evaluate the accuracy of our models under two white-bow attacks[4]: (i) single-step FGSM [279] and (ii) multi-step L_∞-PGD [280] with $t = 5$ steps of size $\alpha = 0.5$. Both attackers perturb the input image with max magnitude $\epsilon = 1$. We also evaluate our models on ImageNet-A [281], a dataset containing naturally "adversarial" examples from ImageNet. Note however that since this dataset is built from the flaws of a ResNet, it is potential unfair to CNNs.
- **Distribution shifts**: we use ImageNet-R [282], a dataset with various stylized "renditions" of ImageNet images ranging from paintings to embroidery, which strongly modify the local image statistics.

The full table of results is presented in Tab. G.1 of App. G.1, and illustrated in Figs. 8.1 and 8.2. The T-ResNet-RS substantially outperforms the ResNet-RS on all robustness benchmarks. For example, our T-ResNet101-RS, which is 50% faster than ResNet200-RS, reaches similar or better results all robustness tasks, despite its lower top-1 accuracy on ImageNet-1k. This demonstrates that SA improves robustness more than it improves classification accuracy. The most striking improvement is in terms of adversarial robustness, where the smallest T-ResNet-RS is on par with the largest ResNet-RS despite requiring 5 times less compute.

To better understand where the benefits come from, we decompose the improvement of the T-ResNet50-RS over the various corruption severeties and categories of ImageNet-C in Fig. 8.3. We observe

[4]We use the toolkit provided by https://github.com/bethgelab/foolbox.

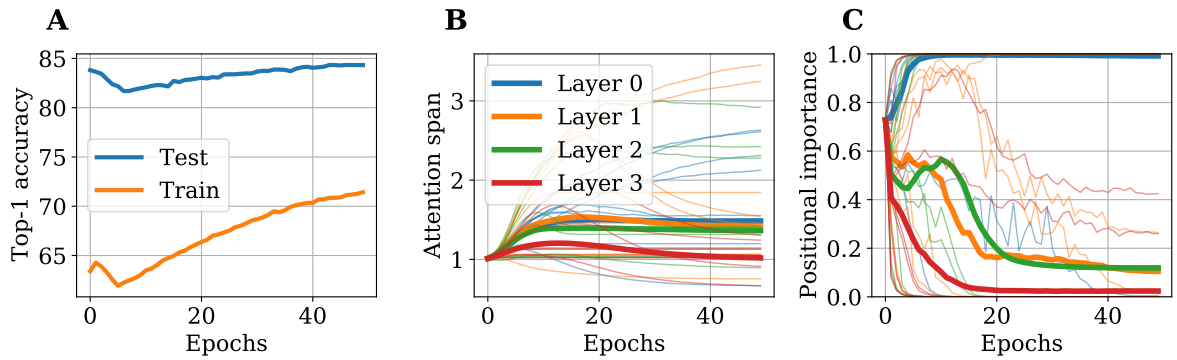


Figure 8.4: **The later layers effectively escape the convolutional configuration. A:** top-1 accuracy throughout the 50 epochs of fine-tuning of a T-ResNet270-RS. **B:** size of the receptive field of the various heads h (thin lines), calculated as α_h^{-1} (see Eq. 7.4). Thick lines represent the average over the heads. **C:** depicts how much attention the various heads h (thin lines) pay to positional information, through the value of $\sigma(\lambda_h)$ (see Eq. 7.7). Thick lines represent the average over the heads.

that improvement increases almost linearly with corruption severity. Although performance is higher in all corruption categories, there is a strong variability: the T-CNN shines particularly in tasks where the objects in the image are less sharp due to lack of contrast, bad weather or blurriness. We attribute this to the ability of SA to distinguish shapes in the image, as investigated in Sec 8.4.

8.4 Dissecting the Transformed CNNs

In this section, we provide some theoretical analysis of the inner workings of the Transformed CNNs.

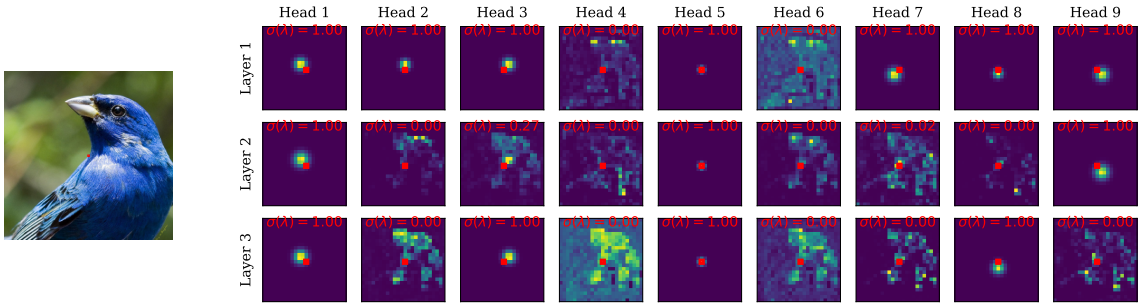
8.4.1 Representations learnt

We begin by analyzing various observables to understand how the representations of a T-ResNet270-RS evolve from those of the ResNet270-RS throughout training.

Unlearn to better relearn In Fig. 8.4A, we display the train and test accuracy throughout training[5]. The dynamics decompose into two distinct phases: accuracy dips down during the learning rate warmup phase (first 5 epochs of training), then increases back up as the learning rate is decayed.

As shown in App. G.2, the depth of the dip depends on the learning rate. For too small learning rates, the dip is small, but the test accuracy increases too slowly after the dip; for too large learning rates, the test accuracy increases rapidly after the dip, but the dip is too deep to be compensated for. This suggests that the T-CNN needs to "unlearn" to some extent, a phenomenon reminiscent of the "catapult" mechanism of [283] which propels models out of sharp minima to land in wider minima.

[5]The train accuracy is lower than the test accuracy due to the heavy data augmentation used during fine-tuning.



(a) Input image (b) Attention maps

Figure 8.5: **GPSA layers combine local and global attention in a complementary way.** We depicted the attention maps of the four GPSA layers of the T-ResNet270-RS, obtained by feeding the image on the left through the convolutional backbone, then selecting a query pixel in the center of the image (red box). For each head h, we indicate the value of the gating parameter $\sigma(\lambda_h)$ in red (see Eq. 7.7). In each layer, at least one of the heads learns to perform content-based attention ($\sigma(\lambda_h) = 0$).

Escaping the convolutional representation In Fig. 8.4B, we show the evolution of the "attention span" $1/\alpha_h$ (see Eq. 7.5), which reflects the size of the receptive field of attention head h. On average (thick lines), this quantity increases in the first three layers, showing that the attention span widens, but variability exists among different attention heads (thin lines): some broaden their receptive field, whereas others contract it.

In Fig. 8.4C, we show the evolution of the gating parameters λ^h of Eq. 7.7, which reflect how much attention head h pays to position versus content. Interestingly, the first layer stays strongly convolutional on average, as $\mathbb{E}_h\sigma(\lambda_h)$ rapidly becomes close to one (thick blue line). The other layers strongly escape locality, with most attention heads focusing on content information at the end of fine-tuning.

In Fig. 8.5, we display the attention maps after fine-tuning. A clear divide appears between the "convolutional" attention heads, which remain close to their initialization, and the "content-based" attention heads, which learn more complex dependencies. Notice that the attention head initially focusing on the query pixel (head 5) stays convolutional in all layers. Throughout the layers, the edges of the central object is more and more clearly visible, as observed in [284]. This supports the hypothesis that robustness gains obtained for blurry corruptions (see Fig. 8.3) are partly due to the ability of the SA layers to isolate objects from the background.

8.4.2 When should one start learning the self-attention layers?

We have demonstrated the benefits of initializing T-CNNs from pre-trained CNNs, a very compelling procedure given the wide availability of pretrained models. But one may ask: how does this compare to training a hybrid model from scratch? More generally, given a computational budget, how long should the SA layers be trained compared to the convolutional backbone?

Transformed CNN versus hybrid models To answer the first question, we consider a ResNet-50 trained on ImageNet for 400 epochs. We use SGD with momentum 0.9 and a batch size of 1024, warming up the learning rate for 5 epochs before a cosine decay. To achieve a strong baseline, we use the same augmentation scheme as in [217] for the DeiT. Results are reported in Tab. 8.2. In this modern training

Name	t_1	t_2	Train time	Top-1
Vanilla CNN	400	0	2.0k mn	79.04
Vanilla CNN↑320	450	0	2.4k mn	**79.78**
T-CNN	400	50	2.3k mn	79.88
T-CNN↑320	400	50	2.7k mn	**80.84**
Vanilla hybrid	0	400	2.8k mn	79.95
T-CNN*	100	300	2.6k mn	**80.44**
T-CNN*	200	200	2.4k mn	80.28
T-CNN*	300	100	2.2k mn	79.28

Table 8.2: **The benefit of late reparametrization.** We report the top-1 accuracy of a ResNet-50 on ImageNet reparameterized at various times t_1 during training. ↑320 stands for fine-tuning at resolution 320. The models with a $\star$ keep the same optimizer after reparametrization, in contrast with the usual T-CNNs.

setting, the vanilla ResNet50 reaches a solid performance of 79.04% on ImageNet, well above the 77% usually reported in litterature.

The T-CNN obtained by fine-tuning the ResNet for 50 epochs at same resolution obtains a top-1 accuracy of 79.88%, with a 15% increase in training time, and 80.84 at resolution 320, with a 35% increase in training time. In comparison, the hybrid model trained for 400 epochs in the same setting only reaches 79.95%, in spite of a 40% increase in training time.

Hence, fine-tuning yields better results than training the hybrid model from scratch, while require less time to train.

What is the best time to reparametrize? We now study a scenario between the two extreme cases: what happens if we reparametrize halfway through training? To investigate this question in a systematic way, we train the ResNet50 for t_1 epochs, then reparametrize and resume training for another t_2 epochs, ensuring that $t_1 + t_2 = 400$ in all cases. Hence, $t_1 = 400$, amounts to the vanilla ResNet50, whereas $t_1 = 0$ corresponds to the hybrid model trained from scratch. To study how final performance depends on t_1 in a fair setting, we keep the same optimizer and learning rate after the reparametrization, in contrast with the fine-tuning procedure which uses fresh optimizer.

Results are presented in Tab. 8.2. Interestingly, the final performance evolves non-monotonically, reaching a maximum of 80.44 for $t_1 = 100$, then decreasing back down as the SA layers have less and less time to learn. This non-monotonicity is remarkably similar to that observed in [285], where reparameterizing a CNN as a FCN in the early stages of training enables the FCN to outperform the CNN. Crucially, this result suggests that reparametrizing during training not only saves time, but also helps the T-CNN find better solutions.

8.5 Conclusion

In this chapter, we showed that complex building blocks such as self-attention layers need not be trained from start. Instead, one can save in compute time while gaining in performance and robustness by

initializing them from pre-trained convolutional layers. At a time where energy savings and robustness are key stakes, we believe this finding is important.

On the practical side, our fine-tuning method offers an interesting new direction for practitioners. One limitation of our method is the prohibitive cost of reparametrizing the early stages of CNNs. This cost could however be alleviated by using linear attention methods [261], an important direction for future work. Note also that while our T-CNNs significantly improve the robustness of CNNs, they do not always reach the performance of end-to-end Transformers such as the DeiT (for example on ImageNet-C, see Fig. 8.1). Bridging this gap is an important next step for hybrid models.

Part III

Understanding Learning Dynamics

Chapter 9

Optimal Learning Rate Schedules in Non-Convex Optimization

Learning rate schedules are ubiquitously used to speed up and improve optimisation. Many different policies have been introduced on an empirical basis, and many theoretical analyses have been developed for convex settings. However, in many realistic problems the loss-landscape is high-dimensional and non convex – a case for which results are scarce. In this chapter we present a first analytical study of the role of learning rate scheduling in this setting, focusing on Langevin optimization with a learning rate decaying as $\eta(t) = t^{-\beta}$. We begin by considering models where the loss is a Gaussian random function on the D-dimensional sphere ($D \to \infty$), featuring an extensive number of critical points. We find that to speed up optimization without getting stuck in saddles, one must choose a decay rate $\beta < 1$, contrary to convex setups where $\beta = 1$ is generally optimal. We then add to the problem a signal to be recovered. In this setting, the dynamics decompose into two phases: an *exploration* phase where the dynamics navigates through rough parts of the landscape, followed by a *convergence* phase where the signal is detected and the dynamics enter a convex basin. In this case, it is optimal to keep a large learning rate during the exploration phase to escape the non-convex region as quickly as possible, then use the convex criterion $\beta = 1$ to converge rapidly to the solution. Finally, we demonstrate that our conclusions hold in a common regression task involving neural networks, and discuss their generality based on results from out of equilibrium physics.

9.1 Introduction

Learning rate schedules are used across all areas of modern machine learning, yet very little is known on which schedule is most suited for a given problem. This question has been thoroughly studied for convex problems, where the optimal learning rate schedule generally goes as $\eta(t) \sim 1/t$ [286, 287]. However, deep neural networks and other high-dimensional modern optimization problems are known to operate in highly non-convex loss landscapes [288, 289]. Developing a theory to understand the impact of scheduling in this setting remains a crucial challenge.

In this chapter we present, to the best of our knowledge, the first analytical study of this problem for gradient-based algorithms. We focus on the high-dimensional inference problem of retrieving a ground truth signal $x^\star \in \mathbb{R}^D$ from observations via a noisy channel. When the noise dominates the signal, the loss simply boils down to a Gaussian random function on the D-dimensional sphere ($D \to \infty$). This

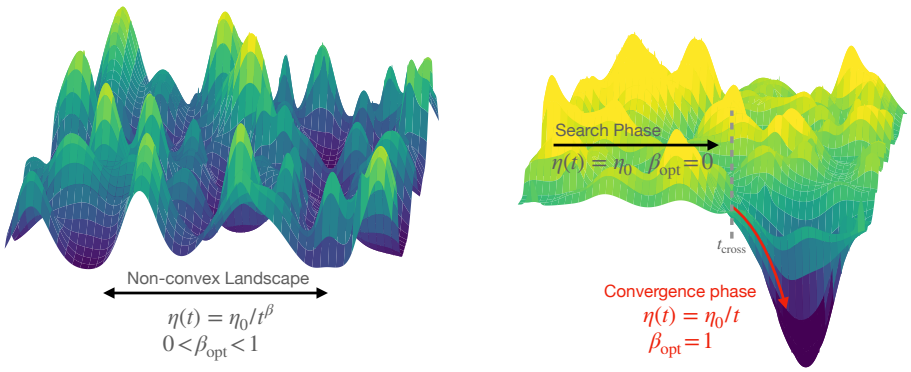


Figure 9.1: **The optimal learning rate schedule depends on the structure of the landscape.** *(Left):* in the purely non-convex landscapes of Sec. 9.3, the learning rate must be decayed as $\eta(t) = \eta_0/t^\beta$ with $\beta < 1$ to speed up optimization. *(Right):* the landscapes of Secs. 9.4 and 9.5 feature basins of attraction due to the presence of a signal to recover. One must first keep a large constant learning rate to escape the rough parts of the landscape as quickly as possible, then decay the learning rate as $\eta(t) = \eta_0/t$ once inside a convex basin.

optimization problem has been studied in the literature for constant learning rate, both using rigorous methods and techniques from statistical physics, see [112, 290–293] and references therein.

Setup Learning rate decay is generally used to reduce the noise induced by optimization schemes used in practice. For example, SGD with batch size B typically induces a noise which scales as the learning rate divided by batch size η/B [214, 294–296]. To mimick this optimization noise, we focus on Langevin dynamics [287, 297–299]. Given a loss function $\mathcal{L}$ and a temperature T, this consists in minimising $\mathcal{L}$ by updating the estimate $x \in \mathbb{R}^D$ of the signal from a random initial condition according to the equation:

$$\frac{\mathrm{d}x_i(t)}{\mathrm{d}t} = -\eta(t)\left(\frac{\partial \mathcal{L}(x, x^\star)}{\partial x_i} + \xi_i(t) + z(t)x_i(t)\right), \tag{9.1}$$

where $\xi(t) \in \mathbb{R}^D$ is a Gaussian noise with 0 mean and variance $\langle \xi_i(t)\xi_j(t')\rangle = 2T\delta_{ij}\delta(t-t')$, and the Lagrange multiplier $z(t)$ is used to enforce the spherical constraint $\|x\|^2 = D$ which we impose throughout the paper ($z(t)$ can be thought of as a weight decay that evolves during training to keep the norm of the estimator fixed). The temperature T represents the strength of the noise inherent to the optimisation algorithm, i.e. $1/B$ for SGD (we consider $T < 1$ in the following). To study scheduling, we decay the learning rate as $\eta(t) = \eta_0/t^\beta$, as commonly chosen in the literature [300, 301]. Note that here we are considering gradient-flow – our results are confirmed by experiments performed with gradient descent.

We consider two models for the loss $\mathcal{L}$: the (planted) *Sherrington-Kirkpatrick* (SK) model [302], where the signal is scrambled by a random matrix, and the more involved *spiked matrix-tensor* (SMT) model [303], where the signal is additionally observed through its contraction with a random tensor of order p. The first setup is analytically tractable both at infinite and finite dimensions [304, 305], and its landscape features a number of critical points which grows linearly with the dimension. The second setup is more involved and requires a mean-field treatment in the infinite dimensional limit [303,

306, 307]. The number of critical points grows exponentially with the dimension and has been studied analytically with the *Kac-Rice* method [308, 309]. This distinction allows us to grasp how the amount of non-convexity impacts the optimal decay of the learning rate.

9.1.1 Contributions

We begin by considering the purely non-convex setup where the signal is undetectable (left panel of Fig. 9.1). The loss is then a Gaussian random function on the D-dimensional sphere with zero mean and a covariance $\mathbb{E}[\mathcal{L}(x)\mathcal{L}(x')] \propto (x \cdot x')^p$. We determine the optimal learning rate to reach the lowest value of the loss function on an arbitrarily large (but finite) time in the high-dimensional limit. For the $p = 2$ case, corresponding to the spherical SK model, we find $\beta = 1/2$ whereas for $p > 2$ we obtain $\beta = 2/5$. The higher degree of non-convexity of the latter requires the learning rate to be decayed more slowly; we generalize these findings by leveraging results from out-of-equilibrium physics. Note that inverse square root decay is commonly used among practitioners in state-of-the-art endeavours such as training Transformers [78]; our analysis provides theoretical evidence for its soundness in a particular class of non-convex landscapes.

We then study the influence of a detectable signal (right panel of Fig. 9.1), and we determine the optimal learning rate schedule to find the signal in the shortest amount of time. In this case, a crossover time emerges between two phases [310]: a *search* phase, where the signal is weak and the dynamics travel through a rugged landscape, followed by a *convergence* phase the signal is detected and the problem becomes locally convex. We show that the optimal schedule is to keep a large constant learning rate during the first phase to speed up the search, then, once in the convex basin, to decay the learning rate as $1/t$. This protocol allows to speed up convergence and find lower loss solutions, and is reminiscent of schedules used in practice.

Finally, we show through experiments that these insights are reflected in practice when training neural networks on a teacher-student regression task with SGD.

Reproducibility The code to reproduce the figures in this chapter is available at `https://github.com/mariaref/nonconvex-lr`.

9.1.2 Related work

Empirical schedules Typically, learning rate schedules consist in a large learning rate phase followed by a decay phase. A body of works have shown that this allows to learn easy patterns early on and complex patterns later [311, 312]. Although stepwise decays of the learning rate were used for a long time [77, 313], most recent works have turned to smooth decays such as inverse square root [78] and cosine annealing [314], which involve less hyperparameters to tune. Other possibilities include cyclical learning rates [315] and automatic schedulers [316]. The use of a warmup [317] before decaying the learning rate has shown to be effective in avoiding instabilities arising from large learning rates [318, 319]. Another common practice is to use adaptive optimizers, which select a different learning rate for each learning parameter [320–322], although these have been shown to often degrade generalization [323–325].

Theortical works On the theoretical side, several works have studied Langevin dynamics for mean-field spin glasses. Particularly relevant to us are those which focus on the spherical SK setup [304, 305], as well as those showing the existence of a search and a convergence phase for the SMT model [303].

However, to the best of our knowledge, no previous works have studied these kind of highly non-convex optimization problems in the context of a non-constant learning rate. Our analysis is based on common methods in theoretical physics which have been to a large extent made rigorous in recent years [290, 291, 307, 308], and is confirmed by numerical experiments.

9.2 The speed-noise trade-off in a simple convex problem

Before studying non-convex problems, it is instructive to recall the effect learning rate decay has on optimisation in a simple 1D convex basin of curvature κ, for which $\mathcal{L}(x) = \frac{1}{2}\kappa x^2$. The Langevin equation (Eq. 9.1) can easily be solved and yields (see App. I.1):

$$\langle \mathcal{L}(t) \rangle = \underbrace{\frac{\kappa x(t_0)^2}{2} e^{-2\kappa \int_{t_0}^{t} \mathrm{d}\tau \eta(\tau)}}_{\bar{\mathcal{L}}(t)} + \underbrace{\frac{\kappa T}{2} \int_{t_0}^{t} \mathrm{d}t' \eta(t')^2 e^{-2\kappa \int_{t'}^{t} \mathrm{d}\tau \eta(\tau)}}_{\delta\mathcal{L}(t)},$$

where $\langle . \rangle$ denotes an average over the noise ξ. The first term is an *optimization* term, which amounts to forgetting the initial condition $x(t_0)$. It is present in absence of noise ($T = 0$) and its decrease is related to the way the dynamics descend in the loss landscape. The second term is a *noise* term, which is proportional to the strength of the noise T, and reflects the impact Langevin noise has on optimization.

To converge to the solution $x = 0$ as quickly as possible, one is faced with a dilemma: reducing the learning rate suppresses the effect of the noise term $\delta\mathcal{L}$, but also slows down the dynamics, leading to a larger optimization term $\bar{\mathcal{L}}$. The ideal tradeoff is found when these two effects are comparable. By taking $\eta(t) = \eta_0/t$ we obtain:

$$\bar{\mathcal{L}}(t) \propto t^{-2\eta_0\kappa}, \qquad \delta\mathcal{L}(t) \propto 1/t. \tag{9.2}$$

Hence, the loss decays to zero as $1/t$ if we take $\eta_0 \geq 1/2\kappa$, as found in many previous works [286, 287]. Note that if we take a slower decay such as $\eta(t) \sim 1/t^\beta$ with $\beta < 1$, $\bar{\mathcal{L}}(t)$ converges to 0 exponentially fast, but $\delta\mathcal{L}(t) \propto \eta(t)$ decays slower and bottlenecks the loss. Conversely, if we take a faster schedule, i.e. $\beta > 1$, then the noise term decays faster, but the dynamics stop before reaching the solution, as $\bar{\mathcal{L}}(t)$ does not converge to zero when $t \to \infty$.

This simple example illustrates the trade-off between the speed of optimisation and the noise suppressing effect, which will be the cornerstone of proper scheduling in the high-dimensional non-convex settings studied below.

9.3 Optimal decay rates in random landscapes

In this section, we consider purely non-convex optimization landscapes, where the loss $\mathcal{L}$ is a Gaussian random function defined on the D-dimensional sphere, with zero mean and covariance:

$$\mathbb{E}[\mathcal{L}(x)\mathcal{L}(x')] = \frac{D}{2}(x \cdot x')^p, \quad p \geq 2.$$

This setup, which has been studied in great detail in the context of statistical physics, can be viewed as a special case of the inference problems of Sec. 9.4 where the noise is too strong for the signal to be detectable. The aim is not to retrieve a signal, but simply to decrease the loss as quickly as possible on an arbitrarily large (but finite) time.

9.3.1 Sherrington-Kirkpatrick model

We start by focusing on the case $p = 2$. This can be achieved with the spherical version of the spin glass model introduced by [302]. Here, the variables x_i and x_j interact with each other via random symmetric couplings[1] $J_{ij} \sim \mathcal{N}(0, 1)$, and, as throughout the paper, are required to satisfy the spherical constraint $\|x(t)\|^2 = D$. The loss function is given by:

$$\mathcal{L}(x) = -\frac{1}{\sqrt{D}} \sum_{i<j}^{D} J_{ij} x_i x_j. \tag{9.3}$$

In this section, we consider the high-dimensional limit $D \to \infty$; finite-dimensional effects are discussed in Sec. 9.4.1.

Solving the dynamics To obtain the value of the loss function at all times, we multiply the original Langevin equation by x_i and sum over all components. Using Ito's lemma, and the concentration of $z(t)$ in the $D \to \infty$ limit, leads to the simple relation:

$$\begin{aligned} 0 = \left\langle \frac{\partial \|x\|^2}{\partial t} \right\rangle &= \eta(t) \left[-2\mathcal{L}(t) - Dz(t) \right] + D\eta(t)^2 T \\ \Rightarrow \mathcal{L}(t) &= -\frac{D}{2} \left(z(t) - \eta(t) T \right) \end{aligned} \tag{9.4}$$

As in the convex setup, we find a competition between an optimization term and a noise term. Since the temperature is fixed, the latter decays as $\eta(t)$. To obtain the value of the Lagrange multiplier $z(t)$, we impose the spherical constraint at all times, yielding (see App. I.2):

$$z(t) = 2 - \frac{3(1-\beta)}{4t^{1-\beta}}. \tag{9.5}$$

Hence, the scaled loss $\ell = \mathcal{L}/D$ converges to the ground state (global minimum) $\ell_{GS} = -1$ as a sum of power-laws:

$$\ell(t) - \ell_{GS} = \frac{\eta_0 T}{2t^\beta} + \begin{cases} \frac{3(1-\beta)}{8\eta_0 t^{1-\beta}}, & \beta < 1 \\ \frac{3}{8\eta_0 \log t}, & \beta = 1 \end{cases}. \tag{9.6}$$

Optimal decay rate At long times, Eq. 9.6 implies a power-law decay of the loss with an exponent $\min(\beta, 1-\beta)$ due to the speed-noise tradeoff. Hence, the optimal decay rate at long times is $\beta_{\text{opt}} = 1/2$. This is confirmed by numerical simulations at finite size, see Fig. 9.2. Note that this decay rate is empirically chosen to train many state-of-the-art neural networks such as the original Transformer [78], but, to the best of our knowledge, has never been justified from a theoretical point-of-view in a non-convex high-dimensional setting.

[1] As discussed in App. I.2, due to the universality typical of random matrix theory distributions, our results hold for a broad class of distributions for the couplings. Note also that the diagonal terms do not matter in the large D limit but for simplicity we take $J_{ii} \sim \mathcal{N}(0, 2)$.

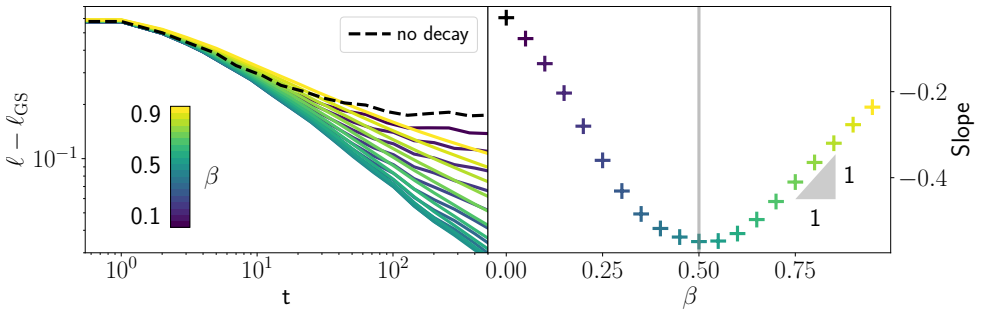


Figure 9.2: **In the SK model, the optimal decay rate is $\beta_{opt} = 0.5$.** (*Left*) Loss curves of the SK model when decaying the learning rate as $\eta(t) = \eta_0/t^\beta$ for various values of β (colored lines). (*Right*) Decay exponent of $\ell - \ell_{\rm GS}$ at long times as a function of β. We recognize a decay exponent of $\min(\beta, 1-\beta)$ as predicted by Eq. 9.6, which is fastest for $\beta_{\rm opt} = 1/2$. *Parameters:* $D = 3000$, $T = 1$, $\eta_0 = 0.1$.

Curvature analysis To gain better understanding, it is informative to study the local curvature of the effective landscape the dynamics take place in. To do so, one needs to compute the spectrum of the effective Hessian taking into account the spherical constraint of Eq. 9.1, defined as:

$$\text{Hess} = \frac{1}{\sqrt{D}} J + z(t) I. \tag{9.7}$$

In the the $D \to \infty$ limit, the spectral density of the first term, defined as $\rho(\mu) = \sum_{i=1}^{D} \delta(\mu - \mu_i)$, converges to a semi-circle law [326]:

$$\rho_{sc}(\mu) = \frac{1}{2\pi}\sqrt{4-\mu^2}, \quad \forall \mu \in [-2, 2]. \tag{9.8}$$

The spectral density of Hess is shifted to the right during the dynamics by the Lagrange multiplier $z(t)$, reflecting the way in which the local curvature changes with t. As show in Fig 9.3 and known from previous works [304], there remains negative eigenvalues at any finite time: the right edge of the spectrum only reaches 0 asymptotically as $t \to \infty$, since $z(t) \to 2$.

Hence, the dynamics never completely escape the saddles of the landscape at $D \to \infty$. This ruggedness of the landscape entails slow "glassy" dynamics, characterized by a power-law decay of the optimization term for any $\beta < 1$, contrary to the exponential decay obtained in the convex setup (Sec. 9.2).

9.3.2 The p-spin model

We now turn to the analysis of the p-spin model which has been extensively studied in physics as a model of structural glasses, see e.g. [327]. To us, it is an ideal candidate as it corresponds to a random Gaussian landscape (with $p > 2$) for which the *Kac-Rice* approach rigorously shows the existence of a number of critical points growing exponentially with the dimension [328]. It is thus intrinsically harder, i.e. more strongly non-convex than the SK model above. The loss of the p-spin model (for $p > 2$) is written as:

$$\mathcal{L} = -\sqrt{\frac{(p-1)!}{D^{p-1}}} \sum_{i_1 < \ldots < i_p} J_{i_1 \ldots i_p} x_{i_1} \ldots x_{i_p}. \tag{9.9}$$

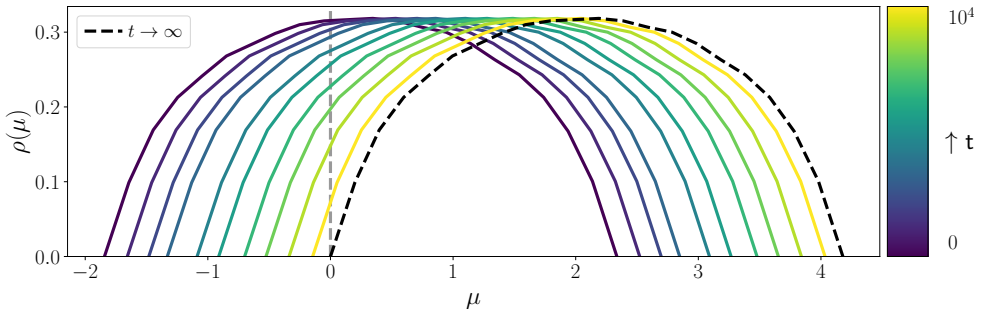


Figure 9.3: **In the SK model, the dynamics never reach a convex region.** During training, the local curvature, i.e. the spectral density of the Hessian (Eq. 9.7) shifts to the right. The left hand side of the spectrum only reaches 0 at $t \to \infty$, signalling that there remains negative eigenvalues at any finite time. *Parameters:* $D = 3000$, $T = 1$, $\eta_0 = 0.1$, $\beta = 0.8$.

Solving the dynamics In the high-dimensional limit $D \to \infty$, the Langevin dynamics of the system can be reduced to a closed set of PDEs for a set of "macroscopic" quantities, which concentrate with respect to the randomness in the couplings J and the thermal noise in the dynamics ξ, as shown rigorously in [290]. These quantities are the two-point correlation $C(t, t')$ of the system at times t, t' and the response $R(t, t')$ of the system at time t to a perturbation in the loss function at an earlier time t':

$$C(t, t') = \lim_{D \to \infty} \frac{1}{D} \mathop{\mathbb{E}}_{\xi, J} \sum_{i=1}^{D} x_i(t) x_i(t'), \tag{9.10}$$

$$R(t, t') = \lim_{D \to \infty} \frac{1}{D} \mathop{\mathbb{E}}_{\xi, J} \sum_{i=1}^{D} \frac{\delta x_i(t)}{\delta \xi_i(t')}. \tag{9.11}$$

Their dynamics is described by a closed set of integro-differential equations, dubbed the Crisanti-Horner-Sommers-Cugliandolo-Kurchan (CHSCK) equations [290, 306, 329]. We extend these equations to the non-constant learning rate case using the methods reviewed in [330]:

$$\frac{\partial R(t_1, t_2)}{\partial t_1} = F_R^p(z, R, C, \eta), \tag{9.12}$$

$$\frac{\partial C(t_1, t_2)}{\partial t_1} = F_C^p(z, R, C, \eta), \tag{9.13}$$

$$z(t) = T\eta(t) + p \int dt_2 \eta(t_2) R(t_2, t) C^{p-1}(t_2, t), \tag{9.14}$$

where we deferred the full expression of the update functions F_R^p and F_C^p as well as their derivation to App. I.4.1.

Imposing the the spherical constraint $C(t, t) = 1$ allows to find the value of the spherical constraint $z(t)$. To compute the loss, we follow the same procedure as in the SK model and obtain:

$$\ell(t) \equiv \frac{\mathcal{L}}{D} = -\frac{1}{p} \left(z(t) - T\eta(t) \right). \tag{9.15}$$

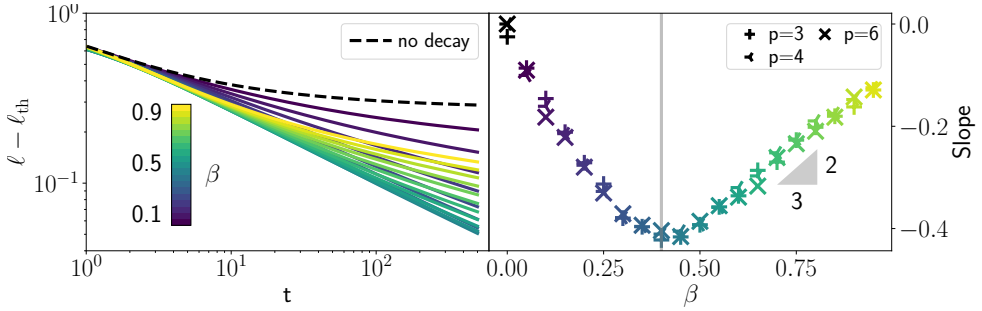


Figure 9.4: **In the p-spin model, the optimal decay rate is $\beta_{opt} = 0.4$.** *(Left)* Loss curves of the 3-spin model at $T = 1$ when decaying the learning rate as $\eta(t) = \eta_0/t^\beta$ for various values of β (colored lines). *(Right)* Decay exponent of $\ell - \ell_{th}$ at long times, for various p, as a function of β. We recognize a decay exponent of $\min(\beta, \gamma(1-\beta))$, as predicted by Eq. 9.17, which is fastest for $\beta_{\rm opt} = 2/5$. *Parameters:* $dt = 10^{-2}$, $\eta_0 = 0.5$, $T = 1$.

Optimal decay rate Here again we find that two competing terms contribute to the loss, the first related to optimisation and the second to noise. By choosing a learning rate $\eta(t) = \eta_0/t^\beta$, the later decays as $t^{-\beta}$. The decay of the former is more complex due to the high complexity of the landscape. It can be shown [306] that the system never reaches the ground state, instead remaining trapped in so-called threshold states where the Hessian has many zero eigenvalues (the density of eigenvalues is a Wigner semicircle whose left edge is zero as in the SK model). The loss is then given by:

$$\ell_{th} = -\frac{\sqrt{4(p-1)}}{p} > \ell_{GS}. \tag{9.16}$$

The relaxation towards the threshold states is characterised by a power-law due to the rough energy landscape, but with a different exponent this time: $z_{th} - z(t) \propto t^{-\gamma}$, with $\gamma = 2/3$ at $T = 0$ [331]. Using the CHSCK equations (9.12), we analytically show in App. I.3 that with decaying learning rate the exponent becomes $\gamma(1-\beta)$. Hence, similarly to the SK model, the decay of the loss is controlled by a competition between two power-laws:

$$\ell(t) - \ell_{\rm th} \sim t^{-\min(\beta, \gamma(1-\beta))} \Rightarrow \beta_{\rm opt} = \frac{\gamma}{1+\gamma} = \frac{2}{5}. \tag{9.17}$$

In Fig. 9.4, we numerically integrate Eqs. 9.12 for $p = 3, 4, 6$, confirming that the optimal decay rate to balance the noise and the optimization terms is $\beta_{opt} = 2/5$. The numerical integration is non-trivial and we implement it using the tools developed in [303].

9.3.3 Relation with annealing in physics

The results found in this section can be put in a very general framework that was developed in physics of out of equilibrium systems. As shown in App. I.3.1, using a learning rate schedule is equivalent to annealing the temperature of the physical system as a power-law $t^{-\beta/1-\beta}$. Thus, finding the optimal learning rate schedule to minimize the loss is equivalent to determining the optimal annealing protocol to decrease the energy. A key ingredient in the solution is how fast the dynamics descend in the loss landscape in absence of noise. In physical systems, this optimization term generally follows a power-law decay with exponent γ [332–334].

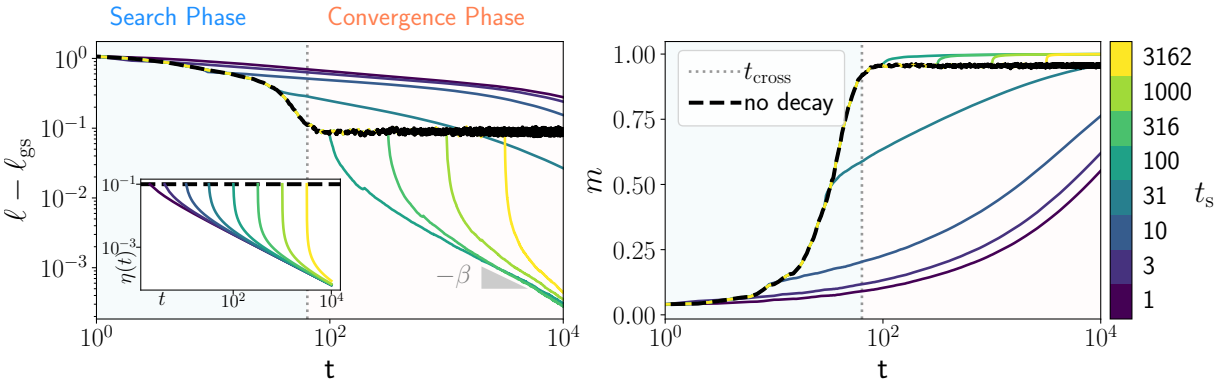


Figure 9.5: **Emergence of a crossover time in the planted SK model.** The dashed black show the loss **(left)** and overlap with the signal **(right)** at constant learning rate $\eta_0 = 0.1$. The colored lines, show the result of keeping the learning rate constant until t_s and then decaying as $\eta(t) = \eta_0/(t - t_s)^\beta$ (as shown in the inset). The dashed vertical line marks the theoretical crossover time $t_{\text{cross}} = \frac{\log D}{2\eta_0 \kappa}$, it matches with the time at which the loss, at constant learning rate, saturates. Before the crossover, decaying the learning rate is detrimental. After the crossover, it allows the model to converge to zero loss as $t^{-\beta}$ and to perfectly recover the signal. We set $D = 3000, T = 1., \eta_0 = 0.1, \beta = 0.8, \kappa = 0.5$.

At finite temperature, the speed-noise tradeoff requires this decay rate to be equal to that of the temperature, $\beta/1-\beta$, leading to $\beta_{\text{opt}} = \gamma/1+\gamma$. The exponent γ has been determined in many statistical physics problems, corresponding to different high-dimensional non-convex landscapes, and typically ranges from zero (logarithmic relaxation) to one. Our results extend to all these problems and, and predict optimal annealing exponents varying between 0 and $1/2$.

9.4 Recovering a signal: the two phases of learning

We now move to the setup where there is a signal $x^\star$ in the problem, which the algorithm aims to retrieve in the shortest time possible. In addition to the random Gaussian function, the loss now contains a deterministic term forming an attraction basin in the landscape, as sketched in the right panel of Fig. 9.1.

9.4.1 Spiked Sherrington-Kirkpatrick model

We first consider the so-called planted SK model, where the objective is to retrieve a ground truth $x^\star$ such that $\|x^\star\|^2 = D$, i.e. maximize the overlap with the signal $m = \sum_i x_i \cdot x_i^\star / D$. We enforce as before the spherical constraint $\|x\|^2 = D$ which induces $m \in [-1, 1]$, and sample randomly the initial configuration of x, such that the initial overlap is of order $1/\sqrt{D}$. The loss function takes the form:

$$\mathcal{L}(x) = -\frac{D}{2}m^2 - \frac{\Delta}{\sqrt{D}} \sum_{i<j}^{D} J_{ij} x_i x_j = \frac{1}{2} x H x^\top, \tag{9.18}$$

with $H = -\frac{\Delta}{\sqrt{D}} J - \frac{1}{D} x^\star x^{\star\top}$.

Decreasing Δ makes the signal easier to detect, leading to an easier problem. For $\Delta < 1/2$, an eigenvalue of H pops out of the semicircle law (9.8) as a BBP transition takes place [335], leading to the

follow spectrum:

$$\rho(\mu) = \left(1 - \frac{1}{D}\right) \rho_{sc}(\mu/\Delta) + \frac{1}{D}\rho(\mu - 1) \tag{9.19}$$

This is the regime in which the signal overcomes the noise, i.e. the global minimum of the loss has a finite overlap with the signal, which can then be retrieved by gradient flow (or gradient descent).

In the following, we assume that $\Delta < 1/2$, and define the gap between the largest and second largest eigenvalue as $\kappa \equiv 1 - 2\Delta$. In App. I.2, we analytically show the emergence of a crossover time,

$$t_{\text{cross}} = \left(\frac{\log D}{2\eta_0 \kappa}\right)^{\frac{1}{1-\beta}}. \tag{9.20}$$

Before t_{cross}, the system behaves as if the signal was absent, i.e. as in Sec. 9.3.1: this is the *search* phase. After t_{cross}, the signal is detected: this is the *convergence* phase. The loss becomes:

$$\ell(t) - \ell_{GS} = \frac{\eta_0 T}{2t^\beta} + \begin{cases} \mathcal{O}(e^{-2\eta_0 \kappa t^{1-\beta}}), & \beta < 1 \\ \mathcal{O}(t^{-2\eta_0 \kappa}), & \beta = 1 \end{cases} \tag{9.21}$$

with $\ell_{GS} = -1$. We recognize here the exact same result as obtained in the convex setup of Eq. 9.2: as long as $\eta_0 > 1/2\kappa$, the optimal learning rate schedule is $\eta = \eta_0/t$. This indicates that the dynamics has entered a convex basin of curvature κ.

Optimal learning rate schedule To speed up the initial phase where the signal hasn't yet aligned with the signal, one needs to reduce t_{cross}, which is achieved by using a large learning rate η_0 without any decay ($\beta = 0$). Passed this crossover, the system enters a convex basin, and the optimal exponent becomes $\beta = 1$. Ergo, the best schedule is to keep the learning rate constant up to t_{cross}, then to decay it with $\beta = 1$, in contrast with the case without signal where $\beta = 1/2$ was optimal, see Sec. 9.3.1. This is confirmed by the numerical experiments of Fig. 9.5, where we start decaying the learning rate as $\eta_0/(t-t_s)^{-\beta}$ for different "switch" times t_s. Decaying too early, with $t_s < t_{\text{cross}}$, slows down the dynamics, whereas $t_s > t_{\text{cross}}$ enables the system to reach the ground state at a rate $t^{-\beta}$.

Finite-dimensional effects The two phases in the dynamics are a general feature when there is a finite gap κ between the largest and second largest eigenvalue of H. In the $D \to \infty$ limit, this only occurs when $\Delta < 1/2$. However, when $\Delta > 1/2$, there is a finite gap at finite D due to the discrete nature of the spectrum, which scales as $\kappa \sim D^{-2/3}$ [336]. This induces a crossover time $t_{\text{cross}} \sim D^{2/3}$. Hence, decaying the learning rate as β_{opt} remains optimal for any finite time budget $t < t_{\text{cross}}$, but for a large budget $t > t_{\text{cross}}$, using the two-step schedule described in this section becomes optimal.

9.4.2 Spiked Matrix-Tensor model

We finally move to the analysis of the SMT model for which the loss function is [307]:

$$\mathcal{L}(x) = -\frac{D}{2\Delta_2} m^2 - \sqrt{\frac{1}{\Delta_2 D}} \sum_{i<j} J_{i,j} x_i x_j - \frac{D}{p\Delta_p} m^p - \sqrt{\frac{(p-1)!}{\Delta_p D^{p-1}}} \sum_{i_1 < \ldots < i_p} J_{i_1,\ldots,i_p} x_{i_1} .. x_{i_p},$$

where both J_{ij} and $J_{i_1,..,i_p}$ sampled i.i.d. from $\mathcal{N}(0,1)$. As understood from the loss function, the signal is observed through its contraction with a matrix and a tensor of order p. This model is a natural

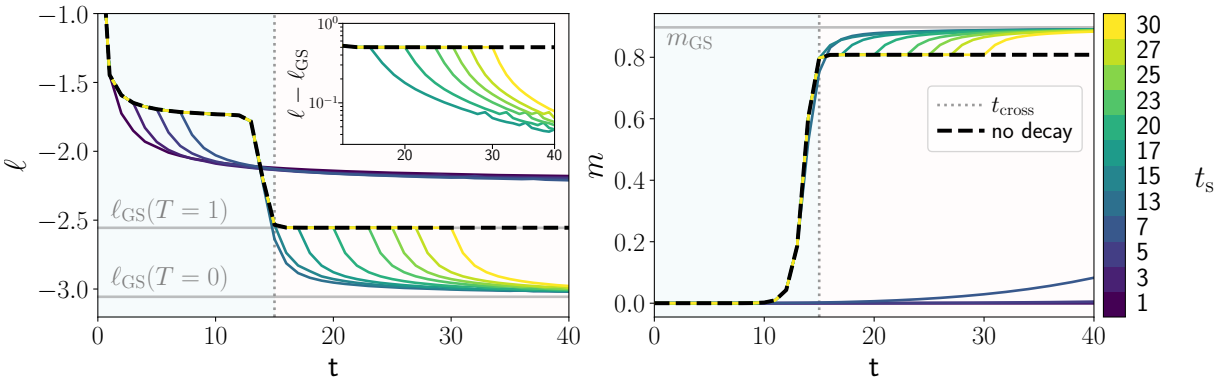


Figure 9.6: **Emergence of a crossover time in the SMT model.** By fixing β from start, or anytime before $t_{\rm cross}$, a randomly initialised system will remain stuck at *threshold* states at high loss until $t_{\rm cross}$ which is minimal for constant learning rate $\beta = 0$. In contrast, by decaying the learning rate at long times allows to reach lower loss solutions (*left*) with higher overlap with the signal (*right*). The optimal schedule is to keep η constant until $t_{\rm cross}$ and then set $\beta = 1$. By doing so, we get the best of both worlds: the first phase minimises $t_{\rm cross}$ while the second allows to reach more informative solutions. *Parameter:* $\beta = 0.8$, $\Delta_2 = 0.2$, $\Delta_p = 6$, $\eta_0 = 1$, $T = 1$, $dt = 10^{-2}$, $m_0 = 10^{-10}$.

next step for our analysis: its loss landscape is extremely non-convex, but its dynamics are exactly solvable in the $D \to \infty$ limit. They can be described by a closed set of PDEs describing the dynamical evolution of the quantities $m(t)$, $C(t,t')$, $R(t,t')$ and $z(t)$ described in Sec. 9.3.2. The derivation of these equations is deferred to the appendix I.4.1.

The difficulty of the problem is controlled by the values of Δ_2 and Δ_p. Here, we focus on the *Langevin easy* phase, defined in [307], where a randomly initialized system recovers the signal and the overlap converges to a value of order one.[2] The dynamics in this setting have been well understood at constant learning rate in [303], and are shown as a black line in Fig. 9.6 for $\eta_0 = 1$: the system remains trapped in the exponentially many *threshold states* until a time $t_{\rm cross}$. At $t_{\rm cross}$, the system finally detects the signal and the overlap jumps to a value $m_{\rm gs}$ of order one. This behavior is reminiscent of the *grokking* phenomenon observed for neural networks [338].

The colored lines of Fig. 9.6 show that decaying the learning rate from a time t_s affects optimisation in two different ways. (i) If we choose $t_s < t_{\rm cross}$, the loss actually starts by dropping, in contrast with what was observed in Fig. 9.5. However, this drop in the loss does not yield an increase of the overlap with the signal, and the system rapidly gets stuck, remaining in a state of low overlap even after $t_{\rm cross}$. (ii) If we choose $t_s > t_{\rm cross}$, once the signal is detected, the noise is suppressed, allowing the system to converge to the ground state and the overlap to increase. Hence, the optimal schedule is again to keep a constant large learning rate during the search phase (i.e. until $t_{\rm cross}$) then decay with $\beta = 1$. We provide further theoretical justification for this behavior in App. I.4.2.

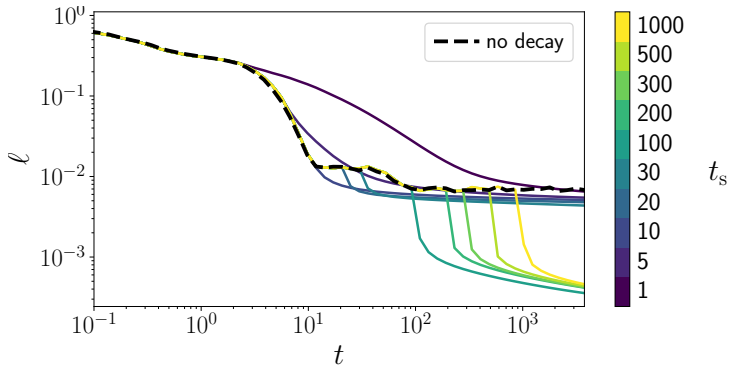


Figure 9.7: **The crossover time is also reflected in a regression task with SGD.** A K hidden nodes 2 layer neural network student is trained to reproduce the output of her M hidden nodes teacher on gaussian inputs in D dimensions. As before, we find that decaying the learning rate before the loss plateaus performance, but decaying as $\eta(t) \sim t^{-1}$ once the plateau is reached allows to reach zero loss. *Parameters:* $D = 500$, $P = 10^4$, $\eta_0 = 10^{-1}$, $M = K = 2$, $\beta = 0.8$.

9.5 Turning to SGD: teacher-student regression

Our work has demonstrated the emergence of a crossover time in a class of inference problems, before which one should keep the learning rate constant and after which it becomes useful to decay the learning rate.

We now investigate these findings in a setup that is more realistic but simple enough to be amenable to analytical treatment in the near future: the *teacher-student* regression problem described in the previous chapter. Just as before, a student network is trained to mimick the ouputs of a teacher by minimising the test error (ϵ_g) over a dataset of P input-outputs observations $\{\boldsymbol{x}_\mu, y_\mu\} \in \{\mathbb{R}^D, \mathbb{R}\}$. Here both the student S and the teacher T are two-layer networks:

$$S(x) = \sum_{k=1}^{K} W_2^k \sigma\left(\frac{W_1^k \cdot x}{\sqrt{D}}\right) \quad T(x) = \sum_{m=1}^{M} \tilde{W}_2^m \sigma\left(\frac{\tilde{W}_1^m \cdot x}{\sqrt{D}}\right).$$

We train on i.i.d. gaussian inputs $x_i \sim \mathcal{N}(0,1)$ via SGD, by minimising the error over mini-batches of size B:

$$\epsilon_g = \frac{1}{B}\sum_{\mu=1}^{B} \left(S(x_\mu) - T(x_\mu)\right)^2, \tag{9.22}$$

The optimisation noise is controlled by the batch size B and is absent for full batch SGD. To study the effect of learning rate scheduling, we focus on a mini-batch of size 1 for which optimisation noise is high.

Fig. 9.7 shows the test error (calculated over the whole training set) of a student with $K = 2$ hidden units learning from a teacher with $M = 2$ hidden units (results with different sizes are presented in App. I.5). As before, we keep the learning rate constant η_0 until a time t_s then decay it as $\eta_0/(t-t_s)^{-\beta}$. The phenomenology is remarkably similar to that of Sec. 9.4: there exists a cross-over time t_{cross} such that if the learning rate is decayed before t_{cross}, optimisation remains stuck at high ϵ_g. In contrast, decaying

[2]We must start from a very small initial overlap $m_0 = 10^{-10}$ as explained in [303], since $m_0 = 0$ would cause the system to remain stuck in the $D \to \infty$ limit considered here [337].

the learning rate after after t_{cross} enables to tame the noise associated with optimisation and converge to lower loss solutions.

9.6 Conclusion

In this chapter, we have analyzed learning rate scheduling in a variety of high-dimensional non-convex optimization problems. First, we focused on purely non-convex problems (without any basins of attraction), and showed that the optimal learning rate decay in the high-dimensional limit has an exponent smaller than one, which varies according to the degree of non-convexity of the problem at hand (ranging from 0.4 to 0.5 in the problems considered here). Then, we studied models where a signal must be recovered in presence of noise. In this case, what is important is not *how fast* we decay the learning rate, but *when* we start decaying it. It is better to keep a large learning rate in the *search* phase to find the convex basin as quickly as possible, and only then start decaying the learning rate.

These theoretical findings are remarkably reminiscent of learning rate schedules used in practice. Establishing a tighter connection is an important direction for future work: could the $1/\sqrt{t}$ decay commonly used to train transformers reflect the properties of the landscape the dynamics take place in? Conversely, could one predict the optimal decay rate by inspecting the properties of the landscape? Establishing such connections in simple settings such as that of Sec. 9.5 is certainly within reach thanks to the recent analytical tools developed in [339–343].

Chapter 10

Align, then Memorise: the Dynamics of Learning with Feedback Alignment

Direct Feedback Alignment (DFA) is emerging as an efficient and biologically plausible alternative to backpropagation for training deep neural networks. Despite relying on random feedback weights for the backward pass, DFA successfully trains state-of-the-art models such as Transformers. On the other hand, it notoriously fails to train convolutional networks. An understanding of the inner workings of DFA to explain these diverging results remains elusive. In this chapter, we propose a theory of feedback alignment algorithms. We first show that learning in shallow networks proceeds in two steps: an *alignment* phase, where the model adapts its weights to align the approximate gradient with the true gradient of the loss function, is followed by a *memorisation* phase, where the model focuses on fitting the data. This two-step process has a *degeneracy breaking* effect: out of all the low-loss solutions in the landscape, a network trained with DFA naturally converges to the solution which maximises gradient alignment. We also identify a key quantity underlying alignment in deep linear networks: the conditioning of the *alignment matrices*. The latter enables a detailed understanding of the impact of data structure on alignment, and suggests a simple explanation for the well-known failure of DFA to train convolutional neural networks. Numerical experiments on MNIST and CIFAR10 clearly demonstrate degeneracy breaking in deep non-linear networks and show that the align-then-memorize process occurs sequentially from the bottom layers of the network to the top.

10.1 Introduction

Training a deep neural network on a supervised learning task requires solving the credit assignment problem: how should weights deep in the network be changed, given only the output of the network and the target label of the input? Today, almost all networks from computer vision to natural language processing solve this problem using variants of the back-propagation algorithm (BP) popularised several decades ago by [344].

For concreteness, we illustrate BP using a fully-connected deep network of depth L with weights W_l in the lth layer. Given an input $x \equiv h_0 \in \mathbb{R}^D$, the output $\hat{y} \in \mathbb{R}^C$ (with C the output dimension) is computed sequentially as $\hat{y} = f_y(a_L)$, with $a_l = W_l h_{l-1}$ and $h_l = \sigma(a_l)$, where σ is a pointwise non-linearity. For regression, the loss function $\mathcal{L}$ is the mean-square error and f_y is the identity. Given

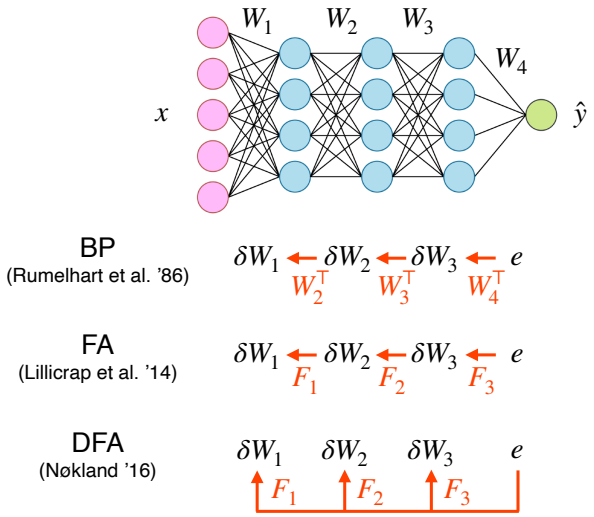


Figure 10.1: **Three approaches to the credit assignment problem in deep neural networks.** In *back-propagation* (BP), the weight updates δW_l are computed sequentially by transmitting the error e from layer to layer using the transpose of the network's weights $W_l^\top$. In *feedback alignment* (FA) [347], $W_l^\top$ are replaced by fixed random feedback matrices F_l. In *direct feedback alignment* (DFA) [348], the error is directly injected to each layer using random feedback matrices F_l, enabling parallelized training.

the error $e \equiv \partial\mathcal{L}/\partial a_L = \hat{y} - y$ of the network on an input x, the update of the last layer of weights reads

$$\delta W_L = -\eta e h_{L-1}^\top \tag{10.1}$$

for a learning rate η. The updates of the layers below are given by $\delta W_l = -\eta \delta a_l h_{l-1}^T$, with factors δa_l defined sequentially as

$$\delta a_l^{\mathrm{BP}} = \partial\mathcal{L}/\partial a_l = \left(W_{l+1}^T \delta a_{l+1}\right) \odot \sigma'(a_l)\,, \tag{10.2}$$

with $\odot$ denoting the Hadamard product. BP thus solves the credit assignment problem for deeper layers of the network by using the transpose of the network's weight matrices to transmit the error signal across the network from one layer to the next, see Fig. 10.1.

Despite its popularity and practical success, BP suffers from several limitations. First, it relies on symmetric weights for the forward and backward pass, which makes it a biologically implausible learning algorithm [345, 346]. Second, BP updates layers sequentially during the backward pass, preventing an efficient parallelisation of training, which becomes ever more important as state-of-the-art networks grow larger and deeper.

In light of these shortcomings, algorithms which only approximate the gradient of the loss are attracting increasing interest. [347] demonstrated that neural networks can be trained successfully even if the transpose of the network weights $W_l^\top$ are replaced by *random* feedback connections F_l in the backward pass, an algorithm they called "feedback alignment" (FA):

$$\delta a_l^{\mathrm{FA}} = (F_l \delta a_{l+1}) \odot \sigma'(a_l)\,. \tag{10.3}$$

In this way, they dispense with the need of biologically unrealistic symmetric forward and backward weights [345, 346]. The "direct feedback alignment" (DFA) algorithm of [348] takes this idea one step

further by propagating the error directly from the output layer to each hidden layer of the network through random feedback connections F_l:

$$\delta a_l^{\text{DFA}} = (F_l e) \odot \sigma'(a_l) \,. \tag{10.4}$$

DFA thus allows updating different layers in parallel. Fig. 10.1 shows the information flow of all three algorithms.

While it was initially unclear whether DFA could scale to challenging datasets and complex architectures [349, 350], recently [351] narrowed the gap with BP when using DFA to train a number of state-of-the-art architectures on problems ranging from neural view synthesis to natural language processing. Yet, feedback alignment notoriously fails to train convolutional networks [350, 352–354]. These varied results underline the need for a theoretical understanding of how and when feedback alignment works.

10.1.1 Contributions

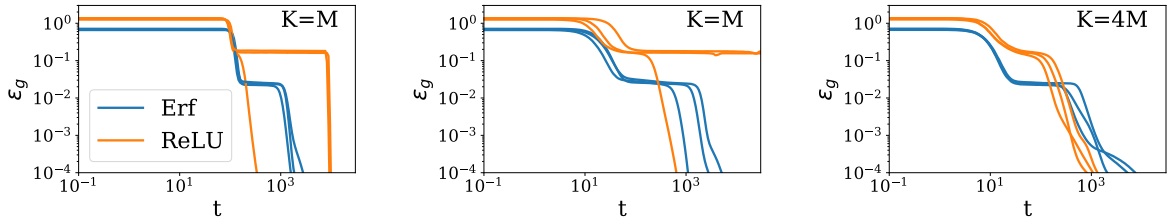
In this chapter, we make the following contributions:

1. We give an analytical description of DFA dynamics in shallow non-linear networks, building on seminal work analysing BP in the limit of infinitely many training samples [355–357].
2. We show that in this setup, DFA proceeds in two steps: an alignment phase, where the forward weights adapt to the feedback weights to improve the approximation of the gradient, is followed by a memorisation phase, where the network sacrifices some alignment to minimise the loss. Out of the same-loss-solutions in the landscape, DFA converges to the one that maximises gradient alignment, an effect we term "degeneracy breaking".
3. We then focus on the alignment phase in the setup of deep linear networks, and uncover a key quantity underlying GA: the conditioning of the alignment matrices. Our framework allows us to analyse the impact of data structure on DFA, and suggests an explanation for the failure of DFA to train convolutional layers.
4. We complement our theoretical results with experiments that demonstrate the occurence of (i) the Align-then-Memorise phases of learning, (ii) degeneracy breaking and (iii) layer-wise alignment in deep neural networks trained on standard vision datasets.

Reproducibility We host all the code to reproduce our experiments online at `https://github.com/sdascoli/dfa-dynamics`.

10.1.2 Related work

[347] gave a first theoretical characterisation of feedback alignment by arguing that for two-layer linear networks, FA works because the transpose of the second layer of weights W_2 tends to align with the random feedback matrix F_1 during training. This *weight alignment* (WA) leads the weight updates of FA to align with those of BP, leading to *gradient alignment* (GA) and thus to successful learning. [358] extended this analysis to the deep linear case for a variant of DFA called "Direct Random Target Projection" (DRTP), under the restrictive assumption of training on a single data point. [348] also introduced a layerwise alignment criterion to describe DFA in the deep nonlinear setup, under the assumption of constant update directions for each data point.



(a) BP, matched (b) DFA, matched (c) DFA, overparametrized

Figure 10.2: **Learning dynamics of back-propagation and feedback alignment for sigmoidal and ReLU neural networks learning a target function.** Each plot shows three runs from different initial conditions for every setting, where a shallow neural network with K hidden nodes tries to learn a teacher network with M hidden nodes. **(a)** All networks trained using BP in the matched case $K = M$ achieve perfect test error. **(b)** Sigmoidal networks achieve perfect test error with DFA, but the algorithm fails in some instances to train ReLU networks ($K = M$). **(c)** In the over-parametrised case ($K > M$), both sigmoidal and ReLU networks achieve perfect generalisation when trained with DFA. *Parameters:* $D = 500, L = 2, M = 2, \eta = 0.1, \sigma_0 = 10^{-2}$.

10.2 A two-phase learning process

We begin with an exact description of DFA dynamics in shallow non-linear networks. Here we consider a high-dimensional scalar regression task where the inputs $x \in \mathbb{R}^D$ are sampled i.i.d. from the standard normal distribution. We focus on the classic *teacher-student* setup, where the labels $y \in \mathbb{R}$ are given by the outputs of a "teacher" network with random weights [76, 359–362]. In this section, we let the input dimension $D \to \infty$, while both teacher and student are two-layer networks with $K, M \sim O(1)$ hidden nodes.

We consider sigmoidal, $\sigma(x) = \text{erf}\left(x/\sqrt{2}\right)$, and ReLU activation functions, $\sigma(x) = \max(0, x)$. We asses the student's performance on the task through its the *generalisation error*, or test error:

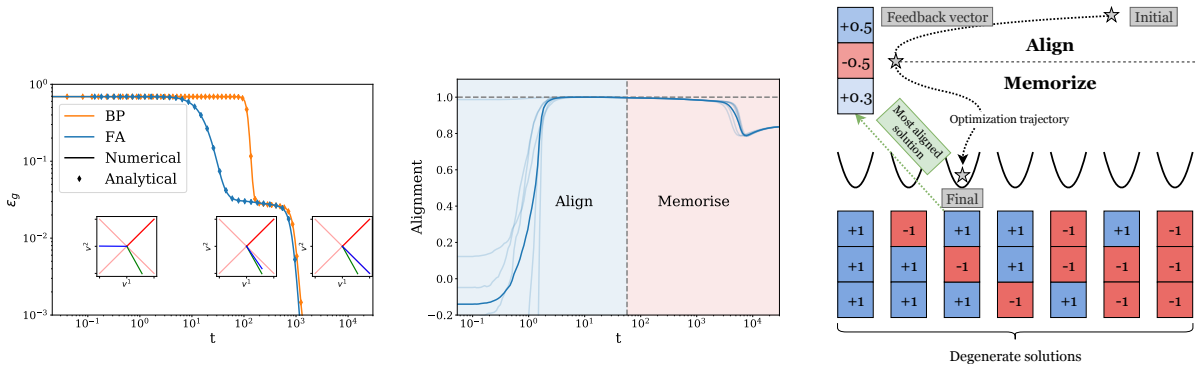
$$\epsilon_g(\theta, \tilde{\theta}) \equiv \frac{1}{2}\mathbb{E}\left[\hat{y} - y\right]^2 \equiv \frac{1}{2}\mathbb{E}\left[e^2\right], \tag{10.5}$$

where the expectation $\mathbb{E}$ is taken over the inputs for a given teacher and student networks with parameters $\tilde{\theta} = (M, \tilde{W}_1, \tilde{W}_2, \sigma)$ and $\theta = (K, W_1, W_2, \sigma)$. Learning a target function such as the teacher is a widely studied setup in the theory of neural networks [103, 363–374].

In this shallow setup, FA and DFA are equivalent, and only involve one feedback matrix, $F_1 \in \mathbb{R}^K$ which back-propagates the error signal e to the first layer weights W_1. The updates of the second layer of weights W_2 are the same as for BP.

Performance of BP vs. DFA We show the evolution of the test error (10.5) of sigmoidal and ReLU students trained via vanilla BP in the "matched" case $K = M$ in Fig. 10.2 a, for three random choices of the initial weights with standard deviation $\sigma_0 = 10^{-2}$. In all cases, learning proceeds in three phases: an initial exponential decay; a phase where the error stays constant, the "plateau" [76, 355, 372]; and finally another exponential decay towards zero test error.

Sigmoidal students trained by DFA always achieve perfect generalisation when started from different initial weights with a different feedback vector each time (blue in Fig. 10.2 b) raising a first question: if



(a) Generalization dynamics (b) Alignment dynamics (c) Degeneracy breaking

Figure 10.3: **(a) Theory gives exact prediction for the learning dynamics.** We plot learning curves for BP and DFA obtained from (i) a single simulation (solid lines), (ii) integration of the ODEs for BP dynamics [355, 357] (orange dots), (iii) integration of the ODEs for DFA derived here (blue dots). *Insets*: Teacher second-layer weights (red) as well as the degenerate solutions (light red) together with the feedback vector F_1 (green) and the student second-layer weights v (blue) at three different times during training with DFA. *Parameters:* $D = 500, K = M = 2, \eta = 0.1, \sigma_0 = 10^{-2}$.
(b) Align-then-Memorise process. Alignment (cosine similarity) between the student's second layer weights and the feedback vector. In the align phase, the alignment increases, and reaches its maximal value when the test error reaches the plateau. Then it decreases in the memorization phase, as the student recovers the teacher weights.
(c) The degeneracy breaking mechanism. There are multiple degenerate global minima in the optimisation landscape: they are related through a discrete symmetry transformation of the weights that leaves the student's output unchanged. DFA chooses the solution which maximises the alignment with the feedback vector.

the student has to align its second-layer weights with the random feedback vector in order to retrieve the BP gradient [347], i.e. $W_2 \propto F_1$, how can it recover the teacher weights perfectly, i.e. $W_2 = \tilde{W}_2$?

For ReLU networks, over-parametrisation is key to the consistent success of DFA: while some students with $K = M$ fail to reach zero test error (orange in Fig. 10.2 b), almost every ReLU student having more parameters than her teacher learns perfectly ($K = 4M$ in Fig. 10.2 c). A second question follows: how does over-parameterisation help ReLU students achieve zero test error?

An analytical theory for DFA dynamics To answer these two questions, we study the dynamics of DFA in the limit of infinite training data where a previously unseen sample (x, y) is used to compute the DFA weight updates (10.4) at every step. This "online learning" or "one-shot/single-pass" limit of SGD has been widely studied in recent and classical works on vanilla BP [355–357, 363, 375–381].

We work in the regime where the input dimension $D \to \infty$, while M and K are finite. The test error (10.5), i.e. a function of the student and teacher parameters involving a high-dimensional average over inputs, can be simply expressed in terms of a *finite* number of "order parameters" $Q = (Q^{kl}), R = (R^{km}), T = (T^{mn})$,

$$\lim_{D\to\infty} \epsilon_g(\theta, \tilde{\theta}) = \epsilon_g(Q, R, T, W_2, \tilde{W}_2) \tag{10.6}$$

where

$$Q^{kl} = \frac{W_1^k W_1^l}{D},\ R^{km} = \frac{W_1^k \tilde{W}_1^m}{D},\ T^{mn} = \frac{\tilde{W}_1^m \tilde{W}_1^n}{D} \tag{10.7}$$

as well as second layer weights $\tilde{W}_2^m$ and W_2^k [76, 355–357]. Intuitively, R^{km} quantifies the similarity between the weights of the student's kth hidden unit and the teacher's mth hidden unit. The self-overlap of the kth and lth student nodes is given by Q^{kl}, and likewise T^{mn} gives the (static) self-overlap of teacher nodes.

In seminal work, [355] and [357] obtained a closed set of ordinary differential equations (ODEs) for the time evolution of the order parameters Q and R. Our first main contribution is to extend their approach to the DFA setup (see SM H.1 for the details), obtaining a set of ODEs (H.7) that predicts the test error of a student trained using DFA (10.4) at all times. The accuracy of the predictions from the ODEs is demonstrated in Fig. 10.3 a, where the comparison between a single simulation of training a two-layer net with BP (orange) and DFA (blue) and theoretical predictions yields perfect agreement.

10.2.1 Sigmoidal networks learn through "degeneracy breaking"

The test error of a sigmoidal student trained on a teacher with the same number of neurons as herself ($K = M$) contains several global minima, which all correspond to fixed points of the ODEs (H.7). Among these is a student with exactly the same weights as her teacher. The symmetry $\mathrm{erf}(z) = -\mathrm{erf}(-z)$ induces a student with weights $\{\tilde{W}_1, \tilde{W}_2\}$ to have the same test error as a sigmoidal student with weights $\{-\tilde{W}_1, -\tilde{W}_2\}$. Thus, as illustrated in Fig. 10.3 c, the problem of learning a teacher has various degenerate solutions. A student trained with vanilla BP converges to any one of these solutions, depending on the initial conditions.

Alignment phase A student trained using DFA has to fulfil the same objective (zero test error), with an additional constraint: her second-layer weights W_2 needs to align with the feedback vector F_1 to ensure the first-layer weights are updated in the direction that minimises the test error. And indeed, an analysis of the ODEs (cf. Sec. H.2) reveals that in the early phase of training, $\dot{W}_2 \sim F$ and so W_2 grows in the direction of the feedback vector F_1 resulting in an increasing overlap between W_2 and F_1. In this *alignment phase* of learning, shown in Fig. 10.3 b, W_2 becomes perfectly aligned with F_1. DFA has perfectly recovered the weight updates for W_1 of BP, but the second layer has lost its expressivity (it is simply aligned to the random feedback vector).

Memorisation phase The expressivity of the student is restored in the *memorisation* phase of learning, where the second layer weights move away from F_1 and towards the global miminum of the test error that maintains the highest overlap with the feedback vector. In other words, students solve this constrained optimisation problem by consistently converging to the global minimum of the test error that simultaneously maximises the overlap between W_2 and F_1, and thus between the DFA gradient and the BP gradient. For DFA, the global minima of the test error are not equivalent, this "degeneracy breaking" is illustrated in Fig. 10.3 c.

10.2.2 Degeneracy breaking requires over-parametrisation for ReLU networks

The ReLU activation function has a very different kind of symmetry to sigmoidal activation functions. It posses the continuous symmetry $\max(0, x) = \gamma \max(0, x/\gamma)$ for any $\gamma > 0$, however it cannot compensate a change of sign of W_2^k with a change of sign of W_1^k. Consequently, a ReLU student can

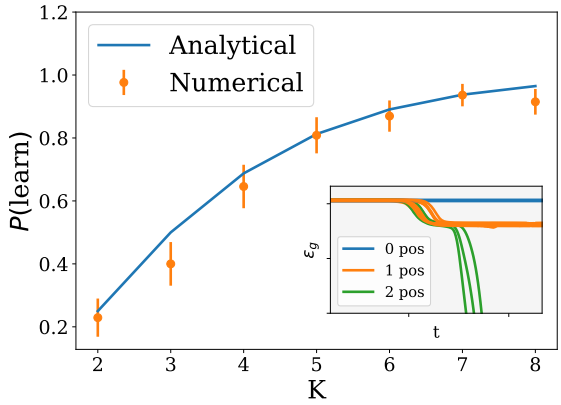


Figure 10.4: **Over-parameterisation improves performance of shallow ReLU networks.** We show the learning dynamics of a student with $K = 3$ hidden nodes trained on a teacher with $M = 2$ nodes and $\tilde{W}_2^m = 1$ if the feedback vector has 0, 1, or 2 positive entries. *Inset*: Probability of achieving zero test error (Eq. 10.8, line) compared to the fraction of simulations that converged to zero test error (out of 50, crosses). *Other parameters*: $D = 500, \eta = 0.1, \sigma_0 = 10^{-2}$.

only simultaneously align to the feedback vector F_1 and recover the teacher's second layer $\tilde{W}_2$ if at least M elements of F_1 have the same sign as $\tilde{W}_2$. The inset of Fig. 10.4 shows that a student trained on a teacher with $M = 2$ second-layer weights $\tilde{W}_2^m = 1$ only converges to zero test error if the feedback vector has 2 positive elements (green). If instead the feedback vector has only 0 (blue) or 1 (orange) positive entry, the student will settle at a finite test error. More generally, the probability of perfect recovery for a student with $K \geq M$ nodes sampled randomly is given analytically as:

$$\mathbb{P}(\text{learn}) = \frac{1}{2^K} \sum_{k=0}^{M} \binom{K}{k}. \tag{10.8}$$

As shown in Fig. 10.4, this formula matches with simulations. Note that the importance of the "correct" sign for the feedback matrices was also observed in deep neural networks by [382].

10.2.3 Degeneracy breaking in deep networks

We explore to what extent degeneracy breaking occurs in deep nonlinear networks by training 4-layer multi-layer perceptrons (MLPs) with 100 nodes per layer for 1000 epochs with both BP and DFA, on the MNIST and CIFAR10 datasets, with Tanh and ReLU nonlinearities (cf. App. H.5.2 for further experimental details). The dynamics of the training loss, shown in the left of Fig. 10.5, are very similar for BP and DFA.

From degeneracy breaking, one expects DFA to drive the optimization path towards a special region of the loss landscape determined by the feedback matrices. We test this hypothesis by measuring whether networks trained with the same feedback matrices from different initial weights converge towards the same region of the landscape. The cosine similarity between the vectors obtained by stacking the weights of two networks trained independently using BP reaches at most 10^{-2} (right of Fig. 10.5), signalling that they reach very distinct minima. In contrast, when trained with DFA, networks reach a cosine similarity

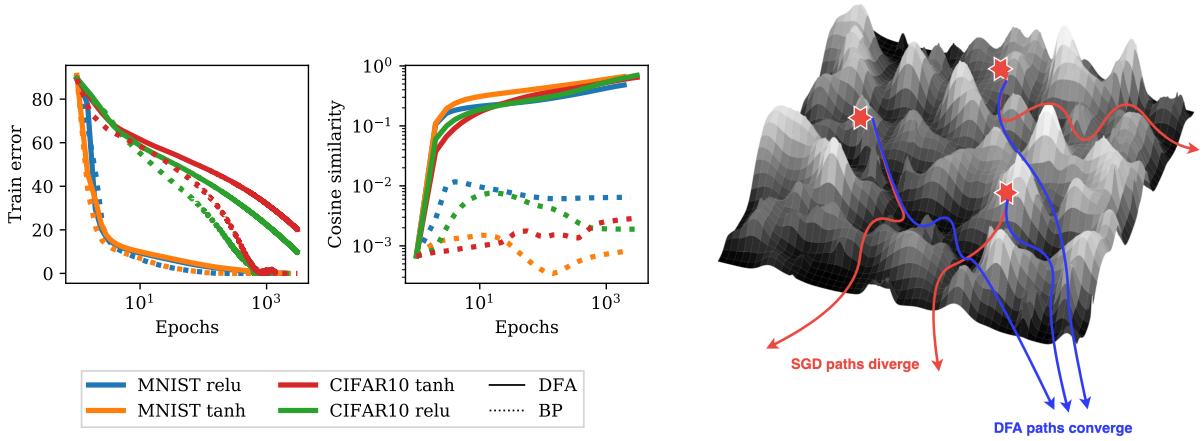


Figure 10.5: **Degeneracy breaking also occurs in deep neural networks.** (**Left**) We plot the training accuracy and the cosine similarity between the weights of four-layer fully-connected neural networks with sigmoidal and ReLU activations during training on MNIST and CIFAR10. Averages taken over 10 runs; for exp. details see Sec. 10.2.3. (**Right**) Cartoon of the degeneracy breaking process in the loss landscape of a deep network: while the optimization paths of models trained with SGD diverge in the loss landscape, with DFA they converge to a region of the landscape determined by the feedback matrices.

between 0.5 and 1 at convergence, thereby confirming that DFA breaks the degeneracy between the solutions in the landscape and biases towards a special region of the loss landscape, both for sigmoidal and ReLU activation functions.

This result suggests that heavily over-parametrised neural networks used in practice can be trained successfully with DFA because they have a large number of degenerate solutions. We leave a more detailed exploration of the interplay between DFA and the loss landscape for future work. As we will discuss in Sec. 10.4, the Align-then-Memorise mechanism sketched in Fig. 10.3 c also occurs in deep non-linear networks.

10.3 How do gradients align in deep networks?

This section focuses on the alignment phase of learning. In the two-layer setup there is a single feedback vector F_1, of same dimensions as the second layer W_2, and to which W_2 must align in order for the first layer to recover the true gradient.

In deep networks, as each layer W_l has a distinct feedback matrix F_l of different size of W_l, it is not obvious how the weights must align to ensure gradient alignment. We study how the alignment occurs by considering deep linear networks with L layers without bias, without any assumption on the training data. While the expressivity of linear networks is naturally limited, their learning dynamics is non-linear and rich enough to give insights that carry over to the non-linear case both for BP [72, 73, 364, 383, 384] and for DFA [347, 348, 358].

10.3.1 Weight alignment as a natural structure

In the following, we assume that the weights are initialised to zero. With BP, they would stay zero at all times, but for DFA the layers become nonzero sequentially, from the bottom to the top layer. In the linear setup, the updates of the first two layers at time t can be written in terms of the corresponding input and error vectors using Eq. (10.4)[1]:

$$\delta W_1^t = -\eta(F_1 e_t)x_t^T, \quad \delta W_2^t = -\eta(F_2 e_t)(W_1 x_t)^\top \tag{10.9}$$

Summing these updates shows that the first layer performs Hebbian learning modulated by the feedback matrix F_1:

$$W_1^t = -\eta \sum_{t'=0}^{t-1} F_1 e_{t'} x_{t'}^\top = F_1 A_1^t, \tag{10.10}$$

$$A_1^t = -\eta \sum_{t'=0}^{t-1} e_{t'} x_{t'}^\top \tag{10.11}$$

Plugging this result into δW_2^t, we obtain:

$$W_2^t = -\eta \sum_{t'=0}^{t-1} F_2 e_t (A_1^{t'} x_{t'})^\top F_1^\top = F_2 A_2^t F_1^\top, \tag{10.12}$$

$$A_2^t = \eta^2 \sum_{t'=0}^{t-1} \sum_{t''=0}^{t'-1} (x_{t'} \cdot x_{t''}) e_{t'} e_{t''}^\top. \tag{10.13}$$

When iterated, the procedure above reveals that DFA naturally leads to *weak weight alignment* of the network weights to the feedback matrices:

$$\textbf{Weak WA: } W_{1<l<L}^t = F_l A_l^t F_{l-1}^\top, \quad W_L^t = A_L^t F_{L-1}^\top, \tag{10.14}$$

where we defined the *alignment matrices* $A_{l\geq 2}^t \in \mathbb{R}^{C\times C}$ (C being the output dimension):

$$A_{l\geq 2}^t = \eta^2 \sum_{t'=0}^{t-1} \sum_{t''=0}^{t'-1} (B_l^{t'} x_{t'}) \cdot (B_l^{t''} x_{t''}) e_{t'} e_{t''}^\top. \tag{10.15}$$

$B_l \in \mathbb{R}^{C\times C}$ is defined recursively as a function of the feedback matrices only and its expression together with the full derivation is deferred to App. H.3. These results can be adapted both to DRTP [358], another variant of feedback alignment where $e_t = -y_t$ and to FA by performing the replacement $F_l \to F_l F_{l+1} \dots F_{L-1}$.

10.3.2 Weight alignment leads to gradient alignment

Weak WA builds throughout training, but does not directly imply GA. However, if the alignment matrices become proportional to the identity, we obtain *strong weight alignment*:

$$\textbf{Strong WA: } \quad W_{1<l<L}^t \propto F_l F_{l-1}^\top, \quad W_L^t \propto F_{L-1}^\top. \tag{10.16}$$

[1] We implicitly assume a minibatch size of 1 for notational simplicity, but conclusions are unchanged in the finite-batch setup.

Additionally, since GA requires $F_l e \propto W_{l+1}^\top \delta a_{l+1}$ (Eqs. 10.4 and 10.2), strong WA directly implies GA if the feedback matrices $F_{l\geq 2}$ are assumed left-orthogonal, i.e. $F_l^\top F_l = I_C$. Strong WA of (10.16) induces the weights, by the orthogonality condition, to cancel out by pairs of two:

$$W_{l+1}^\top \delta a_{l+1} \propto F_l F_{l+1}^\top F_{l+1} \dots F_{L-1}^\top F_{L-1} e = F_l e. \tag{10.17}$$

The above suggests that taking the feedback matrices left-orthogonal is favourable for GA. If the feedback matrices elements are sampled i.i.d. from a Gaussian distribution, GA still holds in expectation since $\mathbb{E}\left[F_l^\top F_l\right] \propto I_C$.

Quantifying gradient alignment Our analysis shows that key to GA are the alignment matrices: the closer they are to identity, i.e. the better their conditioning, the stronger the GA. This comes at the price of restricted expressivity, since layers are encouraged to align to a product of (random) feedback matrices. In the extreme case of strong WA, the freedom of layers $l \geq 2$ is entirely sacrificed to allow learning in the first layer! This is not harmful for the linear networks as the first layer alone is enough to maintain full expressivity[2]. Nonlinear networks, as argued in Sec. 10.2, rely on the Degeneracy Breaking mechanism to recover expressivity.

10.4 The case of deep nonlinear networks

In this section, we show that the theoretical predictions of the previous two sections hold remarkably well in deep nonlinear networks trained on standard vision datasets.

10.4.1 Weight Alignment occurs like in the linear setup

To determine whether WA described in Sec. 10.3 holds in the deep nonlinear setup of Sec. 10.2.3, we introduce the global and layerwise alignment observables:

$$\mathrm{WA} = \measuredangle\left(\mathbf{F}, \mathbf{W}\right), \qquad \mathrm{GA} = \measuredangle\left(\mathbf{G}^{\mathrm{DFA}}, \mathbf{G}^{\mathrm{BP}}\right) \tag{10.18}$$

$$\mathrm{WA}_{l\geq 2} = \measuredangle(\mathbf{F}_l, \mathbf{W}_l), \qquad \mathrm{GA}_{l\geq 2} = \measuredangle\left(\mathbf{G}_l^{\mathrm{DFA}}, \mathbf{G}_l^{\mathrm{BP}}\right), \tag{10.19}$$

where $\measuredangle(\mathbf{A}, \mathbf{B}) = \mathrm{Vec}(\mathbf{A}) \cdot \mathrm{Vec}(\mathbf{B}) / \|\mathbf{A}\| \|\mathbf{B}\|$ and

$$\begin{aligned}
\mathbf{F} &= \left(F_2 F_1^\top, \dots, F_{L-1} F_{L-2}^\top, F_{L-1}^\top\right), \\
\mathbf{W}(t) &= \left(W_2^t, \dots, W_{L-1}^t, W_L^t\right), \\
\mathbf{G}(t) &= \left(\delta a_1^t, \dots, \delta a_{L-1}^t\right).
\end{aligned}$$

Note that the layer-wise alignment of W_l with $F_l F_{l-1}^\top$ was never measured before: it differs from the alignment of F_l with $W_{l+1} \dots W_L$ observed in [386], which is more akin to GA.

If $\mathbf{W}$ and $\mathbf{F}$ were uncorrelated, the WA defined in (10.18) would be vanishing as the width of the layer grows large. Remarkably, WA becomes of order one after a few epochs as shown in Fig. 10.6 (left), and strongly correlates with GA (right). This suggests that the layer-wise WA uncovered for linear networks with weights initialized to zero also drives GA in the general case.

[2] such an alignment was indeed already observed in the linear setup for BP [385].

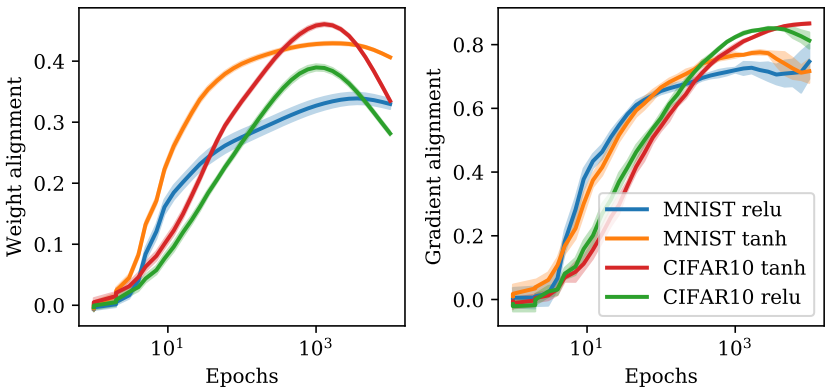


Figure 10.6: **Global alignment dynamics of deep nonlinear networks exhibits Align-then-Memorise.** Global weight and gradient alignments, as defined in (10.18), varying the activation function and the dataset. Shaded regions represent the (small) variability over 10 runs.

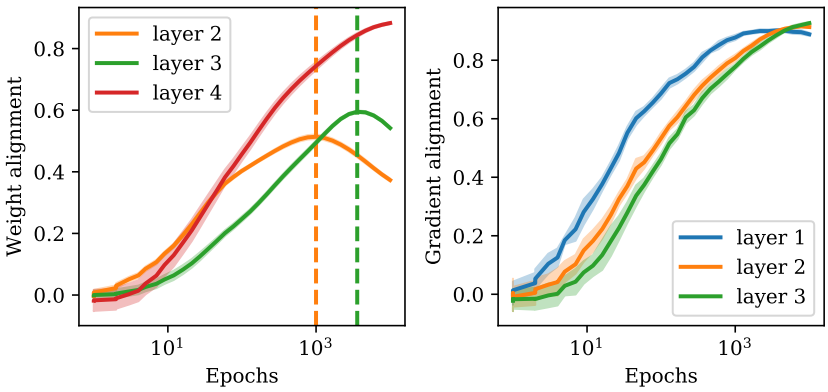


Figure 10.7: **Layerwise alignment dynamics reveal sequential Align-then-Memorise.** Layerwise weight and gradient alignments as defined in (10.19), for a ReLU network trained on CIFAR10 with 10% label corruption. Shaded regions represent the (small) variability over 10 runs.

10.4.2 Align-then-Memorise occurs from bottom layers to top

As can be seen in Fig. 10.6, WA clearly reaches a maximum then decreases, as expected from the Align-then-Memorise process. Notice that the decrease is stronger for CIFAR10 than it is for MNIST, since CIFAR-10 is much harder to fit than MNIST: more WA needs to be sacrificed. Increasing label corruption similarly makes the datasets harder to fit, and decreases the final WA, as detailed in SM H.5.2. However, another question arises: why does the GA keep increasing in this case, in spite of the decreasing WA?

To answer this question, we need to disentangle the dynamics of the layers of the network, as in Eq. (10.19). In Fig. 10.7, we focus on the ReLU network applied to CIFAR10, and shuffle 10% of the labels in the training set to make the Align-then-Memorise procedure more easily visible. Although the network contains 4 layers of weights, we only have 3 curves for WA and GA: WA is only defined for layers 2 to 4 according to Eq. (10.19), whereas GA of the last layer is not represented here since it is always equal to one.

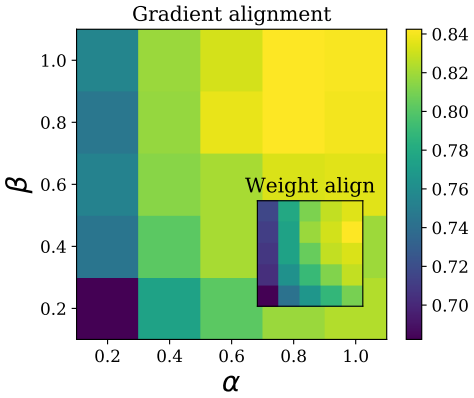


Figure 10.8: **Badly conditioned output statistics can hamper alignment.** WA and GA at the final point of training decrease when the output classes are correlated ($\beta < 1$) or of different variances ($\alpha < 1$).

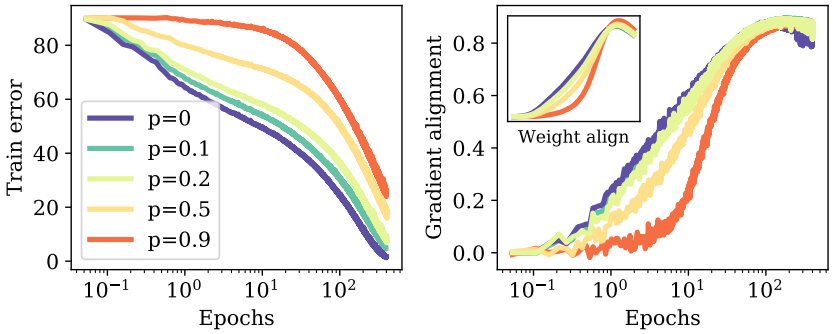


Figure 10.9: **Label corruption hampers alignment in the early stages of training.** We see that the higher the label corruption, the more time WA and GA take to start increasing, since the network initially predicts equal probabilities over the output classes.

As can be seen, the second layer is the first to start aligning: it reaches its maximal WA around 1000 epochs (orange dashed line), then decreases. The third layer starts aligning later and reaches its maximal WA around 2000 epochs (green dashed line), then decreases. As for the last layer, the WA is monotonically increasing. Hence, the Align-then-Memorise mechanism operates in a layerwise fashion, starting from the bottom layers to the top layers.

Note that the WA of the last layers is the most crucial, since it affects the GA of all the layers below, whereas the WA of the second layer only affects the GA of the first layer. It therefore makes sense to keep the WA of the last layers high, and let the bottom layers perform the memorization first. This is reminiscent of the linear setup, where all the layers align except for the first, which does all the learning. In fact, this strategy enables the GA of each individual layer to keep increasing until late times: the diminishing WA of the bottom layers is compensated by the increasing WA of the top layers.

10.5 What can hamper alignment?

We demonstrated that GA is enabled by the WA mechanism, both theoretically for linear networks and numerically for nonlinear networks. In this section, we leverage our analysis of WA to identify situations in which GA fails.

10.5.1 Alignment is data-dependent

In the linear case, GA occurs if the alignment matrices presented in Sec. 10.3 are well conditioned. Note that if the output size C is equal to one, e.g. for scalar regression or binary classification tasks, then the alignment matrices are simply scalars, and GA is guaranteed. When this is not the case, one can obtain the deviation from GA by studying the expression of the alignment matrices (10.15). They are formed by summing outer products of the error vectors $e_{t'}e_{t''}^\top$, where $e_t = \hat{y}_t - y_t$. Therefore, good conditioning requires the different components of the errors to be uncorrelated and of similar variances. This can be violated by (i) the targets y, or (ii) the predictions $\hat{y}$.

(i) Structure of data The first scenario can be demonstrated in a simple regression task on i.i.d. Gaussian inputs $x \sim \mathbb{R}^{10}$. The targets $y \in \mathbb{R}^2$ are randomly sampled from the following distribution:

$$y \sim \mathcal{N}(0, \Sigma), \quad \Sigma = \begin{pmatrix} 1 & \alpha(1-\beta) \\ \alpha(1-\beta) & \alpha^2 \end{pmatrix}, \quad \alpha, \beta \leq 1. \tag{10.20}$$

In Fig. 10.8, we show the final WA and GA of a 3-layer ReLU network trained for 10^3 epochs on 10^3 examples sampled from this distribution (further details in SM H.5.3). As predicted, imbalanced ($\alpha < 1$) or correlated ($\beta < 1$) target statistics hamper WA and GA. Note that the inputs also come into play in Eq. (10.15): a more detailed theoretical analysis of the impact of input and target statistics on alignment is deferred to SM H.4.

(ii) Effect of noise For classification tasks, the targets y are one-hot encodings whose statistics are naturally well conditioned. However, alignment can be degraded if the statistics of the predictions $\hat{y}$ become correlated.

One can enforce such a correlation in CIFAR10 by shuffling a fraction p of the labels. The WA and GA dynamics of a 3-layer ReLU network are shown in Fig. 10.9. At high p, the network can only perform random guessing during the first few epochs, and assigns equal probabilities to the 10 classes. The correlated structure of the predictions prevents alignment until the network starts to fit the random labels: the predictions of the different classes then decouple and WA takes off, leading to GA.

10.5.2 Alignment is impossible for convolutional layers

As we have seen in Chap. 6, a convolutional layer with filters H_l can be represented by a large fully-connected layer whose weights are represented by a block Toeplitz matrix $\phi(H_l)$. This matrix has repeated blocks due to weight sharing, and most of its weights are equal to zero due to locality. In order to verify WA and therefore GA, the following condition must hold: $\phi(H_l) \propto F_l F_{l-1}^\top$. Yet, due to the very constrained structure of $\phi(H_l)$, this is impossible for a general choice of F_l. Therefore, the WA mechanism suggests a simple explanation for why GA doesn't occur in vanilla CNNs, and confirms the previously stated hypothesis that CNNs don't have enough flexibility to align [353].

In the case of convolutional layers, this lack of alignment makes learning near to impossible, and has lead practitioners to design alternatives [352, 354]. However, the extent to which alignment correlates with good performance in the general setup (both in terms of fitting and generalisation) is a complex question which we leave for future work. Indeed, nothing prevents DFA from finding a good optimization path, different from the one followed by BP. Conversely, obtaining high gradient alignment at the end of training is not a sufficient condition for DFA to retrieve the results of BP, e.g. if the initial trajectory leads to a wrong direction.

10.6 Conclusion

In this chapter, we have analyzed the success and failure modes of the DFA algorithm, which stands as one of the most promising algorithms for biologically-plausible deep learning.

We have shown that their ability to approximate the gradient of the loss function relies on a *weight alignment* mechanism whereby the weights of the network adapt to the feedback matrices, generally in the ealy stages of learning. This mechanism imposes a strong constraint on the learning dynamics which can prevent learning in architectures with restricted expressivity such as CNNs.

However, it also provides us with some theoretical tools to solve cases where DFA fails: in future work, it would be interesting to study whether appropriate choices of the intial weights or feedback matrices could ease the dynamics.

Part IV

Deep learning for Symbolic Regression

Chapter 11

Deep Symbolic Regression for Recurrent Sequences

Symbolic regression, i.e. predicting a function from the observation of its values, is well-known to be a challenging task. In this chapter, we train Transformers to infer the function or recurrence relation underlying sequences of integers or floats, a typical task in human IQ tests which has hardly been tackled in the machine learning literature. We evaluate our integer model on a subset of OEIS sequences, and show that it outperforms built-in Mathematica functions for recurrence prediction. We also demonstrate that our float model is able to yield informative approximations of out-of-vocabulary functions and constants, e.g. $\mathrm{bessel0}(x) \approx \frac{\sin(x)+\cos(x)}{\sqrt{\pi x}}$ and $1.644934 \approx \pi^2/6$.

11.1 Introduction

Given the sequence [1,2,4,7,11,16], what is the next term? Humans usually solve such riddles by noticing patterns in the sequence. In the easiest cases, one can spot a function: [1,4,9,16,25] are the first five squares, so the n-th term in the series is $u_n = n^2$, and the next one is 36. Most often however, we look for relations between successive terms: in the sequence [1,2,4,7,11,16], the differences between successive values are 1, 2, 3, 4, and 5, which makes it likely that the next term will be $16 + 6 = 22$. Mathematically, we are inferring the recurrence relation $u_n = u_{n-1} + n$, with $u_0 = 1$.

In all cases, we handle such problems as symbolic regressions: starting from a sequence of numbers, we try to discover a function or a recurrence relation that they satisfy, and use it to predict the next terms. This can lead to very challenging problems as the complexity of the unknown recurrence relation $u_n = f(n, \{u_i\}_{i<n})$ increases, e.g. $u_n = \tan^{-1}(u_{n-3}) \exp(\cos(n^2))$.

In this chapter, we train neural networks to infer the recurrence relation f from the observation of the first terms of the sequence. The majority of studies in machine learning for symbolic regression have focused on non-recurrent functions, i.e. expressions of the form $y = f(x)$. Recurrence relations provide a more general setting, which gives a deeper account of the underlying process: if the sequence corresponds to successive time steps, the recurrence relation is a discrete time differential equation for the system considered.

OEIS	Description	First terms	Predicted recurrence
A000792	$a(n) = \max\{(n-i)a(i), i < n\}$	1, 1, 2, 3, 4, 6, 9, 12, 18, 27	$u_n = u_{n-1} + u_{n-3} - u_{n-1}\%u_{n-3}$
A000855	Final two digits of 2^n	1, 2, 4, 8, 16, 32, 64, 28, 56, 12	$u_n = (2u_{n-1})\%100$
A006257	Josephus sequence	0, 1, 1, 3, 1, 3, 5, 7, 1, 3	$u_n = (u_{n-1} + n)\%(n-1) - 1$
A008954	Final digit of $n(n+1)/2$	0, 1, 3, 6, 0, 5, 1, 8, 6, 5	$u_n = (u_{n-1} + n)\%10$
A026741	$a(n) = n$ if n odd, $n/2$ if n even	0, 1, 1, 3, 2, 5, 3, 7, 4, 9	$u_n = u_{n-2} + n//(u_{n-1} + 1)$
A035327	n binary, switch 0's and 1's, then decimal	1, 0, 1, 0, 3, 2, 1, 0, 7, 6	$u_n = (u_{n-1} - n)\%(n-1)$
A062050	n-th chunk contains numbers $1, ..., 2^n$	1, 1, 2, 1, 2, 3, 4, 1, 2, 3	$u_n = (n\%(n - u_{n-1})) + 1$
A074062	Reflected Pentanacci numbers	5, -1, -1, -1, -1, 9, -7, -1, -1, -1	$u_n = 2u_{n-5} - u_{n-6}$

Table 11.1: **Our integer model yields exact recurrence relations on a variety of interesting OEIS sequences.** Predictions are based on observing the first 25 terms of each sequence.

11.1.1 Contributions

We show that transformers can learn to infer a recurrence relation from the observation of the first terms of a sequence. We consider both sequences of integers and floats, and train our models on a large set of synthetic examples.

We first demonstrate that our symbolic regression model can predict complex recurrence relations that were not seen during training. We also show that those recurrence relations can be used to extrapolate the next terms of the sequence with better precision than a "numeric" model of similar architecture, trained specifically for extrapolation.

We then test the out-of-domain generalization of our models. On a subset of the Online Encyclopedia of Integer Sequences, our integer models outperform built-in Mathematica functions, both for sequence extrapolation and recurrence prediction, see Table 11.1 for some examples. We also show that our symbolic float model is capable of predicting approximate expressions for out-of-vocabulary functions and constants (e.g. $\text{bessel0}(x) \approx \frac{\sin(x)+\cos(x)}{\sqrt{\pi x}}$ and $1.644934 \approx \pi^2/6$), see Tables 11.2, 11.3 for more examples.

We conclude by discussing potential limitations of our models and future directions.

Reproducibility We provide an interactive demonstration of our models at https://symbolicregression.metademolab.com/. An open-source implementation of our code will be released publicly at https://github.com/facebookresearch/recur.

11.1.2 Related work

AI for maths The use of neural networks for mathematics has recently gained attention: several works have demonstrated their surprising reasoning abilities [387, 388], and have even sparked some interesting mathematical discoveries [389]. In particular, four types of tasks have recently been tackled in the deep learning literature.

First, converting a symbolic expression to another symbolic expression. Direct examples are integration and differentation [390], expression simplification [391] or equation solving [392]. Second, converting a symbolic expression to numerical data. This includes predicting quantitative properties of a mathematical structure, for example the local stability of a differential system [393]. Third, converting

Constant	Approximation	Rel. error
0.3333	$(3+\exp(-6))^{-1}$	10^{-5}
0.33333	$1/3$	10^{-5}
3.1415	$2\arctan(\exp(10))$	10^{-7}
3.14159	π	10^{-7}
1.6449	$1/\arctan(\exp(4))$	10^{-7}
1.64493	$\pi^2/6$	10^{-7}
0.123456789	$10/9^2$	10^{-9}
0.987654321	$1-(1/9)^2$	10^{-11}

Table 11.2: **Our float model learns to approximate out-of-vocabulary prefactors with its own vocabulary.** We obtain the approximation of each constant C by feeding our model the 25 first terms of $u_n = Cn$.

Expression u_n	Approximation $\hat{u}_n$	Comment
$\operatorname{arcsinh}(n)$	$\log(n+\sqrt{n^2+1})$	Exact
$\operatorname{arccosh}(n)$	$\log(n+\sqrt{n^2-1})$	Exact
$\operatorname{arctanh}(1/n)$	$\frac{1}{2}\log(1+2/n)$	Asymptotic
$\operatorname{catalan}(n)$	$u_{n-1}(4-6/n)$	Asymptotic
$\operatorname{dawson}(n)$	$\frac{n}{2n^2-u_{n-1}-1}$	Asymptotic
$\mathrm{j0}(n)$ (Bessel)	$\frac{\sin(n)+\cos(n)}{\sqrt{\pi n}}$	Asymptotic
$\mathrm{i0}(n)$ (mod. Bessel)	$\frac{e^n}{\sqrt{2\pi n}}$	Asymptotic

Table 11.3: **Our float model learns to approximate out-of-vocabulary functions.** For simple functions, our model predicts an exact expression; for complex functions, our model manages to predict the first order of the asymptotic expansion.

numerical data to numerical data using mathematical rules. Examples range from learning basic arithmetics [394, 395] to linear algebra [396]. Fourth, converting numerical data to a symbolic expression: this is the framework of symbolic regression, which we will be focusing on.

Symbolic regression Two types of approaches exist for symbolic regression, which one could name selection-based and pretraining-based.

In selection-based approaches, we only have access to one set of observations: the values of the function we are trying to infer. Typical examples of this approach are evolutionnary programs, which have long been the *de facto* standard for symbolic regression [397–400], despite their computational cost and limited performance. More recently, neural networks were used following this approach [401–403].

In pretraining-based approaches, we train neural networks on a large dataset containing observations from many different function [404, 405], hoping that the model can generalize to new expressions. Although the pretraining is computationally expensive, inference is much quicker as one simply needs to perform a forward pass, rather than search through a set of functions. In this chapter and the next, we choose this approach.

Recurrent sequences All studies on symbolic regression cited above consider the task of learning non-recurrent functions from their values at a set of points. Our contribution is, to the best of our knowledge, the first to tackle the setup of recurrent expressions. Although this includes non-recurrent expressions as a special case, one slight restriction is that the inputs need to be sampled uniformly. Hence, instead of feeding the model with input-output pairs $\{(x_i, y_i)\}$, we only need to feed it the terms of the sequence $\{u_i\}$. Another important difference is that order of these terms matters, hence permutation invariant representations [405] cannot be used.

Integer sequences, in particular those from the Online Encyclopedia of Integer Sequences (OEIS) [406], have been studied using machine learning methods in a few recent papers. [407] trains various classifiers to predict the label of OEIS sequences, such as “interesting”, “easy”, or “multiplicative”. [408] use embeddings trained on OEIS to investigate the mathematical properties of integers. [409] use fully connected networks to predict the next term of OEIS sequences (a numeric rather than symbolic

regression task). [410] investigate different architectures (most of them recurrent networks) for digit-level numeric regression on sequences such as Fibonacci, demonstrating the difficulty of the task.

11.2 Methods

Broadly speaking, we want to solve the following problem: given a sequence of n points $\{u_0, \dots, u_{n-1}\}$, find a function f such that for any $i \in \mathbb{N}$, $u_i = f(i, u_{i-1}, \dots, u_{i-d})$, where d is the recursion degree. Since we cannot evaluate at an infinite number of points, we declare that a function f is a solution of the problem if, given the first n_{input} terms in the sequence, it can predict the next n_{pred} following ones.

Under this form, the problem is underdetermined: given a finite number of input terms, an infinite number of recurrence relations exist. However in practice, one would like the model to give us the simplest solution possible, following Occam's principle. This is generally ensured by the fact that simple expressions are more likely to be generated during training.

Integer vs. float We consider two settings for the numbers in the sequences: integers and floats. For integer sequences, the recurrence formula only uses operators which are closed in $\mathbb{Z}$ (e.g. $+, \times, \mathrm{abs}$, modulo and integer division). For float sequences, additional operators and functions are allowed, such as real division, $\exp$ and $\cos$ (see Table 11.4 for the list of all used operators). These two setups are interesting for different reasons. Integer sequences are an area of strong interest in mathematics, particular for their relation with arithmetics. The float setup is interesting to see how our model can generalize to a larger set of operators, which provides more challenging problems.

Symbolic vs. numeric We consider two tasks: symbolic regression and numeric regression. In symbolic regression, the model is tasked to predict the recurrence relation the sequence was generated from. At test time, this recurrence relation is evaluated by how well it approximates the n_{pred} following terms in the sequence.

In numeric regression, is tasked to directly predict the values of the n_{pred} following terms, rather than the underlying recurrence relation. At test time, the model predictions are compared with the true values of the sequence.

	Integer	Float
Unary	`abs, sqr, sign, relu`	`abs, sqr, sqrt, inv, log, exp, sin, cos, tan, atan`
Binary	`add, sub, mul, intdiv, mod`	`add, sub, mul, div`

Table 11.4: **Operators used in our generators.**

11.2.1 Data generation

All training examples are created by randomly generating a recurrence relation, randomly sampling its initial terms, then computing the next terms using the recurrence relation. More specifically, we use the steps below:

1. Sample the **number of operators** o between 1 and o_{max}, and build a unary-binary tree with o nodes, as described in [390]. The number of operators determines the difficulty of the expression.

2. Sample the **nodes** of the tree from the list of operators in Table 11.4. Note that the float case uses more operators than the integer case, which makes the task more challenging by expanding the problem space.

3. Sample the **recurrence degree** d between 1 and d_{max}, which defines the recurrence depth: for example, a degree of 2 means that u_{n+1} depends on u_n and u_{n-1}.

4. Sample the **leaves** of the tree: either a constant, with probability p_{const}, or the current index n, with probability p_n, or one of the previous terms of the sequence u_{n-i}, with $i \in [1, d]$, with probability p_{var}.

5. Recalculate the true recurrence degree d considering the deepest leaf u_{n-i} sampled during the previous step, then sample d **initial terms** from a random distribution $\mathcal{P}$.

6. Sample l between l_{min} and l_{max} and compute the next l terms of the sequence using the initial conditions. The total **sequence length** is hence $n_{input} = d_{\text{eff}} + l$.

We provide the values of the parameters of the generator in Table 11.5. Note that in the last step, we interrupt the computation if we encounter a term larger than 10^{100}, or outside the range of one of the operators: for example, a division by zero, or a negative square root.

Parameter	Description	Value
d_{max}	Max degree	6
o_{max}	Max number of operators	10
l_{min}	Min length	5
l_{max}	Max length	30
p_{const}	Prob of constant leaf	$1/3$
p_{index}	Prob of index leaf	$1/3$
p_{var}	Prob of variable leaf	$1/3$
$\mathcal{P}$	Distrib of first terms	$\mathcal{U}(-10, 10)$

Table 11.5: **Hyperparameters of our generator.**

11.2.2 Encodings

Model inputs are sequences of integers or floats. The outputs are recurrence relations for the symbolic task, and sequences of numbers for the numeric task. To be processed by transformers, inputs and outputs must be represented as sequences of tokens from a fixed vocabulary. To this effect, we use the encodings presented below.

Integers We encode integers using their base b representation, as a sequence of $1 + \lceil \log_b |x| \rceil$ tokens: a sign (which also serves as a sequence delimiter) followed by $\lceil \log_b |x| \rceil$ digits from 0 to $b - 1$, for a

total vocabulary of $b + 2$ tokens. For instance, $x = -325$ is represented in base $b = 10$ as the sequence `[-, 3, 2, 5]`, and in base $b = 30$ as `[-, 10, 25]`.

Choosing b involves a tradeoff between the length of the sequences fed to the Transformer and the vocabulary size of the embedding layer. We choose $b = 10,000$ and limit integers in our sequences to absolute values below 10^{100}, for a maximum of 26 tokens per integer, and a vocabulary of order 10^4 words.

Floats Following [396], we represent float numbers in base 10 floating-point notation, round them to four significant digits, and encode them as sequences of 3 tokens: their sign, mantissa (between `0` and `9999`), and exponent (from `E-100` to `E100`)[1]. For instance, $1/3$ is encoded as `[+, 3333, E-4]`. Again, the vocabulary is of order 10^4 words.

For all operations in floating-point representation, precision is limited to the length of the mantissa. In particular, when summing elements with different magnitudes, sub-dominant terms may be rounded away. Partly due to this effect, when approximating complex functions, our symbolic model typically only predicts the largest terms in its asymptotic expansion, as shown in Table 11.3. We discuss two methods for increasing precision when needed in Section J.4 of the Appendix.

Recurrence relations To represent mathematical trees as sequences, we enumerate the trees in prefix order, i.e. direct Polish notation, and encode each node as a single autonomous token. For instance, the expression $\cos(3x)$ is encoded as `[cos,mul,3,x]`.

Note that the generation procedure implies that the recurrence relation is not simplified (i.e. expressions like $1 + u_{n-1} - 1$ can be generated). We tried simplifying them using Sympy before the encoding step (see Appendix J.2), but this slows down generation without any benefit on the performance of our models, which turn out to be surprisingly insensitive to the syntax of the formulas.

11.2.3 Experimental details

Similarly to [390], we use a simple Transformer architecture [78] with 8 hidden layers, 8 attention heads and an embedding dimension of 512 both for the encoder and decoder.

Training and evaluation The tokens generated by the model are supervised via a cross-entropy loss. We use the Adam optimizer, warming up the learning rate from 10^{-7} to 2.10^{-4} over the first 10,000 steps, then decaying it as the inverse square root of the number of steps, following [78]. We train each model for a minimum of 250 epochs, each epoch containing 5M equations in batches of 512. On 16 GPU with Volta architecture and 32GB memory, one epoch is processed in about an hour.

After each epoch, we evaluate the in-distribution performance of our models on a held-out dataset of 10,000 equations. Unless specified otherwise, we generate expressions using greedy decoding. Note that nothing prevents the symbolic model from generating an invalid expression such as `[add,1,mul,2]`. These mistakes, counted as invalid predictions, tend to be very rare: in models trained for more than a few epochs, they occur in less than 0.1% of the test cases.

Hypothesis ranking To predict recurrence relations for OEIS sequences (Section 11.4.1), or the examples displayed in Tables 11.1,11.2,11.3, we used a beam size of 10. Usually, hypotheses in the beam are ranked according to their log-likelihood, normalized by the sequence length. For our symbolic model,

[1] By convention, we represent zero as `[+, 0, E0]`.

Model	Integer		Float	
	$n_{op} \leq 5$	$n_{op} \leq 10$	$n_{op} \leq 5$	$n_{op} \leq 10$
Symbolic	**92.7**	**78.4**	**74.2**	**43.3**
Numeric	83.6	70.3	45.6	29.0

Table 11.6: **Average in-distribution accuracies of our models.** We set $\tau = 10^{-10}$ and $n_{pred} = 10$.

we can do much better, by ranking beam hypotheses according to how well they approximate the initial terms of the original sequence. Specifically, given a recurrence relation of degree d, we recompute for each hypothesis the n_{input} first terms in the sequence (using the first d terms of the original sequence as initial conditions), and rank the candidates according to how well they approximate these terms. This ranking procedure is impossible for the numerical model, which can only perform extrapolation.

In some sense, this relates our method to evolutionary algorithms, which use the input terms for candidate selection. Yet, as shown by the failure modes presented in Section J.6 of the Appendix, our model is less prone to overfitting since the candidates are not directly chosen by minimizing a loss on the input terms.

11.3 In-domain generalization

We begin by probing the in-domain accuracy of our model, i.e. its ability to generalize to unseen sequences generated with the same procedure as the training data. As discussed in Section J.3 of the Appendix, the diversity of mathematical expressions and the random sampling of the initial terms ensures that almost all the examples presented at test time have not been seen during training: one cannot attribute the generalization to mere memorization.

Prediction accuracy Due to the large number of equivalent ways one can represent a mathematical expression, one cannot directly evaluate the accuracy by comparing (token by token) the predicted recurrence relation to the ground truth. Instead, we use the predicted expression to compute the next n_{pred} terms of the sequence $\{\hat{u}_i\}$, and compare them with those computed from the ground truth, $\{u_i\}$. The prediction accuracy is then defined as:

$$acc(n_{pred}, \tau) = \mathbb{P}\left(\max_{1 \leq i \leq n_{pred}} \left| \frac{\hat{u}_i - u_i}{u_i} \right| \leq \tau \right) \tag{11.1}$$

By choosing a small enough $\tau \geq 0$ and a large enough n_{pred}, one can ensure that the predicted formula matches the true formula. In the float setup, $\tau = 0$ must be avoided for two reasons: (i) equivalent solutions represented by different expressions may evaluate differently because due to finite machine precision, (ii) setting $\tau = 0$ would penalize the model for ignoring sub-dominant terms which are indetectable due to the finite precision encodings. Hence, we select $\tau = 10^{-10}$ and $n_{pred} = 10$ unless specified otherwise[2]. Occasionally, we will consider larger values of τ, to assess the ability of our model to provide approximate expressions.

[2]For the float numeric model, which can only predict values up to finite precision ϵ, we round the values of target function to the same precision. This explains the plateau of the accuracy at $\tau < \epsilon$ in Figure 11.1.

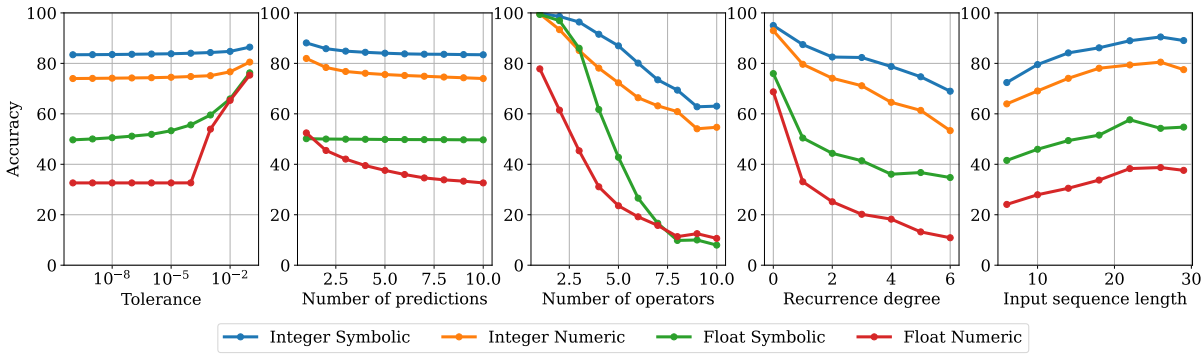


Figure 11.1: **The symbolic model extrapolates further and with higher precision than the numeric model.** From left to right, we vary the tolerance τ, the number of predictions n_{pred}, the number of operators o, the recurrence degree d and the number of input terms l. In each plot, we use the following defaults for quantities which are not varied: $\tau = 10^{-10}$, $n_{pred} = 10$, $o \in [\![1, 10]\!]$, $d \in [\![1, 6]\!]$, $l \in [\![5, 30]\!]$.

Results The average in-distribution accuracies of our models are reported in Table 11.6 with $\tau = 10^{-10}$ and $n_{pred} = 10$. Although the float setup is significantly harder that the integer setup, our symbolic model reaches good accuracies in both cases. In comparison, the numeric model obtains worse results, particularly in the float setup. The interested reader may find additional training curves, attention maps and visualizations of the model predictions in Appendix J.6.

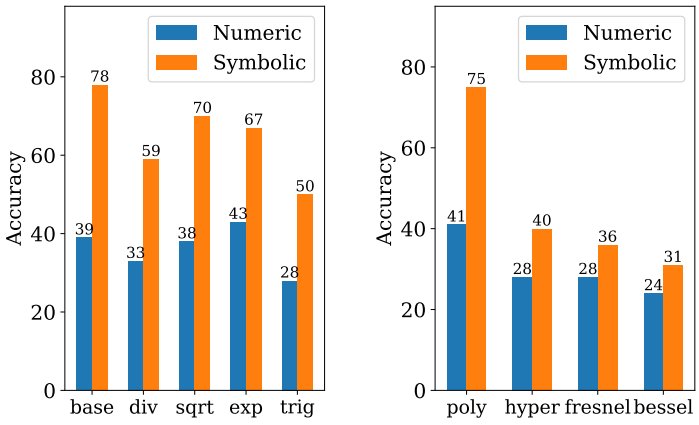
Ablation over the evaluation metric The two first panels of Figure 11.1 illustrate how the accuracy changes as we vary the tolerance level τ and the number of predictions n_{pred}. The first important observation is that the symbolic model performs much better than the numeric model at low tolerance. At high tolerance, it still keeps an advantage in the integer setup, and performs similarly in the float setup. This demonstrates the advantage of a symbolic approach for high-precision predictions.

Second, we see that the accuracy of both models degrades as we increase the number of predictions n_{pred}, as one could expect. However, the decrease is less important for the symbolic model, especially in the float setup where the curve is essentially flat. This demonstrates another strong advantage of the symbolic approach: once it has found the correct formula, it can predict the whole sequence, whereas the precision of the numeric model deteriorates as it extrapolates further.

Ablation over the example difficulty The three last panels of Figure 11.1 decompose the accuracy of our two models along three factors of difficulty: the number of operators o[3], the recurrence degree d and the sequence length n_{input} (see Section 11.2.1).

Unsurprisingly, accuracy degrades rapidly as the number of operators increases, particularly in the float setting where the operators are more diverse: the accuracy of the symbolic model drops from 100% for $o=1$ to 10% for $o=10$. We attempted a curriculum learning strategy to alleviate the drop, by giving higher probabilities to expressions with many operators as training advances, but this did not bring any improvement. Increasing the recurrence degree has a similar but more moderate effect: the accuracy decreases from 70% for $d=0$ (non-recurrent expressions) to 20% for $d=6$ in the float setup. Finally, we

[3]Since expressions are not simplified, o may be overestimated.



(a) In-domain (b) Out-of-domain

Figure 11.2: **Accuracy of our models on various in-domain and out-of-domain groups.** We set $\tau = 10^{-10}$, $n_{pred} = 10$.

observe that shorter sequences are harder to predict as they give less information on the underlying recurrence; however, even when fed with less than 10 terms, our models achieve surprisingly high accuracies.

Ablation over operator families To understand what kind of operators are the hardest for our float model, we bunch them into 5 groups:

- **base**: {add, sub, mul}.
- **division**: base + {div, inv}.
- **sqrt**: base + {sqrt}.
- **exponential**: base + {exp, log}.
- **trigonometric**: base + {sin, cos, tan, arcsin, arccos, arctan}.

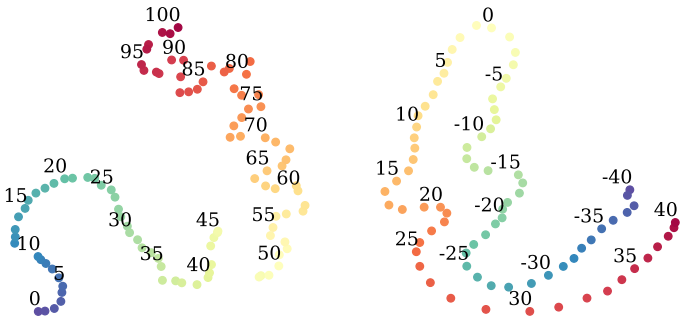
Results are displayed in Figure 11.2a. We see that the main difficulties lie in division and trigonometric operators, but the performance of both models stays rather good in all categories.

Visualizing the embeddings To give more intuition on the inner workings of our symbolic models, we display a t-SNE [411] projection of the embeddings of the integer model in Figure 11.3a and of exponent embeddings of the float model in Figure 11.3b.

Both reveal a sequential structure, with the embeddings organized in a clear order, as highlighted by the color scheme. In Appendix J.5, we study in detail the pairwise distances between embeddings, unveiling interesting features such as the fact that the integer model naturally learns a base-6 representation.

11.4 Out-of-domain generalization

In this section, we evaluate the ability of our model to generalize out-of-domain. Recurrence prediction being a previously unexplored branch of symbolic regression, there are no official benchmarks we can



(a) Integer embeddings (b) Exponent embeddings

Figure 11.3: **The number embeddings reveal intriguing mathematical structure**. We represented the t-SNE of the embeddings of the integer model and the exponent embeddings of the float model. We depicted the first 100 integer embeddings (10,000 in the model), and the exponent embeddings -40 to 40 (-100 to 100 in the model).

compare our models to. For integer sequences, we use a subset of OEIS as our out-of-domain benchmark; for float sequences, we use a generator with out-of-vocabulary constants and operators. In Appendix J.1, we also show that our models can and can be made robust noise in the inputs.

11.4.1 Integer sequences: OEIS dataset

The Online Encyclopedia of Integer Sequences (OEIS) is an online database containing over 300,000 integer sequences. It is tempting to directly use OEIS as a testbed for prediction; however, many sequences in OEIS do not have a closed-form recurrence relation, such as the stops on the New York City Broadway line subway (A000053). These will naturally cause our model to fail.

Preprocessing Luckily, OEIS comes with keywords, and 22% of the sequences are labelled as "easy", meaning that there is a logic to find the next terms (although this logic is by no means easy in most cases). Note that this logic cannot always be formulated as a recurrence relation: for example, the sequence of primes or decimals of π are included in this category, but intractable for our models. We keep the first 10,000 of these sequences as our testbed. Evaluation consists in in showing our models the first $n_{input} \in \{15, 25\}$ terms of each sequence and asking it to predict the $n_{pred} \in \{1, 10\}$ following terms.

Results Results are reported in Table 11.7. With only $n_{input} = 15$ terms, the numeric model reaches an impressive accuracy of 53% at next term prediction, and 27% for predicting the next ten terms. The symbolic model achieves lower results, with 33% and 19% respectively; we attribute this to the large number of non-analytic sequences in the testbed. Nonetheless, this means that our model can retrieve a valid recurrence relation for almost a fifth of the sequences, which is rather impressive: we give a few interesting examples in Table 11.1. Increasing n_{input} to 25 increases our performances rather marginally.

As a comparison, we ran two built-in Mathematica functions for the task at hand: FindSequenceFunction, which finds non-recurrent expressions, and FindLinearRecurrence, which finds linear recurrence relations. These functions are much more sensitive to the number of terms given as input: they obtain

similar accuracies at $n_{input} = 15$, but FindLinearRecurrence performs significantly better at $n_{input} = 25$, while FindSequenceFunction performs pathologically worse. Both these functions perform less well than our symbolic model in all cases.

Model	$n_{input} = 15$		$n_{input} = 25$	
	$n_{pred} = 1$	$n_{pred} = 10$	$n_{pred} = 1$	$n_{pred} = 10$
Symbolic (ours)	33.4	19.2	34.5	21.3
Numeric (ours)	53.1	27.4	54.9	29.5
FindSequenceFunction	17.1	12.0	8.1	7.2
FindLinearRecurrence	17.4	14.8	21.2	19.5

Table 11.7: **Accuracy of our integer models and Mathematica functions on OEIS sequences.** We use as input the first $n_{input} = \{15, 25\}$ first terms of OEIS sequences and ask each model to predict the next $n_{pred} = \{1, 10\}$ terms. We set the tolerance $\tau = 10^{-10}$.

11.4.2 Float sequences: robustness to out-of-vocabulary tokens

One major difficulty in symbolic mathematics is dealing with out-of-vocabulary constants and operators: the model is forced to approximate them using its own vocabulary. We investigate these two scenarios separately for the float model.

Model	$[\![-10, 10]\!] \cup \{e, \pi, \gamma\}$		$\mathcal{U}(-10, 10)$	
	$n_{op} \leq 5$	$n_{op} \leq 10$	$n_{op} \leq 5$	$n_{op} \leq 10$
Symbolic	**81.9**	**60.4**	60.1	42.1
Numeric	72.4	60.7	**72.2**	**60.2**

Table 11.8: **Our symbolic model can approximate out-of-vocabulary prefactors.** We report the accuracies achieved when sampling the constants uniformly from $[\![-10, 10]\!] \cup \{e, \pi, \gamma\}$, as during training, versus sampling uniformly in $[-10, 10]$. We set $\tau = 0.01$ (note the higher tolerance threshold as we are considering approximation) and $n_{pred} = 10$.

Out-of-vocabulary constants The first possible source of out-of-vocabulary tokens are prefactors. For example, a formula as simple as $u_n = 0.33n$ is hard to predict perfectly, because our decoder only has access to integers between -10 and 10 and a few mathematical constants, and needs to write 0.33 as $(3 * 10 + 3)/(10 * 10)$, i.e. `[div,add,mul,3,10,3,mul,10,10]`. To circumvent this issue, [404] use a separate optimizer to fit the prefactors, once the skeleton of the equation is predicted.

In contrast, our model is end-to-end, and is surprisingly good at approximating out-of-vocabulary prefactors with its own vocabulary. For $u_n = 0.33n$, one could expect the model to predict $u_n = n/3$, which would be a decent approximation. Yet, our model goes much further, and outputs $u_n = -\cos(3)n/3$, which is a better approximation. We give a few other spectacular examples in Table 11.2. Our model is remarkably capable of using the values of operators such as $\exp$ and $\arctan$, as if it were able to perform computations internally.

To investigate the approximation capabilities of our model systematically, we evaluate the its performance when sampling the prefactors uniformly in $[-10, 10]$, rather than in $\{-10, -9, ...9, 10\} \cup$

$\{e, \pi, \gamma\}$ as done usually. It is impossible for the symbolic model to perfectly represent the correct formulas, but since we are interested in its approximation capabilities, we set the tolerance to 0.01. Results are shown in Table 11.8. Unsurprisingly, the performance of the numeric model is unaffected as it does not suffer from any out-of-vocabulary issues, and becomes better than the symbolic model. However, the symbolic model maintains very decent performances, with its approximation accuracy dropping from 60% to 42%.

This approximation ability in itself an impressive feat, as the model was not explicitly trained to achieve it. It can potentially have strong applications for mathematics: for example, if a sequence converges to a numerical value such as 1.64493, it can be useful to ask the model to approximate it, yielding $\pi^2/6$. In fact, one could further improve this approximation ability by training the model only on degree-0 sequences with constant leaves ; we leave this for future work.

Out-of-vocabulary functions A similar difficulty arises when dealing with out-of-vocabulary operators, yet again, our model is able to express or approximate them with its own vocabulary. We show this by evaluating our model on various families of functions from `scipy.special`:

- **polynomials**: base + orthogonal polynomials of degree 1 to 5 (Legendre, Chebyshev, Jacobi, Laguerre, Hermite, Gegenbauer)
- **hyperbolic**: base + {sinh, cosh, tanh, arccosh, arcsinh, arctanh}
- **bessel**: base + {Bessel and modified Bessel of first and second kinds}
- **fresnel**: base + {erf, Faddeeva, Dawson and Fresnel integrals}.

The results in Figure 11.2b show that both the numeric and symbolic models cope surprisingly well with these functions. The symbolic model has more contrasted results than the numeric model, and excels particularly on functions which it can easily build with its own vocabulary such as polynomials. Surprisingly however, it also outperforms the numeric model on the other groups.

In Table 11.3, we show a few examples of the remarkable ability of our model to yield high-quality asymptotic approximations, either using a recurrence relation as for the Catalan numbers and the Dawson function, either with a non-recurrent expression as for the Bessel functions.

11.5 Conclusion

In this chapter, we have shown that Transformer models can successfully infer recurrent mathematical formulas from observations. We applied our model to challenging out-of-distribution tasks, showing that it outperforms Mathematica functions for recurrence prediction on integer sequences and yields very informative approximations of complex functions as well as numerical constants.

Scope of our approach One may ask to what extent our model can be used for real-world applications, such as time-series forecasting. Although robustness to noise is an encouraging step in this direction, we believe our model is not directly adapted to such applications, for two reasons.

First, real-world time-series often cannot be described by a simple mathematical formula, in which case numeric approaches will generally outperform symbolic approaches. Second, even when they can be expressed by a formula, the latter will contain complex prefactors and non-deterministic terms which will make the task extremely challenging.

Future directions As mentioned earlier, recurrence relations can be seen as discretized differential equations. Hence, a natural follow-up to this work would be to move to the continuous setup, and infer the differential equation underlying a set of of trajectories in the spirit of [412]; this will be explored in future work. Along this line, one could model partial differential equations by considering multi-dimensional sequences, and even stochastic differential equations by including non-deterministic terms.

To bring our model closer to real-world problems, one crucially needs to handle non-integer prefactors, either by extending the vocabulary of the decoder, or by using a separate solver to fit them; this is the object of the upcoming chapter, which closes the thesis.

Chapter 12

End-to-end Symbolic Regression with Transformers

In the previous chapter, we explored an innovative use-case for symbolic regression: inferring recurrence relations. However, as explained in the final section, our model is not compatible with real-world data, mainly because it is trained on expressions which feature a very limited set of constants (integers between -10 and 10, e, π, γ).

Dealing with constants in the general framework of symbolic regression is a tricky issue which is usually solved in two steps: predicting the "skeleton" of the expression up to the choice of numerical constants, then fitting the constants by optimizing a non-convex loss function. The dominant approach is genetic programming, which evolves candidates by iterating this subroutine a large number of times. Neural networks have recently been tasked to predict the correct skeleton in a single try, but remain much less powerful.

In this chapter, we challenge this two-step procedure, and task a Transformer to directly predict the full mathematical expression, constants included. One can subsequently refine the predicted constants by feeding them to the non-convex optimizer as an informed initialization. We present ablations to show that this end-to-end approach yields better results, sometimes even without the refinement step. We evaluate our model on problems from the SRBench benchmark and show that our model approaches the performance of state-of-the-art genetic programming with several orders of magnitude faster inference.

12.1 Introduction

Inferring mathematical laws from experimental data is a central problem in natural science; having observed a variable y at n points $\{x_i\}_{i \in \mathbb{N}_n}$, it implies finding a function f such that $y_i \approx f(x_i)$ for all $i \in \mathbb{N}_n$. Two types of approaches exist to solve this problem. In *parametric statistics* (PS), the function f is defined by a small number of parameters that can directly be estimated from the data. On the other hand, *machine learning* (ML) techniques such as decision trees and neural networks select f from large families of non-linear functions by minimizing a loss over the data. The latter relax the assumptions about the underlying law, but their solutions are more difficult to interpret, and tend to overfit small experimental data sets, yielding poor extrapolation performance.

Symbolic regression (SR) stands as a middle ground between PS and ML approaches: f is selected from a large family of functions, but is required to be defined by an interpretable analytical expression.

It has already proved extremely useful in a variety of tasks such as inferring physical laws [413, 414].

SR is usually performed in two steps. First, predicting a "skeleton", a parametric function using a pre-defined list of operators – typically, the basic operations ($+, \times, \div$) and functions ($\mathrm{sqrt}, \exp, \sin$, etc.). It determines the general shape of the law up to a choice of constants, e.g. $f(x) = \cos(ax + b)$. Then, the constants in the skeleton (a, b) are estimated using optimization techniques, typically the Broyden–Fletcher–Goldfarb–Shanno algorithm (BFGS).

The leading algorithms for SR rely on genetic programming (GP). At each generation, a population of candidates is predicted, and the fittest ones are selected based on the data, and mutated to build the next generation. The algorithm iterates this procedure until a satisfactory level of accuracy is achieved.

While GP algorithms achieve good prediction accuracy, they are notably slow (see the Pareto plot of Fig. 12.1). Indeed, the manually predefined function space to search is generally vast, and each generation involves a costly call to the BFGS routine. Also, GP does not leverage past experience: every new problem is learned from scratch. This makes GP techniques inapplicable to situations where fast computation is needed, for instance in reinforcement learning and physics environments [415, 416].

Pre-training neural networks built for language modelling on large datasets of synthetic examples has recently been proposed for SR [404, 405]. These references follow the two-step procedure (predicting the skeleton then fitting the constants) inherited from GP. Once the model is pre-trained, the skeleton is predicted via a simple forward pass, and a single call to BFGS is needed, thus resulting in a significant speed-up compared to GP. However, these methods are not as accurate as state-of-the-art GP, and have so far been limited to low-dimensional functions ($D \leq 3$). We argue that two reasons underlie their shortcomings.

First, skeleton prediction is an ill-posed problem that does not provide sufficient supervision: different instances of the same skeleton can have very different shapes, and instances of very different skeletons can be very close. Second, the loss function minimized by BFGS can be highly non-nonconvex: even when the skeleton is perfectly predicted, the correct constants are not guaranteed to be found. For these reasons, we believe, and will show, that doing away with skeleton estimation as a intermediary step can greatly facilitate the task of SR for language models.

12.1.1 Contributions

In this chapter, we train Transformers over synthetic datasets to perform **end-to-end (E2E)** symbolic regression: solutions are predicted directly, without resorting to skeletons. To this effect, we leverage a hybrid **symbolic-numeric vocabulary**, that uses both symbolic tokens for the operators and variables and numeric tokens for the constants. One can then perform a **refinement** of the predicted constants by feeding them as informed guess to BFGS, mitigating non-linear optimization issues. Finally, we introduce **generation and inference techniques** that allow our models to scale to larger problems: up to 10 input features against 3 in concurrent works.

Evaluated over the SRBench benchmark [417], our model significantly narrows the accuracy gap with state-of-the-art GP techniques, while providing several orders of magnitude of inference time speedup (see Fig. 12.1). We also demonstrate strong robustness to noise and extrapolation capabilities.

Finally, we will provide an online demonstration of our model at `https://symbolicregression.metademolab.com/` and will open-source our implementation as a Scikit-learn compatible regressor at the following address: `https://github.com/facebookresearch/symbolicregression`.

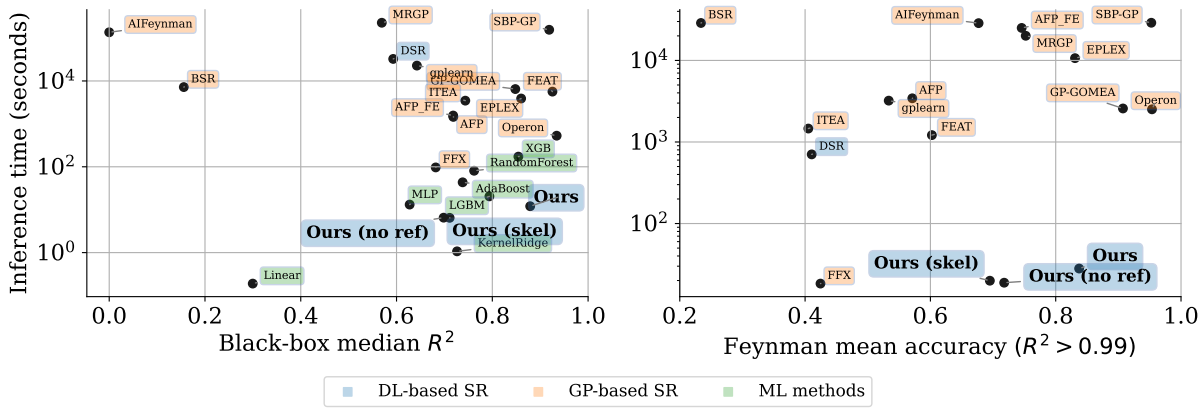


Figure 12.1: **Our model outperforms previous DL-based methods and offers at least an order of magnitude inference speedup compared to SOTA GP-based methods.** Pareto plot comparing the average test performance and inference time of our models with baselines provided by the SRbench benchmark [417], both on Feynman SR problems [413] and black-box regression problems. We use colors to distinguish three families of models: **deep-learning based SR**, **genetic programming-based SR** and **classic machine learning methods** (which do not provide an interpretable solutions). A similar Pareto plot against formula complexity is provided in Fig. K.6.

12.1.2 Related work

SR is a challenging task that traces back from a few decades ago, with a large number of open-source and commercial softwares, and has already been used to accelerate scientific discoveries [418–420]. Most popular frameworks for symbolic regression use GP [398, 421–428] (see [417] for a recent review), but SR has also seen growing interest from the Deep Learning (DL) community, motivated by the fact that neural networks are good at identifying qualitative patterns.

Neural networks have been combined with GP algorithms, e.g. to simplify the original dataset [413], or to propose a good starting distribution over mathematical expressions[403]. [401, 429] propose modifications to feed-forward networks to include interpretable components, i.e. replacing usual activation functions by operators such as $\cos, \sin$, however these are hard to optimize and prone to numerical issues.

Language models, and especially Transformers [78], have been trained over synthetic datasets to solve various mathematical problems: integration [390], dynamical systems [393], linear algebra [396], formal logic [430] and theorem proving [431]. A few papers apply these techniques to symbolic regression: the aforementioned references [404, 405] train Transformers to predict function skeletons, while the previous chapters studies one-dimensional recurrence relations in sequences of numbers.

The recently introduced SRBench [417] provides a benchmark for rigorous evaluation of SR methods, in addition to 14 SR methods and 7 ML baselines which we will compare to in this chapter.

12.2 Data generation

Our approach consists in pre-training language models on vast synthetic datasets. Each training example is a pair: a set of N points $(x, y) \in \mathbb{R}^D \times \mathbb{R}$ as the input, and a function f such that $y = f(x)$ as the

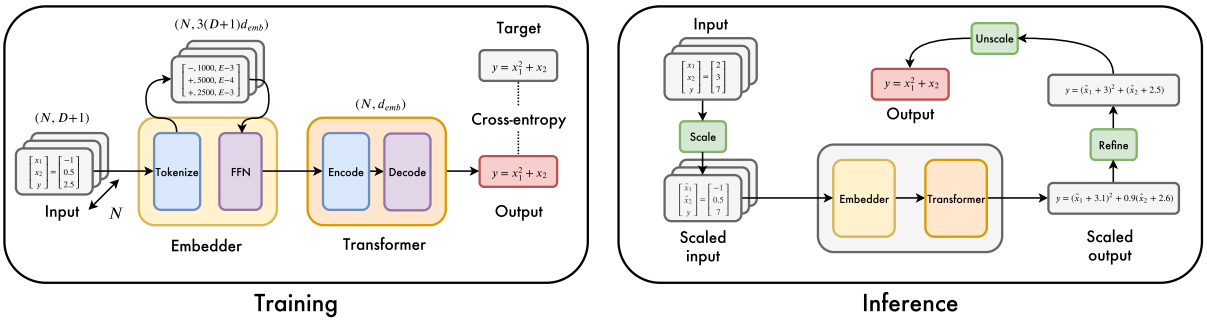


Figure 12.2: **Sketch of our model.** During training, the inputs are all whitened. At inference, we whiten them as a pre-processing step; the predicted function must then be unscaled to account for the whitening.

target[1] Examples are generated by (i) sampling a random function f, (ii) sampling a set of N input values $(x_i)_{i\in\mathbb{N}_N}$ in $\mathbb{R}^D$, and (iii) computing $y_i = f(x_i)$.

12.2.1 Generating functions

To sample functions f, we follow the seminal approach of Lample and Charton [390], and generate random trees with mathematical operators as internal nodes and variables or constants as leaves. The procedure is detailed below (see Table K.1 in the Appendix for the values of parameters):

1. Sample the desired **input dimension** D of the function f from $\mathcal{U}\{1, D_{\text{max}}\}$.
2. Sample the number of **binary operators** b from $\mathcal{U}\{D-1, D+b_{\text{max}}\}$ then sample b operators from $\mathcal{U}\{+,-,\times\}$[2].
3. Build a **binary tree** with those b nodes, using the sampling procedure of [390].
4. For each **leaf** in the tree, sample one of the variables x_d, $d \in \mathbb{N}_D$.
5. Sample the number of **unary operators** u from $\mathcal{U}\{0, u_{\text{max}}\}$ then sample u operators from the list O_u in Table K.1, and insert them at random positions in the tree.
6. For each variable x_d and unary operator u, apply a random **affine transformation**, i.e. replace x_d by $ax_d + b$, and u by $au + b$, with (a, b) sampled from $\mathcal{D}_{\text{aff}}$.

Note that since we require independent control on the number of unary operators (which is independent of D) and binary operators (which depends on D), we cannot directly sample a unary-binary tree as in [390]. Note also that the first D variables are sampled in ascending order to obtain the desired input dimension, which means functions with missing variables such as $x_1 + x_3$ are never encountered; this is not an issue as our model can always set the prefactor of x_2 to zero. As discussed quantitatively in App. K.3, the number of possible skeletons as well as the random sampling of numerical constants guarantees that our model almost never sees the same function twice, and cannot simply perform memorization.

[1] We only consider functions from $\mathbb{R}^D$ into $\mathbb{R}$; the general case $f : \mathbb{R}^D \to \mathbb{R}^P$ can be handled as P independent subproblems.

[2] Note that although the division operation is technically a binary operator, it appears much less frequently than additions and multiplications in typical expressions [432], hence we replace it by the unary operator $\text{inv}: x \to 1/x$.

12.2.2 Generating inputs

For each function $f : \mathbb{R}^D \to \mathbb{R}$, we sample $N \in \mathcal{U}\{10D, N_{\text{max}}\}$ **input values** $x_i \in \mathbb{R}^D$ from the distribution $\mathcal{D}_x$ described below, and compute the corresponding **output values** $y_i = f(x_i)$. If any x_i is outside the domain of definition of f or if any y_i is larger 10^{100}, the process is aborted, and we start again by generating a new function. Note that rejecting and resampling out-of-domain values of x_i, the obvious and cheaper alternative, would provide the model with additional information about f, by allowing it to learn its domain of definition.

To maximize the diversity of input distributions seen at training time, we sample our inputs from a mixture of distributions (uniform or gaussian), centered around k random centroids[3], see App. K.1 for some illustrations at $D = 2$. Input samples are generated as follows:

1. Sample a **number of clusters** $k \sim \mathcal{U}\{1, k_{max}\}$ and k **weights** $w_i \sim \mathcal{U}(0, 1)$, which are then normalized so that $\sum_i w_i = 1$.
2. For each cluster $i \in \mathbb{N}_k$, sample a **centroid** $\mu_i \sim \mathcal{N}(0, 1)^D$, a vector of **variances** $\sigma_i \sim \mathcal{U}(0, 1)^D$ and a **distribution shape** (gaussian or uniform) $\mathcal{D}_i \in \{\mathcal{N}, \mathcal{U}\}$.
3. For each cluster $i \in \mathbb{N}_k$, sample $\lfloor w_i N \rfloor$ **input points** from $\mathcal{D}_i(\mu_i, \sigma_i)$ then apply a **random rotation** sampled from the Haar distribution.
4. Finally, **concatenate** all the points obtained and **whiten** them by substracting the mean and dividing by the standard deviation along each dimension.

12.2.3 Tokenization

Following [396], we represent numbers in base 10 floating-point notation, round them to four significant digits, and encode them as sequences of 3 tokens: their sign, mantissa (between `0` and `9999`), and exponent (from `E-100` to `E100`).

To represent mathematical functions as sequences, we enumerate the trees in prefix order, i.e. direct Polish notation, as in [390]: operators and variables and integers are represented as single autonomous tokens, and constants are encoded as explained above.

For example, the expression $f(x) = \cos(2.4242x)$ is encoded as `[cos,mul,+,2424,E-3,x]`. Note that the vocabulary of the decoder contains a mix of symbolic tokens (operators and variables) and numeric tokens, whereas that of the encoder contains only numeric tokens[4].

12.3 Methods

Below we describe our approach for end-to-end symbolic regression; please refer to Fig. 12.2 for an illustration.

12.3.1 Model

Embedder Our model is provided N input points $(x, y) \in \mathbb{R}^{D+1}$, each of which is represented as $3(D + 1)$ tokens of dimension d_{emb}. As D and N become large, this results in long input sequences (e.g. 6600 tokens for $D = 10$ and $N = 200$), which challenge the quadratic complexity of Transformers. To mitigate this, we introduce an embedder to map each input point to a single embedding.

[3] For $k \to \infty$, such a mixture could in principe approximate any input distribution.

[4] The embeddings of numeric tokens are *not* shared between the encoder and decoder.

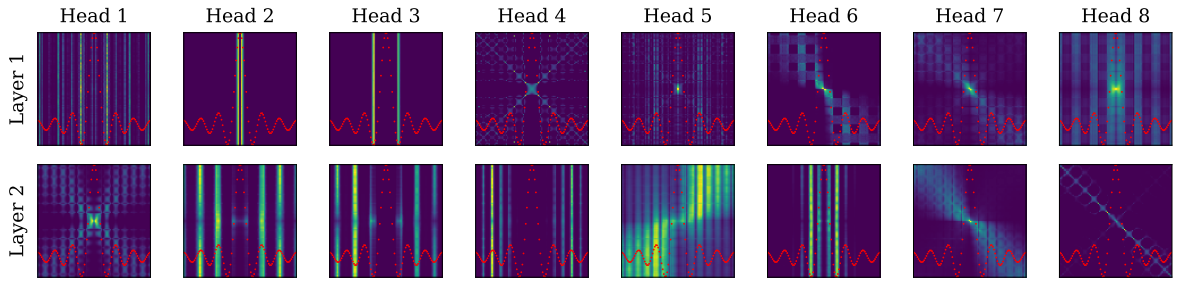


Figure 12.3: **Attention heads reveal intricate mathematical analysis.** We considered the expression $f(x) = \sin(x)/x$, with $N = 100$ input points sampled between -20 and 20 (red dots; the y-axis is arbitrary). We plotted the attention maps of a few heads of the encoder, which are $N \times N$ matrices where the element (i, j) represents the attention between point i and point j. Notice that heads 2, 3 and 4 of the second layer analyze the periodicity of the function in a Fourier-like manner.

The embedder pads the empty input dimensions to D_{max}, then feeds the $3(D_{\text{max}}+1)d_{\text{emb}}$-dimensional vector into a 2-layer fully-connected feedforward network (FFN) with ReLU activations, which projects down to dimension d_{emb}[5] The resulting N embeddings of dimension d_{emb} are then fed to the Transformer.

Transformer We use a sequence to sequence Transformer architecture [78] with 16 attention heads and an embedding dimension of 512, containing a total of 86M parameters. Like [396], we observe that the best architecture for this problem is asymmetric, with a deeper decoder: we use 4 layers in the encoder and 16 in the decoder.

Note an important difference compared with the setup of the previous chapter: here the input is a sequence of pairs $(x_0, y_0), (x_1, y_1), ...$ instead of a sequence of numbers $u_0, u_1, ...$, and these pairs are permutation invariant, which is not the case for number sequences. To account for this invariance, we remove the positional embeddings from the encoder.

As shown in Fig. 12.3 and detailed in App. K.2, the encoder captures the most distinctive features of the functions considered, such as critical points and periodicity, and blends a mix of short-ranged heads focusing on local details with long-ranged heads which capture the global shape of the function.

Training We optimize a cross-entropy loss with the Adam optimizer, warming up the learning rate from 10^{-7} to 2.10^{-4} over the first 10,000 steps, then decaying it as the inverse square root of the number of steps, following [78]. We hold out a validation set of 10^4 examples from the same generator, and train our models until the accuracy on the validation set saturates (around 50 epochs of 3M examples).

Input sequence lengths vary significantly with the number of points N; to avoid wasteful padding, we batch together examples of similar lengths, ensuring that a full batch contains a minimum of 10,000 tokens. On 32 GPU with 32GB memory each, one epoch is processed in about half an hour.

12.3.2 Inference tricks

In this section, we describe three tricks to improve the performance of our model at inference.

[5] We explored various architectures for the embedder, but did not obtain any improvement; this does not appear to be a critical part of the model.

Model	Function $f(x, y)$
Target	$\sin(10x)\exp(0.1y)$
Skeleton + BFGS	$-\sin(1.7x)(0.059y + 0.19)$
E2E no BFGS	$\sin(9.9x)\exp(0.1y)$
E2E + BFGS random init	$-\sin(0.095x)\exp(0.27y)$
E2E + BFGS model init	$\sin(10x)\exp(0.1y)$

Table 12.1: **The importance of an end-to-end model with refinement.** The skeleton approach recovers an incorrect skeleton. The E2E approach predicts the right skeleton. Refinement worsens original prediction when randomly initialized, and yields the correct result when initialized with predicted constants.

Refinement Previous language models for SR, such as [404], follow a *skeleton* approach: they first predict equation skeletons, then fit the constants with a non-linear optimisation solver such as BFGS. In this chapter, we follow an *end-to-end* (E2E) approach: predicting simultaneously the function and the values of the constants. However, we improve our results by adding a *refinement* step: fine-tuning the constants a posteriori with BFGS, initialized with our model predictions[6].

This results in a large improvement over the skeleton approach, as we show by training a Transformer to predict skeletons in the same experimental setting. The improvement comes from two reasons: first, prediction of the full formula provides better supervision, and helps the model predict the skeleton; second, the BFGS routine strongly benefits from the informed initial guess, which helps the model predict the constants. This is illustrated qualitatively in Table 12.1, and quantitatively in Table 12.2.

Scaling As described in Section 12.2.2, all input points presented to the model during training are whitened: their distribution is centered around the origin and has unit variance. To allow accurate prediction for input points with a different mean and variance, we introduce a scaling procedure at inference time. Let f the function to be inferred, x be the input points, and $\mu = \text{mean}(x), \sigma = \text{std}(x)$. As illustrated in Fig. 12.2 we pre-process the input data by replacing x by $\tilde{x} = \frac{x-\mu}{\sigma}$. The model then predicts $\hat{f}(\tilde{x}) = \hat{f}(\sigma x + \mu)$, and we can recover an approximation of f by unscaling the variables in $\hat{f}$.

This gives our model the desirable property to be insensitive to the scale of the input points: DL-based approaches to SR are known to fail when the inputs are outside the range of values seen during training [396]. Note that here, the scale of the inputs translates to the scale of the constants in the function f; although these coefficients are sampled in $\mathcal{D}_{\text{aff}}$ during training, coefficients outside $\mathcal{D}_{\text{aff}}$ can be expressed by multiplication of constants in $\mathcal{D}_{\text{aff}}$.

Bagging and decoding Since our model was trained on $N \leq 200$ input points, it does not perform satisfactorily at inference when presented with more than 200 input points. To take advantage of large datasets while accommodating memory constraints, we perform *bagging*: whenever N is larger than 200 at inference, we randomly split the dataset into B bags of 200 input points[7].

For each bag, we apply a forward pass and generate C function candidates via random sampling or beam search using the next token distribution. As shown in App. K.5 (Fig. K.11), the more commonly

[6] To avoid BFGS having to approximate gradients via finite differences, we provide the analytical expression of the gradient using *sympytorch* [433] and *functorch* [434].

[7] Smarter splits, e.g. diversity-preserving, could be envisioned, but were not considered here.

used beam search [435] strategy leads to much less good results than sampling due to the lack of diversity induced by constant prediction (typical beams will look like $\sin(x), \sin(1.1x), \sin(0.9x), \ldots$). This provides us with a set of BC candidate solutions.

Inference time Our model inference speed has two sources: the forward passes described above on one hand (which can be parallelized up to memory limits of the GPU), and the refinements of candidate functions on the other (which are CPU-based and could also be parallelized, although we did not consider this option here).

Since BC can become large, we rank candidate functions (according to their error on *all* input points), get rid of redundant skeleton functions and keep the best K candidates for the refinement step[8]. To speed up the refinement, we use a subset of at most 1024 input points for the optimization. The parameters B, C and K can be used as cursors in the speed-accuracy tradeoff: in the experiments presented in Fig. 12.1, we selected $B = 100$, $C = 10$, $K = 10$.

12.4 Results

In this section, we present the results of our model. We begin by studying in-domain accuracy, then present results on out-of-domain datasets.

12.4.1 In-domain performance

We report the in-domain performance of our models by evaluating them on a fixed validation set of 100,000 examples, generated as per Section 12.2. Validation functions are uniformly spread out over three difficulty factors: number of unary operators, binary operators, and input dimension. For each function, we evaluate the performance of the model when presented $N = [50, 100, 150, 200]$ input points (x, y), and prediction accuracy is evaluated on $N_{\text{test}} = 200$ points sampled from a fresh instance of the multimodal distribution described in Section 12.2.2.

We assess the performance of our model using two popular metrics: R^2-score [417] and accuracy to tolerance τ [404, 436]:

$$R^2 = 1 - \frac{\sum_i^{N_{\text{test}}} (y_i - \hat{y}_i)^2}{\sum_i^{N_{\text{test}}} (y_i - \bar{y})^2}, \qquad \text{Acc}_\tau = \mathbb{1}\left(\max_{1 \leq i \leq N_{\text{test}}} \left| \frac{\hat{y}_i - y_i}{y_i} \right| \leq \tau \right), \tag{12.1}$$

where $\mathbb{1}$ is the indicator function.

R^2 is classically used in statistics, but it is unbounded, hence a single bad prediction can cause the average R^2 over a set of examples to be extremely bad. To circumvent this, we set $R^2 = 0$ upon pathological examples as in [417](such examples occur in less that 1% of cases)[9]. The accuracy metric provides a better idea of the precision of the predicted expression as it depends on a desired tolerance threshold. However, due to the presence of the $\max$ operator, it is sensitive to outliers, and hence to the number of points considered at test time (more points entails a higher risk of outlier). To circumvent this, we discard the 5% worst predictions, following [404].

[8]Though these candidates are the best functions without refinement, there are no guarantees that these would be the best after refinement, especially as optimization is particularly prone to spurious local optimas.

[9]Note that predicting the constant function $f = \bar{y}$ naturally yields an R^2 score of 0.

Model	R^2	$\text{Acc}_{0.1}$	$\text{Acc}_{0.01}$	$\text{Acc}_{0.001}$
Skeleton + BFGS	0.43	0.40	0.27	0.17
E2E no BFGS	0.62	0.51	0.27	0.09
E2E + BFGS random init	0.44	0.44	0.30	0.19
E2E + BFGS model init	**0.68**	**0.61**	**0.44**	**0.29**

Table 12.2: **Our approach outperforms the skeleton approach.** Metrics are computed over the $10,000$ examples of the evaluation set.

End-to-end outperforms skeleton In Table 12.2, we report the average in-domain results of our models. Without refinement, our E2E model outperforms the skeleton model trained under the same protocol in terms of low precision prediction (R^2 and $\text{Acc}_{0.1}$ metrics), but small errors in the prediction of the constants lead to lower performance at high precision ($\text{Acc}_{0.001}$ metric). The refinement procedure alleviates this issue significantly, inducing a three-fold increase in $\text{Acc}_{0.001}$ while also boosting other metrics.

Initializing BFGS with the constants estimated in the E2E phase plays a crucial role: with random initialization, the BFGS step actually *degrades* E2E performance. However, refinement with random initialization still achieves better results than the skeleton model: this suggests that the E2E model predicts skeletons better that the skeleton model.

Ablation Fig. 12.4A,B,C presents an ablation over three indicators of formula difficulty (from left to right): number of unary operators, number of binary operators and input dimension. In all cases, increasing the factor of difficulty degrades performance, as one could expect. This may give the impression that our model does not scale well with the input dimension, but we show that our model scales in fact very well on out-of-domain datasets compared to concurrent methods (see Fig. K.10 of the Appendix).

Fig. 12.4D shows how performance depends on the number of input points fed to the model, N. In all cases, performance increases, but much more signicantly for the E2E models than for the skeleton model, demonstrating the importance of having a lot of data to accurately predict the constants in the expression.

Extrapolation and robustness In Fig. 12.4E, we examine the ability of our models to interpolate/extrapolate by varying the scale of the test points: instead of normalizing the test points to unit variance, we normalize them to a scale σ. As expected, performance degrades as we increase σ, however the extrapolation performance remains decent even very far away from the inputs ($\sigma = 32$).

Finally, in Fig. 12.4F, we examine the effect of corrupting the targets y with a multiplicative noise of variance σ: $y \to y(1+\xi), \xi \sim \mathcal{N}(0, \varepsilon)$. The results reveal something interesting: without refinement, the E2E model is not robust to noise, and actually performs worse than the skeleton model at high noise. This shows how sensitive the Transformer is to the inputs when predicting constants. Refinement improves robustness significantly, but the initialization of constants to estimated values has less impact, since the prediction of constants is corrupted by the noise.

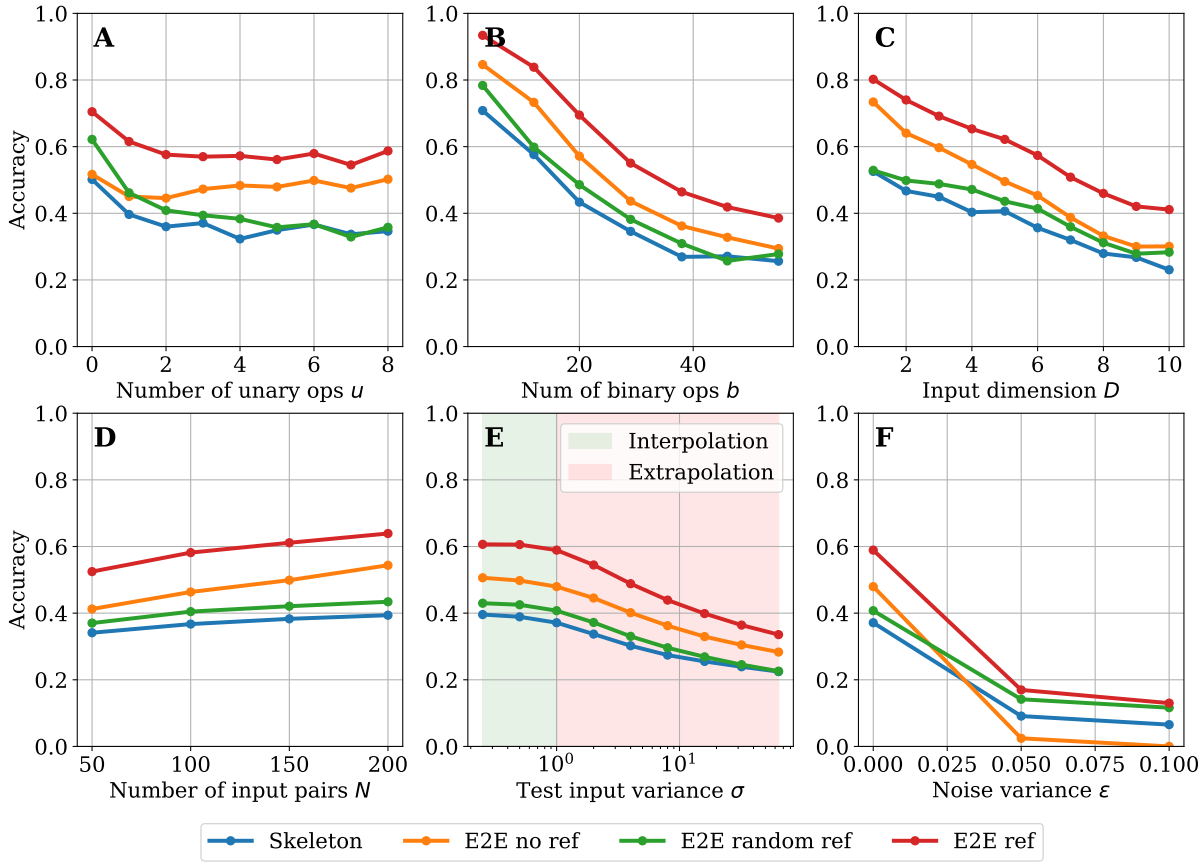


Figure 12.4: **Ablation over the function difficulty (top row) and input difficulty (bottom row).** We plot the accuracy at $\tau = 0.1$ (Eq. 12.1), see App. K.4 for the R^2 score. We distinguish four models: **skeleton**, **E2E without refinement**, **E2E with refinement from random guess** and **E2E with refinement**. **A:** number of unary operators. **B:** number of binary operators. **C:** input dimension. **D:** Low-resource performance, evaluated by varying the number of input points. **E:** Extrapolation performance, evaluated by varying the variance of the inputs. **F:** Robustness to noise, evaluated by varying the multiplicative noise added to the labels.

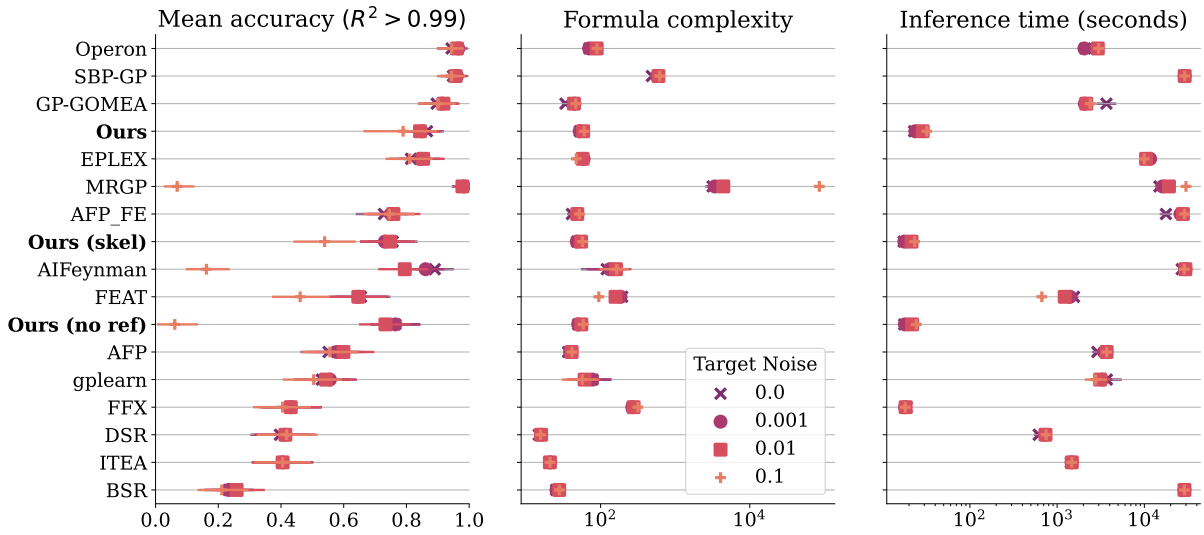


Figure 12.5: **Our model presents strong accuracy-speed-complexity tradeoffs, even in presence of noise.** Results are averaged over all 119 Feynman problems, for 10 random seeds and three target noises each as shown in the legend. The accuracy is computed as the fraction of problems for which the R^2 score on test examples is above 0.99. Models are ranked according to the accuracy averaged over all target noise.

12.4.2 Out-of-domain generalization

We evaluate our method on the recently released benchmark SRBench[417]. Its repository contains a set of 252 regression datasets from the Penn Machine Learning Benchmark (PMLB)[437] in addition to 14 open-source SR and ML baselines. The datasets consist in "ground-truth" problems where the true underlying function is known, as well as "black-box" problems which are more general regression datasets without an underlying ground truth.

We filter out problems from SRBench to only keep regression problems with $D \leq 10$ with continuous features; this results in 190 regression datasets, splitted into 57 black-box problems (combination of real-world and noisy, synthetic datasets), 119 SR datasets from the Feynman [413] and 14 SR datasets from the ODE-Strogatz [438] databases. Each dataset is split into 75% training data and 25% test data, on which performance is evaluated.

The overall performance of our models is illustrated in the Pareto plot of Fig. 12.1, where we see that on both types of problems, our model achieves performance close to state-of-the-art GP models such as Operon with a fraction of the inference time[10]. Impressively, our model outperforms all classic ML methods (e.g. XGBoost and Random Forests) on real-world problems with a lower inference time, and while outputting an interpretable formula.

We provide more detailed results on Feynman problems in Fig. 12.5, where we additionally plot the formula complexity, i.e. the number of nodes in the mathematical tree (see App. K.5 for similar results on black-box and Strogatz problems). Varying the noise applied to the targets noise, we see that our model displays similar robustness to state-of-the-art GP models.

While the average accuracy or our model is only ranked fourth, it outputs formulas with lower

[10]Inference uses a single GPU for the forward pass of the Transformer.

complexity than the top 2 models (Operon and SBP-GP), which is an important criteria for SR problems: see App. K.6 for complexity-accuracy Pareto plots. To the best of our knowledge, our model is the first non-GP approach to achieve such competitive results for SR.

12.5 Conclusion

In this chapter, we introduced a competitive deep learning model for SR by using a novel numeric-symbolic approach. Through rigorous ablations, we showed that predicting the constants in an expression not only improves performance compared to predicting a skeleton, but can also serve as an informed initial condition for a solver to refine the value of the constants.

Our model outperforms previous deep learning approaches by a margin on SR benchmarks, and scales to larger dimensions. Yet, the dimensions considered here remain moderate ($D < 10$): adapting to the truly high-dimensional setup is an interesting future direction, and will likely require qualitative changes in the data generation protocol. While our model narrows the gap between GP and DL based SR, closing the gap also remains a challenge for future work.

This work opens up a whole new range of applications for SR in fields which require real-time inference. We hope that the methods presented here may also serve as a toolbox for many future applications of Transformers for symbolic tasks.

Afterword

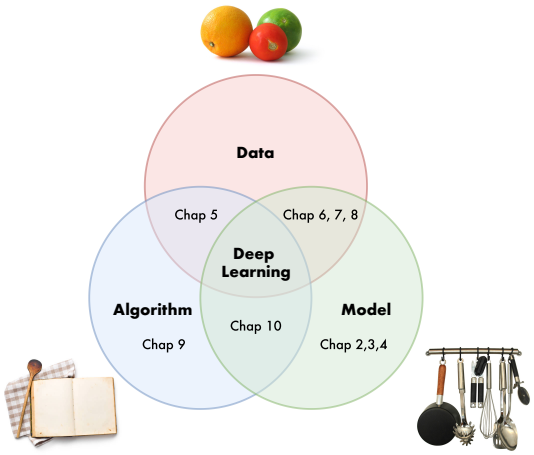


Figure 12.6: The three factors of deep learning.

It is often stated that the success of deep learning stems from three pillars: the **data**, the **model** and the learning **algorithm** (see e.g. [439]). Understanding the interplay between these factors is key to fostering the empirical success of modern neural networks.

In the three first parts of this thesis, we tried to analyze these three components, both in isolation and in conjunction (see Fig. 12.6). By studying over-parametrization in Part I, we were able to elucidate several important questions on the role of the **model**. Part II analyzes the role of architectural bias, which plays in key role in matching the **model** with the **data**. Finally, in Part III, we focused on the dynamics induced by the learning **algorithm** and its interplay with the **model** in the context of Direct Feedback Alignment.

Among the three components, **data** is arguably the most essential, yet the hardest to study from a theoretical point of view. Indeed, while it is rather easy to build toy models of architectures (e.g. the random feature model of Part. I) and dynamics (e.g. the Langevin and DFA algorithms of Part. III), it is less obvious how to build interpretable models of real-world data. Chapter 5 proposes a very simplistic model to study the interplay of data structure with the loss function, but remains extremely far from realistic setups, and only scratches the surface of unanswered questions. Hence, I believe that finding appropriate models of data structure is one of the most critical directions for future work; promising approaches include perceptual manifolds [440], simplexes [441] and latent spaces of generative models [151, 442].

As far as the **algorithm** component is concerned, we have mentioned several future directions in Part III of this thesis, and would like to give a few more. There remains much work to understand the impact of the learning algorithm: the properties of the noise induced by SGD [443, 444], its implicit regularization [445], the effect of learning rate and batch size [295, 446], the benefit of tricks such as momentum and warmup [318, 319], etc. Several theoretical frameworks inspired by statistical mechanics exist: online learning [355, 370], mean-field theory [379, 447] and dynamical mean-field theory [340].

Finally, on the **model** side, two large hurdles need to be overcome. First, the fact that most existing approaches are limited to two-layer networks; deeper models are usually only studied for linear networks [448, 449], and remain a major challenge for future work. The second limitation is the fact that except for a few exceptions [258, 450, 451], most works focus on fully-connected architectures, in spite of their known limitations. One promising route which I did not have time to explore would be

to study mixer-like architectures, which amount to a fully-connected network acting on two separate dimensions of the data and have shown to be able to achieve state-of-the art performance [15, 259, 452].

Bridging together the three pillars of deep learning in a unified theory seems as far from reach as reconciling quantum mechanics and general relativity in the context of physics, yet for opposite reasons. In physics, many candidate theories exist, but we mostly lack the experimental data to confirm one or another. In deep learning, experimental data is ubiquitous, but unified theories are lacking, in great part due to the difficulty of building models which are both analytically tractable and sufficiently complex to encapsulate realistic scenarios. As a result, most recent advances in deep learning have been guided by experience rather than by theory.

However, this gap should not discourage theoretical endeavours, as a few recent works have successfully reversed the scheme. A particularly compelling example is the recent surge of applications of infinite-width results. [98] uses Neural Tangent Kernels [453] to obtain strong results on small regression datasets. [454] leverages infinite-width results to provide a new parametrization of the weights which renders the effect of hyperparameters scale-independent, allowing for example to tune the learning rate of a large model on a much smaller model.

During the last year of my PhD, I transitioned to a more applied topic: symbolic regression, the task of inferring mathematical formulas from numerical data. Our results show that neural networks, which are usually used as numeric regressors devoid of interpretability, are able to provide symbolic outputs extremely accurately, outperforming most existing approaches. This line of research appears to me as particularly promising in the next few years; future directions I would like to explore include (i) inferring differential equations from the observation of trajectories, in the spirit of Chap. 11, (ii) training a model to provide a good initialization point for a non-convex minimization algorithm given the expression of a function, in the spirit of Chap. 12 and finally (iii) applying our symbolic regression methods to various fields of physics – the integer sequence model of Chap. 11 could be useful for counting problems such as those which appear in Feynman diagrams, whereas the end-to-end model of Chap. 12 could be used to infer natural laws in the spirit of [414].

More generally, one of the most exciting prospects of deep learning to me is its potential in speeding up scientific discovery, illustrated by several recent breakthroughs in mathematics [389], biology [455] and physics [456]. I hope to be able to take part in this thrilling adventure as soon as I come back from my long-awaited cycling trip accross the Andes.

Part V

Appendices

Appendix A

From the Jamming Transition to the Double Descent Curve

A.1 Network properties

In the following, we analyze numerically the networks properties that were used in the previous analysis. This provides a numerical confirmation of our arguments, and an in depth characterization of the networks.

A.1.1 Effective number of degrees of freedom

In our discussion the notion of effective number of degrees of freedom is important. In the space of functions going from the neighborhoods of the training set to real numbers, consider the manifold of functions $f(\boldsymbol{x}; \boldsymbol{W})$ obtained by varying $\boldsymbol{W}$. We denote by $P_{\text{eff}}(\boldsymbol{W})$ the dimension of the tangent space of this manifold at $\boldsymbol{W}$. In general we have $P_{\text{eff}}(\boldsymbol{W}) \leq P$. Several reasons can make $P_{\text{eff}}(\boldsymbol{W})$ strictly smaller than P, including:

- Some neurons are never active e.g. for ReLU case: then their associated weights do not contribute to P_{eff}.

  In the extreme case where the signal does not propagate in the network, i.e. $f(\boldsymbol{x}; \boldsymbol{W}) = C_1$ for all $\boldsymbol{x}$ in the neighborhood of the training points $\boldsymbol{x}_\mu$, then $P_{\text{eff}}(\boldsymbol{W}) = 1$. This situation will occur for a poor initialization of the weights, for example if all biases are too negative on the neurons of one layer for ReLU activation function (see for instance [457]). It can also occur if the data $\boldsymbol{x}_\mu$ are chosen in an adversarial manner for a given choice of initial weights. For example, one can choose input patterns so as to not activate the first layer of neurons (which is possible if the number of such neurons is not too large). Poor transmission will be enhanced (and adversarial choices of data will be made simpler) if the architecture presents some bottlenecks. In the situation where $P_{\text{eff}}(\boldsymbol{W}) = 1$, it is very simple to obtain local minima of the loss at finite loss values, even when the model has many parameters.

- The activation function is linear. Then, the output is an affine function of the input, leading to $P_{\text{eff}} \leq d + 1$. Dimension-dependent bounds will also exist if the activation function is polynomial (because the output function then is also restricted to be polynomial).

- Symmetries are present in the network, e.g. the scale symmetry in ReLU networks: since the ReLU function is homogeneous, multiplying the weights of a layer by some factor and dividing the weights in the next layer by the same factor leaves the output function invariant. It will reduce one degrees of freedom per node.

Due to these effects, the function $f(\mathbf{x}; \mathbf{W})$ can effectively depend on less variables that the number of parameters, and thus reduce the dimension of the space spanned by the gradients $\nabla_{\mathbf{W}} f(\mathbf{x}; \mathbf{W})$ that enters in the theory: there are $P - P_{\text{eff}}$ directions in parameter space that do not affect the function. It is tempting to define the effective dimension by considering the dimension of the space spanned by $\nabla_{\mathbf{W}} f(\mathbf{x}_\mu; \mathbf{W})$ as μ varies. This definition is not practical for small number of samples N however, because this dimension would be bounded by N. We can overcome such a problem by considering a neighborhood of each point $\mathbf{x}_\mu$, where the network's function and its gradient can be expanded in the pattern space:

$$f(\mathbf{x}) \approx f(\mathbf{x}_\mu) + (\mathbf{x} - \mathbf{x}_\mu) \cdot \nabla_{\mathbf{x}} f(\mathbf{x}_\mu), \tag{A.1}$$

$$\nabla_{\mathbf{W}} f(\mathbf{x}) \approx \nabla_{\mathbf{W}} f(\mathbf{x}_\mu) + (\mathbf{x} - \mathbf{x}_\mu) \cdot \nabla_{\mathbf{x}} \nabla_{\mathbf{W}} f(\mathbf{x}_\mu). \tag{A.2}$$

Varying the pattern μ and the point $\mathbf{x}$ in the neighborhood of $\mathbf{x}_\mu$, we can build a family M of vectors:

$$M = \left\{ \nabla_{\mathbf{W}} f(\mathbf{x}_\mu) + (\mathbf{x} - \mathbf{x}_\mu) \cdot \nabla_{\mathbf{x}} \nabla_{\mathbf{W}} f(\mathbf{x}_\mu) \right\}_{\mu, \mathbf{x}}. \tag{A.3}$$

We then define the effective dimension P_{eff} as the dimension of M. Because of the linear structure of M, it is sufficient to consider, for each μ, only $d + 1$ values for x, e.g. $x - x_\mu = 0, \hat{\mathbf{e}}_1, \ldots, \hat{\mathbf{e}}_d$, where $\hat{\mathbf{e}}_n$ is the unit vector along the direction n. The effective dimension is therefore

$$P_{\text{eff}} = \text{rk}(G), \tag{A.4}$$

where the elements of the matrix G are defined as

$$G_{i,\alpha} \equiv \partial_{W_i} f(\mathbf{x}_\mu) + \hat{\mathbf{e}}_n \cdot \nabla_{\hat{\mathbf{e}}_n} \partial_{W_i} f(\mathbf{x}_\mu), \tag{A.5}$$

with $\alpha \equiv (\mu, n)$. The index n ranges from 0 to D, and $\hat{\mathbf{e}}_0 \equiv 0$.

In Fig. A.1 we show the effective number of parameters P_{eff} versus the total number of parameters P, in the case of a network with $L = 3$ layers trained on the first 10 PCA components of the MNIST dataset. There is no noticeable difference between the two quantities: the only reduction is due to the symmetries induced by the ReLU functions (there is one such symmetry per neuron. Indeed the ReLU function $\sigma(z) = z\Theta(z)$ satisfies $\Lambda\sigma(z/\Lambda) \equiv \sigma(z)$.) We observed the same results for random data.

A.1.2 $\text{sp}(H_p)$ is symmetric for ReLu activation functions and random data

We consider $\mathcal{H}_p = -\sum_\mu y_\mu \sigma(\Delta_\mu) \hat{\mathcal{H}}_\mu$, where $\hat{\mathcal{H}}_\mu$ is the Hessian of the network function $f(\mathbf{x}_\mu; \mathbf{W})$ and σ is the Relu function. We want to argue that the spectrum of $\mathcal{H}_p$ is symmetric in the limit of large P.

We do two main hypothesis: First, the trace of any finite power of $\mathcal{H}_p$ is self-averaging (concentrates) with respect to the average over the random data:

$$\frac{1}{P} \text{tr}(\hat{\mathcal{H}_p}^n) = \frac{1}{P} \overline{\text{tr}(\hat{\mathcal{H}_p}^n)}. \tag{A.6}$$

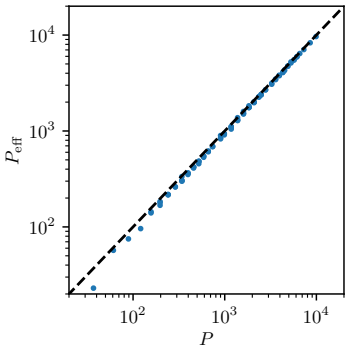


Figure A.1: Results with the MNIST dataset, keeping the first 10 PCA components. $d = 10$ and $L = 3$, varying N and h. Effective P_{eff} vs total number of parameters P. P_{eff} is always smaller than P because there is a symmetry per each ReLU-neuron in the network.

Second,

$$\frac{1}{P}\sum_{\mu_1,\cdots,\mu_n}\overline{y_{\mu_1}\sigma(\Delta_{\mu_1})\cdots y_{\mu_n}\sigma(\Delta_{\mu_n})\mathrm{tr}(\hat{\mathcal{H}}_{\mu_1}\cdots\hat{\mathcal{H}}_{\mu_n})} = \frac{1}{P}\sum_{\mu_1,\cdots,\mu_n}\overline{y_{\mu_1}\sigma(\Delta_{\mu_1})\cdots y_{\mu_n}\sigma(\Delta_{\mu_n})}\cdot\overline{\mathrm{tr}(\hat{\mathcal{H}}_{\mu_1}\cdots\hat{\mathcal{H}}_{\mu_n})} \tag{A.7}$$

The first hypothesis is natural since $\hat{\mathcal{H}}_p$ is a very large random matrix, for which the density of eigenvalues is expected to become a non-fluctuating quantity. The second hypothesis is more tricky: it is natural to assume that the trace concentrates, however one also need to show that the sub-leading corrections to the self-averaging of the trace can be neglected.

Using these two hypothesis and the result, showed below, that

$$\overline{\mathrm{tr}(\hat{\mathcal{H}}_{\mu_1}\cdots\hat{\mathcal{H}}_{\mu_n})} = 0 \tag{A.8}$$

for all n odds, one can conclude that all odds traces of $\hat{\mathcal{H}}_p$ are zero. This implies that the spectrum of $\hat{\mathcal{H}}_p$ is symmetric, more precisely that the fractions of negative and positive eigenvalues are equal.

In order to show that the statement (A.8) above holds, let us argue first that $\overline{\mathrm{tr}(\hat{\mathcal{H}}_\mu^n)} = 0$ for any *odd* n.

$$\mathrm{tr}(\hat{\mathcal{H}}_\mu^n) = \sum_{i_1,i_2,\ldots,i_n}\hat{\mathcal{H}}^\mu_{i_1,i_2}\hat{\mathcal{H}}^\mu_{i_2,i_3}\cdots\hat{\mathcal{H}}^\mu_{i_n,i_1}, \tag{A.9}$$

where the indices $i_1,\ldots,i_n$ stand for synapses connecting a pair of neurons (i.e. each index is associated with a synaptic weight $W^{(j)}_{\alpha,\beta}$: we are not writing all the explicit indexes for the sake of clarity). The term of the hessian obtained when differentiating with respect to weights $W^{(j)}_{\alpha,\beta}$ and $W^{(k)}_{\gamma,\delta}$ reads

$$\hat{\mathcal{H}}^{\mu;(jk)}_{\alpha\beta;\gamma\delta} = \sum_{\pi_0,\ldots,\pi_L}\theta(a^\mu_{L,\pi_L})\cdots\theta(a^\mu_{1,\pi_1})x^\mu_{\pi_0}\cdot \quad \cdot\partial_{W^{(j)}_{\alpha,\beta}}\partial_{W^{(k)}_{\gamma,\delta}}\left[W^{(L+1)}_{\pi_L}W^{(L)}_{\pi_L,\pi_{L-1}}\cdots W^{(1)}_{\pi_1\pi_0}\right]. \tag{A.10}$$

where we denoted with a the inputs in the nodes of the network. Our argument is based on a symmetry of the problem with random data: changing the sign of the weight of the last layer $W^{(L+1)} \longrightarrow -W^{(L+1)}$ and changing the labels $y_\mu \longrightarrow -y_\mu$ leaves the loss unchanged. We will show that this symmetry implies that $\mathrm{tr}(\hat{\mathcal{H}}_\mu^n)$ averaged over the random labels is zero for odd n.

In fact, note that the sum in Equation (A.10) contains a weight per each layer in the network, with the exception of the two layers j, k with respect to which we are deriving. This implies that any element of the hessian matrix where we have not differentiated with respect to the last layer ($j, k < L + 1$) is an odd function of the last layer $W^{(L+1)}$, meaning that if $W^{(L+1)} \longrightarrow -W^{(L+1)}$, then the sign of all these Hessian elements is inverted as well.

If in the argument of the sum in Equation (A.9) there is no index belonging to the last layer, then the whole term changes sign under the transformation $W^{(L+1)} \longrightarrow -W^{(L+1)}$. Suppose now that, on the contrary, there are m terms with one index belonging to the last layer (we need not consider the case of two indices both belonging to the last layer because the corresponding term in the Hessian would be 0, as one can see in Equation (A.10)). For each index equal to $L + 1$ (the last layer), there are exactly two terms: $\hat{\mathcal{H}}^{\mu}_{j,L+1}\hat{\mathcal{H}}^{\mu}_{L+1,k}$ (for some indexes j, k). Since j, k cannot be $L + 1$ too, this implies that the number m of terms with an index belonging to the last layer is always even. Consequently, when the sign of $W^{(L+1)}$ is reversed, the argument of the sum in Equation (A.9) is multiplied by $(-1)^{n-m}$ (once for each term *without* an index belonging to the last layer), which is equal to -1 if n is odd. The same symmetry can be used to show that a matrix made of an odd product of matrices $\hat{\mathcal{H}}_{\mu}$, such as $\hat{\mathcal{H}}_{\mu}\hat{\mathcal{H}}_{\mu'}\hat{\mathcal{H}}_{\mu''}$, must also have a symmetric spectrum, concluding our argument.

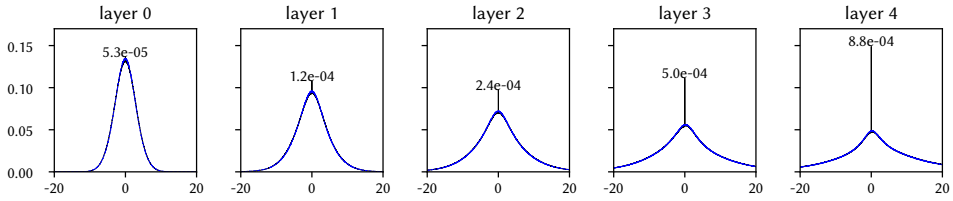


Figure A.2: Density of the pre-activations for each layers with $L = 5$ and random data, averaged over all the runs just above the jamming transition with that architecture. Black: distribution obtained over the training set. Blue: previously unseen random data (the two curves are on top of each other except for the delta in zero). The values indicate the mass of the peak in zero, which is only present when the training set is considered.

A.1.3 Degenerate situations

For adversarially chosen datasets, not all the training data can be fitted. For instance it occurs if two data points are identical but have different labels, $\mathbf{x}_{\mu} = \mathbf{x}_{\nu}$ and $y_{\mu} \neq y_{\nu}$. If the architecture departs from the one used to derive the universal representation theorem, other examples can be built. Consider for instance a ReLU network without biases: such a function defines a homogeneous function $f(|\lambda|\mathbf{x}; \mathbf{W}) \propto |\lambda| f(\mathbf{x}; \mathbf{W})$ and it cannot fit two points that have different labels but are "aligned" as $\mathbf{x}_{\mu} = |\lambda|\mathbf{x}_{\nu}$.

These situations are not generic: a small noise added to the data points is enough to deal with them. When they occur, they affect our conclusions. Consider the case where for a pair of points, $\mathbf{x}_{\mu} = \mathbf{x}_{\nu}$ and $y_{\mu} \neq y_{\nu}$. Then obviously the loss is always positive. In the over-parametrized regime, only these two points contribute to the loss, which is minimised for $f(\mathbf{x}_{\mu}) = 0$. In that symmetric case, $\mathcal{H}_p$ is the sum of two terms that can be shown to cancel out exactly, corresponding to $C_0 = 0$.

A similar situation can take place if propagation does not occur at all (i.e. the output is independent of the input) in a region of the input space in which several data lie. It is presumably possible to build such examples by choosing adversarially the x_i knowing the weights for a ReLU network. However we

have never encountered this case with either real or random data and fully-connected architectures with standard initialization (that let the signal propagate throughout the network).

A.1.4 Density of pre-activations for ReLU activation functions

The densities of pre-activation (i.e. the value of the neurons before applying the activation function) is shown in Fig. A.2 for random data. It contains a delta distribution in zero. The number P_c of pre-activations equal to zero when feeding a network $L = 5$ all its random dataset is $P_c \approx 0.21P$, corresponding to the number of directions in phase space where cusps are present in the loss function. For MNIST data we find $P_c \approx 0.19P$. By taking $L = 2$ and random data we find $P_c \approx 0.25P$. In these directions, stability can be achieved even if the hessian would indicate an instability. For this reason, instead of P_- in Equation (2.12) one should use $P/2 - P_c \approx 0.25P$.

A.2 Parameters used in simulations

A.2.1 Random data

The dataset is composed of N points taken to lie on the D-dimensional hyper-sphere of radius $\sqrt{d}$, $\mathbf{x}_\mu \in \mathcal{S}^d$, with random label $y_\mu = \pm 1$. The networks are fully connected, and have an input layer of size D and L layers with h neurons each, culminating in a final layer of size 1. To find the transition we proceed as follows: we build a network with a number of parameters P large enough for it to be able to fit the whole dataset without errors. Next, we decrease the width h while keeping the depth L fixed, until the network cannot correctly classify all the data anymore within the chosen learning time. We denote this transition point $P^\star$. As initial conditions for the dynamics we use the default initialization of `pytorch`: weights and biases are initialized with a uniform distribution on $[-\sigma, \sigma]$, where $\sigma^2 = 1/f_{in}$ and f_{in} is the number of incoming connections.

When using the cross entropy, the system evolves according to a stochastic gradient descent (SGD) with a learning rate of 10^{-2} for $5 \cdot 10^5$ steps and 10^{-3} for $5 \cdot 10^5$ steps (10^6 steps in total); the batch size is set to $\min(P/2, 1024)$, and batch normalization is used. We do not use any explicit regularization in training the networks. In Fig. A.3 we check that $t = 10^6$ is enough to converge.

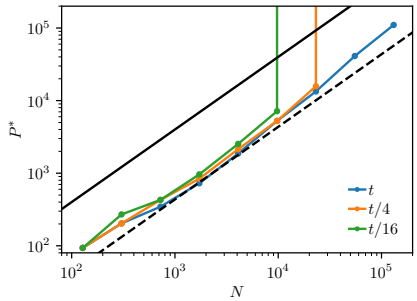


Figure A.3: Convergence of the critical line for networks trained with cross entropy on random data.

When using the hinge loss, we use an orthogonal initialization [458], no batch normalization and $t = 2 \cdot 10^6$ steps of ADAM [459] with batch size N and a learning rate starting at 10^{-4}. In the experiments

of section 2.3.3 (not for the experiments of section 2.4), we progressively divided the learning rate by 10 every 250k steps. Also in this case we do not use any explicit regularization in training the networks.

To observe the discontinuous jump in the number P_Δ of unsatisfied constraints at the transition (Fig. 2.4B and inset), we consider three architectures, both with $P \approx 8000$ and $d = h$ but with different depths $L = 2$, $L = 3$ and $L = 5$. The vicinity of the transition is studied by varying N around the transition value and minimizing for 10^7 steps (a better minimization is needed to improve the precision close to the transition).

Details about Fig 2.4A hinge We took $D = h$ and trained for 2M steps. For some values of $N \in (500, 60\text{k})$, start at large h where we reach $P_\Delta = 0$ and decrease h until $P_\Delta > 0.1P$.

Details about Fig 2.4B We trained networks of depth 2,3,5 with $D = h = 62, 51, 40$ respectively for 10M steps. For $L = 3$ ($D = 51$, $h = 51$) we ran 128 training varying N from 21991 to 25918. For the value of P we take 7854 that correspond to the number of parameters minus the number of neurons, per neuron there is a degree of freedom lost in a symmetry induced by the homogeneity of the ReLU function. 37 of the runs have $P_\Delta = 0$, 74 have $P_\Delta > 0.4P$. Among the 19 remaining ones, 14 of them have P_Δ between 1 and 4, we think that these runs encounter numerical precision issues, we observed that using 32 bit precision accentuate this issue. We think that the 5 left with $4 < P_\Delta < 0.4P$ has been stoped too early. The same observation apply for the other depths.

A.2.2 Real data

The images in the MNIST dataset are gathered into two groups, with even and odd numbers and with labels $y_\mu = \pm 1$. The architecture of the network is as in the previous sections: the D inputs are fed to a cascade of L fully-connected layers with h neurons each, that in the end result in a single scalar output. The loss function used is always the hinge loss.

If we kept the original input size of $28 \times 28 = 784$ (each picture is 28×28 pixels) then the majority of the network's weights would be necessarily concentrated in the first layer (the width h cannot be too large in order to be able to compute the Hessian). To avoid this issue, we opt for a reduction of the input size. We perform a principal component analysis (PCA) on the whole dataset and we identify the 10 dimensions that carry the most variance on the whole dataset; then we use the components of each image along these directions as a new input of dimension $d = 10$. This projection hardly diminishes the performance of the network (which we find to be larger than 90% when using all the data and large P).

Details about Fig 2.4C We trained networks of depth 1,3,5 for 2M steps. For some values of $N \in (100, 50\text{k})$, start at large h where we reach $P_\Delta = 0$ and decrease h until $P_\Delta > 0.1P$.

Details about Fig 2.4D We trained a network of $L = 5$, $d = 10$, $h = 30$ for 3M steps. With N varying from 31k to 68k (using trainset and testset of MNIST).

Details about Fig 2.5 We trained a network of $L = 5$ and $d = 10$ for 500k steps. where $N \in \{10\text{k}, 20\text{k}, 50\text{k}\}$ and h varies from 1 to 3k. Fig A.4 shows a comparison between $L = 5$ and $L = 2$.

Details about Fig 2.6 We trained a network of $L = 5$ and $d = 10$ for $2 \cdot 10^6$ steps. Weights of the network are initialized according to the random orthogonal scheme [458] and all biases are initialized to zero. The network is optimized using ADAM [459] with full batch and the learning rate is set to $\lambda = \min(10^{-1}h^{-1.5}, 10^{-4})$ in order to have a smooth dynamics for all values of h[1].

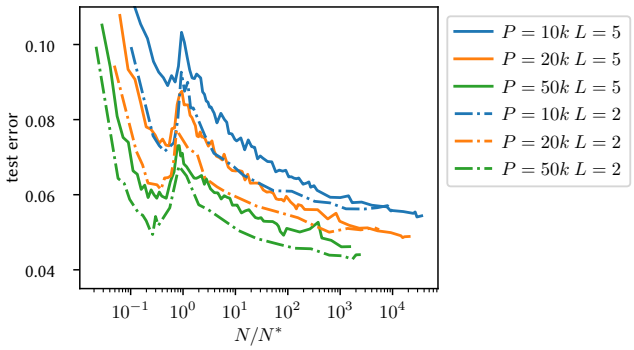


Figure A.4: Generalization on MNIST 10 PCA. Comparison between two depth $L = 2$ and $L = 5$.

A.3 Hessian

The Hessian matrix is a key feature of landscapes, as it characterizes its curvature. It is also a central aspect of the theoretical description above. In this section we systematically analyze the spectra of $\mathcal{H}$, $\mathcal{H}_0$ and $\mathcal{H}_p$.

In Fig. A.5 we show the spectrum of $\mathcal{H}_p$ both for MNIST and random data at varying distance from the jamming transition by varying P. The key observation is that these spectra are symmetric. We also don't observe any accumulation of eigenvalues at $\lambda = 0$, except for the trivial zero modes stemming from the scaling symmetry of ReLU neurons (whose number is the total number of hidden neurons, much smaller than the number of weights).

Fig. A.5B shows the spectrum of $\mathcal{H}_0$ at the end of training for runs close to the jamming transition. As expected it is semi-positive definite, with a delta peak at $\lambda = 0$ corresponding to $P - N_\Delta$ modes. It is followed by a gap and a continuous spectrum, as predicted near the jamming transition of particles if $N_\Delta < P$ [88] (which occurs for elliptic particles [460]). As the loss increases, N_Δ increases and the gap is reduced.

Fig. A.5C shows the spectrum of the full Hessian at the end of training for runs close to the jamming transition. Interestingly the spectrum of the Hessian is not positive definite, but present some unstable modes. This phenomenon stems from our choice of ReLU activation function, which leads to cusps in the landscape as quantified in the Supplementary Material. Such cusps can stabilize directions that would be unstable according to the Hessian.

Overall, as we move from the under-parametrized phase to the over-parametrized one the situation is as follows:

1. N below $P^\star$: There are many constraints with respect to the number of variable N, $\mathcal{H}_0$ is almost

[1]The exponent -1.5 has been empirically chosen so that the number of steps to converge is independent of h [101].

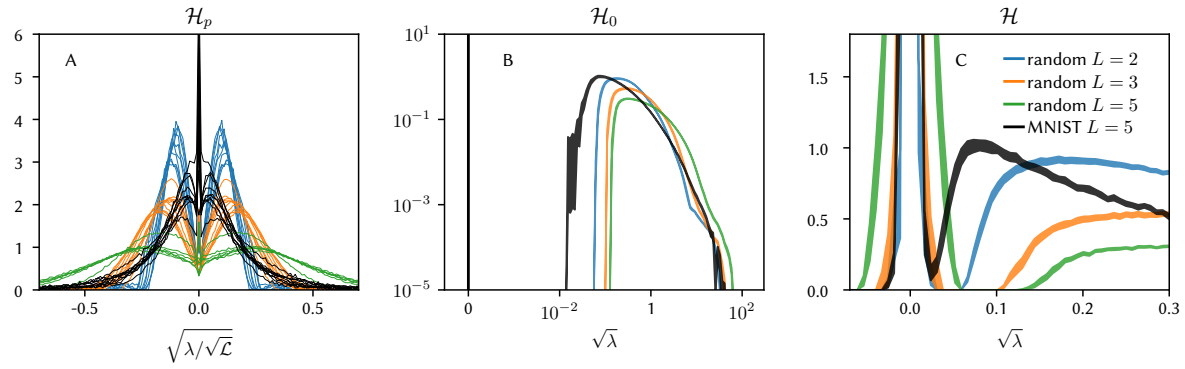


Figure A.5: Spectra of Hessians for runs on random data (colored curves) and MNIST (black curves). The legend is indicated in (D). (A) (Symmetric) spectrum of $\mathcal{H}_p$ at the end of training for 10 runs with different values of P where $\mathcal{L} > 0$. These spectra collapse when plotted in terms of $\lambda/\sqrt{\mathcal{L}}$ (indeed from the definition of $\mathcal{H}_p$ we get that it is proportional to the typical value of the Δ_μ, which scales as $\sqrt{\mathcal{L}}$). Note that they all appear symmetric. (B) Spectrum of $\mathcal{H}_0$ close to jamming. It contains a delta function in zero of weight $P - N_\Delta$, followed by a gap, followed by a continuous spectrum. (C) Spectrum of $\mathcal{H}$ close to jamming. Note that $\mathcal{H}$ has negative eigenvalues, as can occur even in a minimum of the Loss for a ReLU activation fuction, due to the existence of cusps in the landscape. In (B) and (C), we show runs such that $0.7 < N_\Delta/P < 0.8$ and $\mathcal{L} < 10^{-3}$ for random data (colored curves) and $0.54 < N_\Delta/P < 0.7$ and $\mathcal{L} < 1.5 \cdot 10^{-4}$ for MNIST (black curve). The thickness of each line correspond to the standard deviation.

full rank and can easily compensate the negative eigenvalues of $\mathcal{H}_p$. The spectrum of $\mathcal{H}_p$ is symmetric.

2. N approaching $P^\star$ from below: The rank of $\mathcal{H}_0$ decreases but it does not go below $P/2$ since it has to compensate the vanishingly small negative eigenvalues of $\mathcal{H}_p$.

3. As N is large enough, the dynamics finds a global minimum at $\mathcal{L} = 0$ and $\mathcal{H}_p$ vanishes.

Appendix B

Double Trouble in Double Descent: Bias and Variance(s) in the Lazy Regime

B.1 Statement of the Main Result

B.1.1 Assumptions

First, we state precisely the assumptions under which our main result is valid. Note, that these are the same as in [82].

Assumption 1: $\sigma : \mathbb{R} \to \mathbb{R}$ is a weakly differential function with derivative σ'. Assume there exists $c_0, c_1 < \infty \in \mathbb{R}$ such that for all $u \in \mathbb{R}$ $|\sigma(u)|, |\sigma'(u)| \leq c_0 \, \mathrm{e}^{c_1 |u|}$. Then define:

$$\mu_0 = \mathbb{E}\left[\sigma(u)\right], \qquad \mu_1 = \mathbb{E}\left[u\sigma(u)\right], \qquad \mu_\star^2 = \mathbb{E}\left[\sigma^2(u)\right] - \mu_0^2 - \mu_1^2, x \tag{B.1}$$

where the expectation is over $u \sim \mathcal{N}(0, 1)$. To facilitate readability, we specialize to the case $\mu_0 = 0$. This simply amounts to a shift ctivation function $\tilde{\sigma}$ of the network, $\tilde{\sigma}(x) = \sigma(x) - \mu_0$.

Assumption 2: We work in the high-dimensional limit, i.e. in the limit where the input dimension D, the hidden layer dimension P and the number of training points N go to infinity with their ratios fixed. That is:

$$N, P, D \to \infty, \qquad \frac{P}{D} \equiv \psi_1 = \mathcal{O}(1), \quad \frac{N}{D} \equiv \psi_2 = \mathcal{O}(1). \tag{B.2}$$

This condition implies that, in the computation of the risk $\mathcal{R}$, we can neglect all the terms of order $\mathcal{O}(1)$ in favour of the terms of order $\mathcal{O}(D)$.

Assumption 3: The labels are given by a linear ground truth, or teacher function:

$$y_\mu = f_d(\boldsymbol{X}_\mu) + \epsilon_\mu, \quad f_d(\boldsymbol{x}) = \langle \boldsymbol{\beta}, \boldsymbol{x} \rangle, \quad ||\boldsymbol{\beta}|| = F, \quad \epsilon_\mu \sim \mathcal{N}(0, \tau) \, . \tag{B.3}$$

Note that as explained in [82], it is easy to add a non linear component to the teacher, but the latter would not be captured by the model (the student) in the regime $N/D = \mathcal{O}(1)$, and would simply amount to an extra noise term.

B.1.2 Results

Here we give the explicit form of the quantities appearing in our main result. In these expressions, the index $a \in \{v, e, d\}$ distinguishes the *vanilla*, *ensembling* and *bagging* terms.

$$
\begin{aligned}
\Psi_1 &= \frac{1}{D}\operatorname{Tr}\left[H\left[S^v\right]^{-1} H\left[P_{\Psi_1}\right]\right], \\
\Psi_2^v &= \frac{1}{D}\operatorname{Tr}\left[H\left[S^v\right]^{-1} H\left[P_{\Psi_2^v}\right]\right], \\
\Psi_3^v &= \frac{1}{D}\operatorname{Tr}\left[H\left[S^v\right]^{-1} H\left[P_{\Psi_3^v}\right]\right], \\
\Psi_2^e &= \frac{1}{D}\operatorname{Tr}\left[H\left[S^e\right]^{-1} H\left[P_{\Psi_2^e}\right]\right], \\
\Psi_3^e &= \frac{1}{D}\operatorname{Tr}\left[H\left[S^e\right]^{-1} H\left[P_{\Psi_3^e}\right]\right], \\
\Psi_2^b &= \frac{1}{D}\operatorname{Tr}\left[H\left[S^b\right]^{-1} H\left[P_{\Psi_2^b}\right]\right],
\end{aligned}
$$

where the Hessian matrix $H[F]$, for a given function $F : (q, r, \tilde{q}, \tilde{r}) \mapsto \mathbb{R}$ is defined as:

$$
H\left[F\right] = \left[\begin{array}{cccc}
\frac{\partial F}{\partial q \partial q} & \frac{\partial F}{\partial q \partial r} & \frac{\partial F}{\partial q \partial \tilde{q}} & \frac{\partial F}{\partial q \partial \tilde{r}} \\
\frac{\partial F}{\partial q \partial r} & \frac{\partial F}{\partial r \partial r} & \frac{\partial F}{\partial r \partial \tilde{q}} & \frac{\partial F}{\partial r \partial \tilde{r}} \\
\frac{\partial F}{\partial q \partial \tilde{q}} & \frac{\partial F}{\partial r \partial \tilde{q}} & \frac{\partial F}{\partial \tilde{q} \partial \tilde{q}} & \frac{\partial F}{\partial \tilde{q} \partial \tilde{r}} \\
\frac{\partial F}{\partial q \partial \tilde{r}} & \frac{\partial F}{\partial r \partial \tilde{r}} & \frac{\partial F}{\partial \tilde{q} \partial \tilde{r}} & \frac{\partial F}{\partial \tilde{r} \partial \tilde{r}}
\end{array}\right]\Bigg|_{\substack{q=q^* \\ r=r^* \\ \tilde{r}=0 \\ \tilde{q}=0}},
$$

with $q^\star$ and $r^\star$ being the solutions of the fixed point equation for the function $S_0 : (q, r) \mapsto \mathbb{R}$ defined below:

$$
\begin{cases}
\frac{\partial S_0(q,r)}{\partial q} = 0 \\
\frac{\partial S_0(q,r)}{\partial r} = 0.
\end{cases}
$$

$$
S_0(q, r) = \lambda \psi_1^2 \psi_2 q + \psi_2 \log\left(\frac{\mu_\star^2 \psi_1 q}{\mu_1^2 \psi_1 r + 1} + 1\right) + \frac{r}{q} + (1 - \psi_1)\log(q) + \psi_2 \log\left(\mu_1^2 \psi_1 r + 1\right) - \log(r).
$$

The explicit expression of the above quantities in terms of $(q, r, \tilde{q}, \tilde{r})$ is given below.

B.1.3 Explicit expression of the actions

Here we present the explicit formulas for S^v, S^e, S^b, which are defined as the functions $(q, r, \tilde{q}, \tilde{r}) \mapsto \mathbb{R}$ such that:

$$
\begin{aligned}
S^v(q, r, \tilde{q}, \tilde{r}) &= 2\left(S_0(q, r) + \tilde{q} f^v(q, r) + \tilde{r} g^v(q, r)\right) \\
S^e(q, r, \tilde{q}, \tilde{r}) &= S_0(q, r) + \tilde{r}^2 f^e(q, r) + \tilde{q}^2 g^e(q, r) \\
S^b(q, r, \tilde{q}, \tilde{r}) &= S_0(q, r) + \tilde{r}^2 f^b(q, r) + \tilde{q}^2 g^b(q, r),
\end{aligned}
$$

where we defined the functions $(q, r) \mapsto \mathbb{R}$,

$$
\begin{aligned}
f^v(q,r) &= \lambda\psi_1^2\psi_2 + \frac{\mu_\star^2\psi_1\psi_2}{\mu_\star^2\psi_1 q + \mu_1^2\psi_1 r + 1} + \frac{1-\psi_1}{q} - \frac{r}{q^2},\\
g^v(q,r) &= -\frac{\mu_\star^2\mu_1^2\psi_1^2\psi_2 q}{\left(\mu_1^2\psi_1 r + 1\right)\left(\mu_\star^2\psi_1 q + \mu_1^2\psi_1 r + 1\right)} + \frac{\mu_1^2\psi_1\psi_2}{\mu_1^2\psi_1 r + 1} + \frac{1}{q} - \frac{1}{r},\\
f^e(q,r) &= \frac{2r\mu_1^2\psi_1\left(1 + q\mu_\star^2\psi_1\right) + \left(1 + q\mu_\star^2\psi_1\right)^2 - r^2\mu_1^4\psi_1^2(-1+\psi_2)}{r^2\left(1 + r\mu_1^2\psi_1 + q\mu_\star^2\psi_1\right)^2},\\
g^e(q,r) &= \frac{\psi_1}{q^2},\\
f^b(q,r) &= \frac{1}{r^2},\\
g^b(q,r) &= \frac{\psi_1}{q^2}.
\end{aligned}
$$

B.1.4 Explicit expression of the auxiliary terms

Here we present the explicit formulas for the auxiliary terms $P_{\Psi_1}, P^v_{\Psi_2}, P^v_{\Psi_3}, P^e_{\Psi_2}, P^e_{\Psi_3}, P^b_{\Psi_2}$, which are defined as the functions $(q, r, \tilde{q}, \tilde{r}) \mapsto \mathbb{R}$ such that:

$$
\begin{aligned}
P_{\Psi_1} &= \psi_1\psi_2\mu_1^2\left(M_X^{11} + \mu_1^2\mu_\star^2\psi_1^2\left(M_X M_W^v M_X\right)^{11} + \mu_\star^2\psi_1\left(M_X M_W^v\right)^{11}\right),\\
P_{\Psi_2^v} &= D\psi_1^2\psi_2(\mu_1^2\tilde{r} + \mu_\star^2\tilde{q})\left[\mu_1^2 P^v_{XX} - 2\mu_1^2\mu_\star^2\psi_1 P^v_{WX} + \mu_\star^2 P^v_{WW}\right],\\
P_{\Psi_3^v} &= D\psi_1^2\psi_2(\mu_1^2\tilde{r} + \mu_\star^2\tilde{q})\left[\mu_1^2\left(M_X^{12} + \mu_1^2\mu_\star^2\psi_1^2\left[M_X M_W^v M_X\right]^{12}\right) - 2\mu_1^2\mu_\star^2\psi_1\left[M_X M_W^v\right]^{12} + \mu_\star^2\left[M_W^v\right]^{12}\right],\\
P_{\Psi_2^e} &= D\psi_1^2\psi_2\mu_1^2\tilde{r}\left[\mu_1^2 P^e_{XX} - 2\mu_1^2\mu_\star^2\psi_1 P^e_{WX} + \mu_\star^2 P^e_{WW}\right],\\
P_{\Psi_3^e} &= D\psi_1^2\psi_2\mu_1^2\tilde{r}\left[\mu_1^2\left(M_X^{12} + \mu_1^2\mu_\star^2\psi_1^2\left[M_X M_W^e M_X\right]^{12}\right) - 2\mu_1^2\mu_\star^2\psi_1\left[M_X M_W^e\right]^{12} + \mu_\star^2[M_W^e]^{12}\right],\\
P_{\Psi_2^b} &= D\mu_1^2\psi_1\psi_2^2\tilde{r}\left[\psi_1\mu_1^2 P_{XX} + 2\mu_\star^2\mu_1^2\psi_1^2 P_{WX} + \mu_\star^2\psi_1 P_{WW}\right],
\end{aligned}
$$

where we defined the scalars P_{XX}, P_{WX}, P_{WW} as follows:

$$
\begin{aligned}
P^v_{XX} &= \psi_2 N_X^{12} + M_X^{12} + 2\psi_2(\mu_1\mu_\star\psi_1)^2 \left[M_X N_X M_W^a\right]^{12} + (\mu_1\mu_\star\psi_1)^2 \left[M_X M_W^a M_X\right]^{12} \\
&\quad + \psi_2(\mu_1\mu_\star\psi_1)^4 \left[M_X M_W N_X M_W^a M_X\right]^{12}, \\
P^v_{WX} &= \psi_2 \left[N_X M_W^a\right]^{12} + \left[M_X M_W^a\right]^{12} + \psi_2(\mu_1\mu_\star\psi_1)^2 \left[M_X M_W^a N_X M_W^a\right]^{12}, \\
P^v_{WW} &= \left[M_W^a\right]^{12} + \psi_2(\mu_1\mu_\star\psi_1)^2 \left[M_W^a N_X M_W^a\right]^{12}, \\
P^e_{XX} &= P^v_{XX}, \\
P^e_{WX} &= P^v_{WX}, \\
P^e_{WW} &= P^v_{WW}, \\
P^b_{XX} &= \left(N_X^{d11} + 2(\mu_1\mu_\star\psi_1)^2 \left[N_X^b M_W^b M_X^b\right]^{11} + (\mu_1\mu_\star\psi_1)^4 \left[M_X^b M_W^b N_X^b M_W^b M_X^b\right]^{11}\right), \\
P^b_{WX} &= \left[N_X^b M_W^b\right]^{11} + (\mu_1\mu_\star\psi_1)^2 \left[M_X^b M_W^b N_X^b M_W^b\right]^{11}, \\
P^b_{WW} &= (\mu_1\mu_\star\psi_1)^2 \left[M_W^b N_X^b M_W^b\right]^{11},
\end{aligned}
$$

and the 2×2 matrices M_X, M_W, N_X as follows:

$$
M_X^v = \begin{bmatrix} \frac{r}{1+\mu_1^2\psi_1 r} & \frac{\tilde{r}}{\left(1+\mu_1^2\psi_1 r\right)^2} \\ \frac{\tilde{r}}{\left(1+\mu_1^2\psi_1 r\right)^2} & \frac{r}{1+\mu_1^2\psi_1 r} \end{bmatrix}, \; M_W^v = \begin{bmatrix} \frac{q\left(1+\mu_1^2\psi_1 r\right)}{1+2\mu_1^2\psi_1 r+\mu_\star^2\psi_1 q} & \frac{q^2\mu_1^2\mu_\star^2\psi_1^2\tilde{r}}{\left(1+\mu_1^2\psi_1 r+\mu_\star^2\psi_1 q\right)^2} \\ \frac{q^2\mu_1^2\mu_\star^2\psi_1^2\tilde{r}}{\left(1+\mu_1^2\psi_1 r+\mu_\star^2\psi_1 q\right)^2} & \frac{q\left(1+\mu_1^2\psi_1 r\right)}{1+2\mu_1^2\psi_1 r+\mu_\star^2\psi_1 q} \end{bmatrix}, \; N_X^v = \frac{1}{\left(1+\mu_1^2\psi_1 r\right)^2} \begin{bmatrix} r & \tilde{r} \\ \tilde{r} & r \end{bmatrix},
$$

$$
M_X^e = M_X^v, \quad M_W^e = M_W^v + \frac{(1+r\mu_1^2\psi_1)^2\tilde{q}}{\left(1+\mu_1^2\psi_1 r+\mu_\star^2\psi_1 q\right)^2} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \quad N_X^e = \frac{1}{\left(1+\mu_1^2\psi_1 r\right)^2} \begin{bmatrix} r & \tilde{r} \\ \tilde{r} & r \end{bmatrix},
$$

$$
M_X^b = \frac{r}{1+\mu_1^2\psi_1 r^2} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad M_W^b = \frac{q(1+\mu_1^2\psi_1 r)}{1+\mu_1^2\psi_1 r+\mu_\star^2\psi_1 q} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad N_X^b = \frac{1}{(1+\mu_1^2\psi_1 r)^2} \begin{bmatrix} \tilde{r} & 0 \\ 0 & \tilde{r} \end{bmatrix}.
$$

B.2 Replica Computation

B.2.1 Toolkit

Gaussian integrals

In order to obtain the main result for the generalisation error, we perform the averages over all the sources of randomness in the system in the following order: over the dataset X, then over the noise W, and finally over the random feature layers Θ. Here are some useful formulaes used throughout the computations:

$$
\begin{cases}
\int e^{-\frac{1}{2}x_i G_{ij} x_j + J_i x_i} dx & = (\det G)^{-\frac{1}{2}} e^{\frac{1}{2} J_i G^{-1}_{ij} J_j}, \\
\int x_a e^{-\frac{1}{2}x_i G_{ij} x_j + J_i x_i} dx & = P^1_a (\det G)^{-\frac{1}{2}} e^{\frac{1}{2} J_i G^{-1}_{ij} J_j}, \\
\int x_a x_b e^{-\frac{1}{2}x_i G_{ij} x_j + J_i x_i} dx & = P^2_{ab} (\det G)^{-\frac{1}{2}} e^{\frac{1}{2} J_i G^{-1}_{ij} J_j}, \\
\int x_a x_b x_c e^{-\frac{1}{2}x_i G_{ij} x_j + J_i x_i} dx & = P^3_{abc} (\det G)^{-\frac{1}{2}} e^{\frac{1}{2} J_i G^{-1}_{ij} J_j}, \\
\int x_a x_b x_c x_d e^{-\frac{1}{2}x_i G_{ij} x_j + J_i x_i} dx & = P^4_{abcd} (\det G)^{-\frac{1}{2}} e^{\frac{1}{2} J_i G^{-1}_{ij} J_j},
\end{cases}
\tag{B.4}
$$

with

$$
\begin{aligned}
P^1_a &= [G^{-1}J]_a, \\
P^2_{ab} &= ((G^{-1})_{ab} + [G^{-1}J]_a[G^{-1}J]_b), \\
P^3_{abc} &= \sum_{\overline{a},\overline{b},\overline{c}\in perm(abc)} \left((G^{-1})_{\overline{a}\overline{b}}[G^{-1}J]_{\overline{c}} + [G^{-1}J]_{\overline{a}}[G^{-1}J]_{\overline{b}}[G^{-1}J]_{\overline{c}}\right), \\
P^4_{abcd} &= \sum_{\overline{a},\overline{b},\overline{c},\overline{d}\in perm(abcd)} \left((G^{-1})_{\overline{a}\overline{b}}(G^{-1})_{\overline{c}\overline{d}} + [G^{-1}J]_{\overline{a}}[G^{-1}J]_{\overline{b}}[G^{-1}J]_{\overline{c}}[G^{-1}J]_{\overline{d}} + (G^{-1})_{\overline{a}\overline{b}}[G^{-1}J]_{\overline{c}}[G^{-1}J]_{\overline{d}})\right).
\end{aligned}
$$

Replica representation of an inverse matrix

To obtain gaussian integrals we will use the "replica" representation the element (ij) of a matrix M of size D:

$$
M^{-1}_{ij} = \lim_{n\to 0} \int \left(\prod_{\alpha=1}^{n}\prod_{i=1}^{D} d\eta_i^\alpha\right) \eta_i^1\eta_j^1 \exp\left(-\frac{1}{2}\eta_i^\alpha M_{ij}\eta_j^\alpha\right). \tag{B.5}
$$

Indeed, using the gaussian integral representation of the inverse of M,

$$
\begin{aligned}
M^{-1}_{ij} &= \mathcal{Z}^{-1}\int \left(\prod_{i=1}^{D} d\eta_i\right)\eta_i\eta_j \exp\left(-\frac{1}{2}\eta_i M_{ij}\eta_j\right), \\
\mathcal{Z} &= \sqrt{\frac{(2\pi)^D}{\det M}} \\
&= \int \left(\prod_{i=1}^{D} d\eta_i\right) \exp\left(-\frac{1}{2}\eta_i M_{ij}\eta_j\right).
\end{aligned}
$$

Using the replica identity, we rewrite this as

$$
M^{-1}_{ij} = \lim_{n\to 0} \mathcal{Z}^{n-1}\int \left(\prod_{i=1}^{D} d\eta_i\right)\eta_i\eta_j \exp\left(-\frac{1}{2}\eta_i M_{ij}\eta_j\right).
$$

Renaming the integration variable of the integral on the left as η^1 and the $n-1$ others as $\eta^\alpha, \alpha \in \{2, n\}$, we obtain expression (B.5).

B.2.2 The Random Feature model

In what follows, we will explicitly leave the indices of all the quantities used. We use the notation, called Einstein summation convention in physics, in which all repeated indices are summed but the sum is not explicitly written. Indices $i \in \{1...D\}$ are used to refer to the input dimension, $h \in \{1...P\}$ to refer to the hidden layer dimension and $\mu \in \{1...N\}$ to refer to the number of data points.

With a single learner

In the random features model, the predictor can be computed explicitly:

$$\hat{a} = \frac{1}{\sqrt{D}} \boldsymbol{y}_\mu^\top \boldsymbol{Z}_{\mu h} \left(\boldsymbol{Z}^\top \boldsymbol{Z} + \psi_1 \psi_2 \lambda \mathbf{I}_N \right)^{-1}_{hh'} \tag{B.6}$$

$$f(\boldsymbol{x}) = \hat{a}_h \sigma \left(\frac{\boldsymbol{\Theta}_{h'i} \boldsymbol{x}_i}{\sqrt{D}} \right) \tag{B.7}$$

$$= \boldsymbol{y}_\mu^\top \boldsymbol{Z}_{\mu h} \left(\boldsymbol{Z}^\top \boldsymbol{Z} + \psi_1 \psi_2 \lambda \mathbf{I}_N \right)^{-1}_{hh'} \sigma \left(\frac{\boldsymbol{\Theta}_{h'i} \boldsymbol{x}_i}{\sqrt{D}} \right) / \sqrt{D}, \tag{B.8}$$

where

$$\boldsymbol{y}_\mu = f_d \left(\boldsymbol{X}_\mu \right) + \boldsymbol{\epsilon}_\mu, \tag{B.9}$$

$$\boldsymbol{Z}_{\mu h} = \frac{1}{\sqrt{D}} \sigma \left(\frac{1}{\sqrt{D}} \boldsymbol{\Theta}_{hi} \boldsymbol{X}_{\mu i} \right). \tag{B.10}$$

Hence the test error can be computed as:

$$\epsilon_g = \mathbb{E}_{\boldsymbol{x}} \left[\left(f_d(\boldsymbol{x}) - \boldsymbol{y}_\mu^\top \boldsymbol{Z}_{\mu h} \left(\boldsymbol{Z}^\top \boldsymbol{Z} + \psi_1 \psi_2 \lambda \mathbf{I}_N \right)^{-1}_{hh'} \sigma \left(\frac{\boldsymbol{\Theta}_{h'i} \boldsymbol{x}_i}{\sqrt{D}} \right) / \sqrt{D} \right)^2 \right] \tag{B.11}$$

$$= \mathbb{E}_{\boldsymbol{x}} \left[f_d(\boldsymbol{x})^2 \right] - 2 \boldsymbol{y}_\mu^\top \boldsymbol{Z}_{\mu h} \left(\boldsymbol{Z}^\top \boldsymbol{Z} + \psi_1 \psi_2 \lambda \mathbf{I}_N \right)^{-1}_{hh'} \boldsymbol{V}_{h'} / \sqrt{D}$$
$$+ \boldsymbol{y}_\mu^\top \boldsymbol{Z}_{\mu h} \left(\boldsymbol{Z}^\top \boldsymbol{Z} + \psi_1 \psi_2 \lambda \mathbf{I}_N \right)^{-1}_{hh_1} \boldsymbol{U}_{h_1 h_2} \left(\boldsymbol{Z}^\top \boldsymbol{Z} + \psi_1 \psi_2 \lambda \mathbf{I}_N \right)^{-1}_{h_2 h'} \boldsymbol{Z}^\top_{h'\mu'} \boldsymbol{y}_{\mu'} / D, \tag{B.12}$$

where

$$\boldsymbol{V}_h = \mathbb{E}_{\boldsymbol{x}} \left[f_d(\boldsymbol{x}) \sigma \left(\frac{\langle \boldsymbol{\Theta}_{hi} \boldsymbol{x}_i \rangle}{\sqrt{D}} \right) \right], \tag{B.13}$$

$$\boldsymbol{U}_{hh'} = \mathbb{E}_{\boldsymbol{x}} \left[\sigma \left(\frac{\langle \boldsymbol{\Theta}_{hi} \boldsymbol{x}_i \rangle}{\sqrt{D}} \right) \sigma \left(\frac{\boldsymbol{\Theta}_{h'i}, \boldsymbol{x}_i}{\sqrt{D}} \right) \right]. \tag{B.14}$$

Ensembling over K learners

When ensembling over K learners with independently sampled random feature vectors, the predictor becomes:

$$f(\boldsymbol{x}) = \frac{1}{K\sqrt{D}} \sum_k \boldsymbol{y}_\mu^\top \boldsymbol{Z}^{(k)}_{\mu h} \left(\boldsymbol{Z}^{\top(k)} \boldsymbol{Z}^{(k)} + \psi_1 \psi_2 \lambda \mathbf{I}_N \right)^{-1}_{hh'} \sigma \left(\frac{\boldsymbol{\Theta}^{(k)}_{h'i} \boldsymbol{x}_i}{\sqrt{D}} \right), \tag{B.15}$$

where

$$\boldsymbol{Z}^{(k)}_{\mu h} = \frac{1}{\sqrt{D}} \sigma \left(\frac{1}{\sqrt{D}} \boldsymbol{\Theta}^{(k)}_{hi} \boldsymbol{X}_{\mu i} \right). \tag{B.16}$$

The generalisation error is then given by:

$$\epsilon_g = \mathbb{E}_{\boldsymbol{x}}\left[\left(f_d(\boldsymbol{x}) - \frac{1}{K}\sum_k \boldsymbol{y}^\top \boldsymbol{Z}^{(k)}\left(\boldsymbol{Z}^{\top(k)}\boldsymbol{Z}^{(k)} + \psi_1\psi_2\lambda \mathbf{I}_N\right)^{-1}\sigma\left(\frac{\boldsymbol{\Theta}^{(k)}_{h'i}\boldsymbol{x}_i}{\sqrt{D}}\right)/\sqrt{D}\right)^2\right] \tag{B.17}$$

$$= \mathbb{E}_{\boldsymbol{x}}\left[f_d(\boldsymbol{x})^2\right] - \frac{2}{K}\sum_k \boldsymbol{y}^\top \boldsymbol{Z}^{(k)}\left(\boldsymbol{Z}^{\top(k)}\boldsymbol{Z}^{(k)} + \psi_1\psi_2\lambda \mathbf{I}_N\right)^{-1}\boldsymbol{V}^{(k)}/\sqrt{D}$$
$$+ \frac{1}{K^2}\sum_{\substack{k\\ l\neq k}} \boldsymbol{y}^\top \boldsymbol{Z}^{(k)}\left(\boldsymbol{Z}^{\top(k)}\boldsymbol{Z}^{(k)} + \psi_1\psi_2\lambda \mathbf{I}_N\right)^{-1}\boldsymbol{U}^{(kl)}\left(\boldsymbol{Z}^{\top(l)}\boldsymbol{Z}^{(l)} + \psi_1\psi_2\lambda \mathbf{I}_N\right)^{-1}\boldsymbol{Z}^{(l)\top}\boldsymbol{y}/D, \tag{B.18}$$

where

$$\boldsymbol{V}^{(k)}_h = \mathbb{E}_{\boldsymbol{x}}\left[f_d(\boldsymbol{x})\sigma\left(\frac{\langle \boldsymbol{\Theta}^{(k)}_{hi}\boldsymbol{x}_i\rangle}{\sqrt{D}}\right)\right], \tag{B.19}$$

$$\boldsymbol{U}^{(kl)}_{hh'} = \mathbb{E}_{\boldsymbol{x}}\left[\sigma\left(\frac{\langle \boldsymbol{\Theta}^{(k)}_{hi}\boldsymbol{x}_i\rangle}{\sqrt{D}}\right)\sigma\left(\frac{\langle \boldsymbol{\Theta}^{(l)}_{h'i'}\boldsymbol{x}_{i'}\rangle}{\sqrt{D}}\right)\right]. \tag{B.20}$$

Equivalent Gaussian Covariate Model

It was shown in [82] that the random features model is equivalent, in the high-dimensional limit of Assumption 2, to a Gaussian covariate model in which the activation function σ is replaced as:

$$\sigma\left(\frac{\boldsymbol{\Theta}^{(k)}_{hi}\boldsymbol{X}_{\mu i}}{\sqrt{D}}\right) \to \mu_0 + \mu_1\frac{\boldsymbol{\Theta}^{(k)}_{hi}\boldsymbol{X}_{\mu i}}{\sqrt{D}} + \mu_\star \boldsymbol{W}^{(k)}_{\mu h}, \tag{B.21}$$

with $\boldsymbol{W}^{(k)} \in \mathbb{R}^{N\times P}$, $W^{(k)}_{\mu h} \sim \mathcal{N}(0,1)$ and μ_0, μ_1 and $\mu_\star$ defined in (B.1). To simplify the calculations, we take $\mu_0 = 0$, which amounts to adding a constant term to the activation function σ.
This powerful mapping allows to express the quantities $\boldsymbol{U}, \boldsymbol{V}$. We will not repeat their calculations here: the only difference here is $\boldsymbol{U}^{kl}$, which carries extra indices k, l due to the different initialization of the random features $\boldsymbol{\Theta}^{(k)}$. In our case,

$$\boldsymbol{U}^{(kl)}_{hh'} = \frac{\mu_1^2}{D}\boldsymbol{\Theta}^{(k)}_{hi}\boldsymbol{\Theta}^{(l)}_{h'i} + \mu_\star^2\delta_{kl}\delta_{hh'}. \tag{B.22}$$

Hence we can rewrite the test error as

$$\mathbb{E}_{\{\boldsymbol{\Theta}^{(k)}\},\boldsymbol{X},\varepsilon}\left[\epsilon_g\right] = F^2\left(1 - 2\Psi_1^v\right) + \frac{1}{K}\left(F^2\Psi_2^v + \tau^2\Psi_3^v\right) + \left(1 - \frac{1}{K}\right)\left(F^2\Psi_2^e + \tau^2\Psi_3^e\right), \tag{B.23}$$

where $\Psi_1, \Psi_2^v, \Psi_2^e, \Psi_3^v, \Psi_3^e$ are given by:

$$\Psi_1 = \frac{1}{D}\,\mathrm{Tr}\left[\left(\frac{\mu_1}{D}\boldsymbol{X}\boldsymbol{\Theta}^{(1)\top}\right)^\top \boldsymbol{Z}^{(1)}\left(\boldsymbol{Z}^{(1)\top}\boldsymbol{Z}^{(1)} + \psi_1\psi_2\lambda\mathbf{I}_N\right)^{-1}\right]$$

$$\Psi_2^v = \frac{1}{D}\,\mathrm{Tr}\left[\left(\boldsymbol{Z}^{(1)\top}\boldsymbol{Z}^{(1)} + \psi_1\psi_2\lambda\mathbf{I}_N\right)^{-1}\left(\frac{\mu_1^2}{D}\boldsymbol{\Theta}^{(1)}\boldsymbol{\Theta}^{(1)\top} + \mu_*^2\mathbf{I}_N\right)\left(\boldsymbol{Z}^{(1)\top}\boldsymbol{Z}^{(1)} + \psi_1\psi_2\lambda\mathbf{I}_N\right)^{-1}\boldsymbol{Z}^{(1)\top}\left(\frac{1}{D}\boldsymbol{X}\boldsymbol{X}^\top\right)\boldsymbol{Z}^{(1)}\right]$$

$$\Psi_3^v = \frac{1}{D}\,\mathrm{Tr}\left[\left(\boldsymbol{Z}^{(1)\top}\boldsymbol{Z}^{(1)} + \psi_1\psi_2\lambda\mathbf{I}_N\right)^{-1}\left(\frac{\mu_1^2}{D}\boldsymbol{\Theta}^{(1)}\boldsymbol{\Theta}^{(1)\top} + \mu_\star^2\mathbf{I}_N\right)\left(\boldsymbol{Z}^{(1)\top}\boldsymbol{Z}^{(1)} + \psi_1\psi_2\lambda\mathbf{I}_N\right)^{-1}\boldsymbol{Z}^{(1)\top}\boldsymbol{Z}^{(1)}\right]$$

$$\Psi_2^e = \frac{1}{D}\,\mathrm{Tr}\left[\left(\boldsymbol{Z}^{(1)\top}\boldsymbol{Z}^{(1)} + \psi_1\psi_2\lambda\mathbf{I}_N\right)^{-1}\left(\frac{\mu_1^2}{D}\boldsymbol{\Theta}^{(1)}\boldsymbol{\Theta}^{(2)\top}\right)\left(\boldsymbol{Z}^{(2)\top}\boldsymbol{Z}^{(2)} + \psi_1\psi_2\lambda\mathbf{I}_N\right)^{-1}\boldsymbol{Z}^{(2)\top}\left(\frac{1}{D}\boldsymbol{X}\boldsymbol{X}^\top\right)\boldsymbol{Z}^{(1)}\right]$$

$$\Psi_3^e = \frac{1}{D}\,\mathrm{Tr}\left[\left(\boldsymbol{Z}^{(1)\top}\boldsymbol{Z}^{(1)} + \psi_1\psi_2\lambda\mathbf{I}_N\right)^{-1}\left(\frac{\mu_1^2}{D}\boldsymbol{\Theta}^{(1)}\boldsymbol{\Theta}^{(2)\top}\right)\left(\boldsymbol{Z}^{(2)\top}\boldsymbol{Z}^{(2)} + \psi_1\psi_2\lambda\mathbf{I}_N\right)^{-1}\boldsymbol{Z}^{(2)\top}\boldsymbol{Z}^{(1)}\right].$$

B.2.3 Computation of the *vanilla* terms

To start with, let us compute the vanilla terms (those who carry a superscript v), which involve a single instance of the random feature vectors. Note that these were calculated in [82] by evaluating the Stieljes transform of the random matrices of which we need to calculate the trace. The replica method used here makes the calculation of the vanilla terms carry over easily to the the ensembling terms (superscript e) and the bagging term (superscript b). To illustrate the calculation steps, we will calculate Ψ_3^v, then provide the results for Ψ_2^v and Ψ_1.

In the vanilla terms, the two inverse matrices that appear are the same. Hence we use twice the replica identity (B.5), introducing $2n$ replicas which all play the same role:

$$M_{ij}^{-1}M_{kl}^{-1} = \lim_{n\to 0}\int\left(\prod_{\alpha=1}^{2n} d\eta\right)\eta_i^1\eta_j^1\eta_k^2\eta_l^2\exp\left(-\frac{1}{2}\eta^\alpha M_{ij}\eta^\alpha\right). \tag{B.24}$$

The first step is to perform the averages, i.e. the Gaussian integrals, over the dataset X, the deterministic noise W induced by the non-linearity of the activation function and the random features Θ.

Averaging over the dataset

Replacing the activation function by its Gaussian covariate equivalent model and using (B.24), the term Ψ_3 can be expanded as:

$$\begin{aligned}
\Psi_3^v &= \frac{1}{D}\left[\boldsymbol{Z}_{\mu h}\left(\boldsymbol{Z}^\top\boldsymbol{Z} + \psi_1\psi_2\lambda\mathbf{I}_N\right)^{-1}_{hh_1}\left(\frac{\mu_1^2}{D}\boldsymbol{\Theta}_{h_1 i}\boldsymbol{\Theta}_{h_2 i} + \mu_\star^2\delta_{h_1h_2}\right)\left(\boldsymbol{Z}^\top\boldsymbol{Z} + \psi_1\psi_2\lambda\mathbf{I}_N\right)^{-1}_{h_2h'}\boldsymbol{Z}_{h'\mu}\right] \\
&= \frac{1}{D}\left(\frac{\mu_1^2}{D}\boldsymbol{\Theta}_{h_1 i}\boldsymbol{\Theta}_{h_2 i} + \mu_\star^2\delta_{h_1h_2}\right)\left[\boldsymbol{Z}_{\mu h}\boldsymbol{Z}_{h'\mu}\right]\int\left(\prod_{\alpha=1}^{2n} d\eta^\alpha\right)\eta_h^1\eta_{h_1}^1\eta_{h_2}^2\eta_{h'}^2\exp\left(-\frac{1}{2}\eta_h^\alpha\left(\boldsymbol{Z}^\top\boldsymbol{Z} + \psi_1\psi_2\lambda\mathbf{I}_N\right)_{hh'}\eta_{h'}^\alpha\right) \\
&= \frac{1}{D^2}\left(\frac{\mu_1^2}{D}\boldsymbol{\Theta}_{h_1 i}\boldsymbol{\Theta}_{h_2 i} + \mu_\star^2\delta_{h_1h_2}\right)\left(\frac{\mu_1}{\sqrt{D}}\boldsymbol{\Theta}\boldsymbol{X} + \mu_\star\boldsymbol{W}\right)_{h\mu}\left(\frac{\mu_1}{\sqrt{D}}\boldsymbol{\Theta}\boldsymbol{X} + \mu_\star\boldsymbol{W}\right)_{h'\mu} \\
&\int\left(\prod_{\alpha=1}^{2n} d\eta^\alpha\right)\eta_h^1\eta_{h_1}^1\eta_{h_2}^2\eta_{h'}^2\exp\left(-\frac{1}{2}\eta_h^\alpha\left(\frac{1}{D}\left(\frac{\mu_1}{\sqrt{D}}\boldsymbol{\Theta}\boldsymbol{X} + \mu_\star\boldsymbol{W}\right)_{h\mu}\left(\frac{\mu_1}{\sqrt{D}}\boldsymbol{\Theta}\boldsymbol{X} + \mu_\star\boldsymbol{W}\right)_{h'\mu} + \psi_1\psi_2\lambda\delta_{hh'}\right)\eta_{h'}^\alpha\right).
\end{aligned}$$

Now, we introduce $\lambda_i^\alpha := \frac{1}{\sqrt{P}}\eta_h^\alpha\boldsymbol{\Theta}_{hi}$, and enforce this relation using the Fourier representation of the delta-function:

$$1 = \int d\lambda_i^\alpha d\hat{\lambda}_i^\alpha e^{i\hat{\lambda}_i^\alpha(\sqrt{P}\lambda_i^\alpha - \eta_h^\alpha\boldsymbol{\Theta}_{hi})}. \tag{B.25}$$

The average over the dataset $\boldsymbol{X}_{\mu i}$ has the form of (B.4) with:

$$(G_X)_{\mu\mu',ii'} = \delta_{\mu\mu'}\left(\delta_{ii'} + \frac{\mu_1^2\psi_1}{D}\lambda_i^\alpha\lambda_{i'}^\alpha\right), \tag{B.26}$$

$$(J_X)_{\mu,i} = \frac{\mu_1\mu_\star\sqrt{\psi_1}}{D}\sum_{\alpha a}\lambda_i^\alpha\eta_h^\alpha\boldsymbol{W}_{\mu h}. \tag{B.27}$$

Using formulae (B.4), we obtain:

$$\begin{aligned}\Psi_3^v =& \frac{N}{D^2}\int\left(\prod_{\alpha=1}^{2n}d\eta^\alpha d\lambda^\alpha d\hat{\lambda}^\alpha\right)\left(\frac{\mu_1^2 P}{D}\lambda_i^1\lambda_i^2 + \mu_\star^2\eta_i^1\eta_i^2\right)\\ &\left[\mu_\star^2\eta_h^1\boldsymbol{W}_h\boldsymbol{W}_{h'}\eta_{h'}^2 + \mu_1^2\psi_1\lambda_i^1\lambda_{i'}^2\left((G_X^{-1})_{ii'} + (G_X^{-1}J_X)_i(G_X^{-1}J_X)_{i'}\right) + 2\mu_1\mu_\star\sqrt{\psi_1}\lambda_i^1\boldsymbol{W}_h\eta_h^2(G_X^{-1}J_X)_i\right]\\ &\exp\left(-\frac{n}{2}\log\det(G_X) - \frac{1}{2}\eta_h^\alpha\left(\frac{\mu_\star^2}{D}\boldsymbol{W}_h\boldsymbol{W}_{h'}\delta_{\alpha\beta} - \frac{\mu_1^2\mu_\star^2\psi_1}{D^2}\boldsymbol{W}_h\lambda_i^\alpha(G_X)_{ii'}^{-1}\lambda_{i'}^\beta\boldsymbol{W}_{h'} + \psi_1\psi_2\lambda\delta_{hh'}\right)\eta_{h'}^\beta + i\hat{\lambda}_i^\alpha(\sqrt{P}\lambda_i^\alpha - \eta_h^\alpha\boldsymbol{\Theta}_{hi})\right)\end{aligned}$$

Note that due to with a slight abuse of notation we got rid of indices μ, which all sum up trivially to give a global factor N.

Averaging over the deterministic noise

The expectation over the deterministic noise $\boldsymbol{W}_h$ is a Gausssian integral of the form (B.4) with:

$$[G_W]_{hh'} = \delta_{hh'} + \frac{\mu_\star^2}{D}\eta_h^\alpha A^{\alpha\beta}\eta_{h'}^\beta, \tag{B.28}$$

$$[J_W]_h = 0, \tag{B.29}$$

$$A^{\alpha\beta} = \delta_{\alpha\beta} - \mu_1^2\psi_1\frac{1}{D}\sum_{i,j}\lambda_i^\alpha[G_X^{-1}]_{ij}\lambda_j^\beta. \tag{B.30}$$

Note that the prefactor involves, constant, linear and quadratic terms in $\boldsymbol{W}$ since:

$$(G_X^{-1}J_X)_i = \frac{\mu_1\mu_\star\sqrt{\psi_1}}{D}\left[\eta^\alpha\boldsymbol{W}\right]\left[G_X^{-1}\lambda^\alpha\right]_i.$$

Thus, one obtains:

$$\begin{aligned}\Psi_3^v = \frac{\psi_2}{D}\int\left(\prod_{\alpha=1}^{2n}d\eta^\alpha d\lambda^\alpha d\hat{\lambda}^\alpha\right)&\left(\mu_1^2\psi_1\lambda_i^1\lambda_i^2 + \mu_\star^2\eta_i^1\eta_i^2\right)\left[\mu_\star^2\left[\eta^1(G_W^{-1})\eta^2\right] + \mu_1^2\psi_1\left[\lambda^1 H_W\lambda^2\right] + 2\mu_1^2\mu_\star^2\psi_1[\lambda^1 S_W\eta^2]\right]\\ &\exp\left(-\frac{n}{2}\log\det(G_X) - \frac{n}{2}\log\det(G_W) - \frac{1}{2}\psi_1\psi_2\lambda\sum(\eta_h^\alpha)^2 + i\hat{\lambda}_i^\alpha(\sqrt{P}\lambda_i^\alpha - \eta_h^\alpha\boldsymbol{\Theta}_{hi})\right),\end{aligned}$$

with

$$(H_W)_{ij} = (G_X^{-1})_{ij} + \frac{\mu_1^2\mu_\star^2\psi_1}{D^2}\left[\eta^\alpha(G_W^{-1})\eta^\beta\right]\left[G_X^{-1}\lambda^\alpha\right]_i\left[G_X^{-1}\lambda^\beta\right]_j, \tag{B.31}$$

$$(S_W)_{ih} = \frac{1}{D}\left[G_X^{-1}\lambda^\alpha\right]_i\left[G_W^{-1}\eta^\alpha\right]_h. \tag{B.32}$$

Averaging over the random feature vectors

The expectation over the random feature vectors $\mathbf{\Theta}_{hi}$ is a Gausssian integral of the form (B.4) with:

$$[G_\Theta]_{hh',ii'} = \delta_{hh',ii'}, \tag{B.33}$$

$$[J_\Theta]_{hi} = -i\hat{\lambda}_i^\alpha \eta_h^\alpha. \tag{B.34}$$

Preforming this integration results in:

$$\Psi_3^v = \frac{\psi_2}{D}\int\left(\prod_{\alpha=1}^{2n} d\eta^\alpha d\lambda^\alpha d\hat{\lambda}^\alpha\right)\left(\mu_1^2\psi_1\lambda_i^1\lambda_i^2 + \mu_\star^2\eta_i^1\eta_i^2\right)\left[\mu_\star^2\left[\eta^1(G_W^{-1})\eta^2\right] + \mu_1^2\psi_1\left[\lambda^1 H_W\lambda^2\right] + 2\mu_1^2\mu_\star^2\psi_1[\lambda^1 S_W\eta^2]\right]$$
$$\exp\left(-\frac{n}{2}\log\det(G_X) - \frac{n}{2}\log\det(G_W) - \frac{1}{2}\psi_1\psi_2\lambda\sum(\eta_h^\alpha)^2 - \frac{1}{2}\eta_h^\alpha\eta_h^\beta\hat{\lambda}_i^\alpha\hat{\lambda}_i^\beta + i\sqrt{P}\hat{\lambda}_i^\alpha\lambda_i^\alpha\right).$$

Expression of the action and the prefactor

To complete the computation we integrate with respect to $\hat{\lambda}_i^\alpha$, using again formulae (B.4):

$$\left[G_{\hat{\lambda}}\right]_{ii'}^{\alpha\beta} = \delta^{ii'}\eta_h^\alpha\eta_h^\beta, \tag{B.35}$$

$$\left[J_{\hat{\lambda}}\right]_i^\alpha = i\sqrt{P}\lambda_i^\alpha. \tag{B.36}$$

This yields the final expression of the term:

$$\Psi_3^v = \frac{\psi_2}{D}\int\left(\prod_{\alpha=1}^{2n} d\eta^\alpha\right)\left(\prod_{\alpha=1}^{2n} d\lambda^\alpha\right)\left(\mu_1^2\psi_1\lambda_i^1\lambda_i^2 + \mu_\star^2\eta_i^1\eta_i^2\right)\left[\mu_\star^2\left[\eta^1(G_W^{-1})\eta^2\right] + \mu_1^2\psi_1\left[\lambda^1 H_W\lambda^2\right] + 2\mu_1^2\mu_\star^2\psi_1[\lambda^1 S_W\eta^2]\right]$$
$$\exp\left(-\frac{n}{2}\log\det(G_X) - \frac{n}{2}\log\det(G_W) - \frac{D}{2}\log\det(G_{\hat{\lambda}}) - \frac{1}{2}\psi_1\psi_2\lambda\sum(\eta_h^\alpha)^2 - \frac{P}{2}\lambda_i^\alpha(G_{\hat{\lambda}}^{-1})_{ii'}^{\alpha\beta}\lambda_{i'}^\beta\right).$$

The above may be written as

$$\Psi_3^v = \int\left(\prod d\eta\right)\left(\prod d\lambda\right)P_{\Psi_3^v}[\eta,\lambda]\exp\left(-\frac{D}{2}S^v[\eta,\lambda]\right), \tag{B.37}$$

with the *prefactor* $P_{\Psi_3^v}$ and the *action* S^v defined as:

$$P_{\Psi_3^v}[\eta,\lambda] := \frac{\psi_2}{D}\left(\mu_1^2\psi_1\lambda_i^1\lambda_i^2 + \mu_\star^2\eta_i^1\eta_i^2\right)\left[\mu_\star^2\left[\eta^1(G_W^{-1})\eta^2\right] + \mu_1^2\psi_1\left[\lambda^1 H_W\lambda^2\right] + 2\mu_1^2\mu_\star^2\psi_1[\lambda^1 S_W\eta^2]\right],$$
$$S^v[\eta,\lambda] := \psi_2\log\det(G_X) + \psi_2\log\det(G_W) + \log\det(G_{\hat{\lambda}}) + \frac{1}{D}\psi_1\psi_2\lambda\sum(\eta_h^\alpha)^2 + \frac{P}{D}\left(\lambda_i^\alpha(G_{\hat{\lambda}}^{-1})_{ii'}^{\alpha\beta}\lambda_{i'}^\beta\right).$$

Expression of the action and the prefactor in terms of order parameters

Here we see that we have a factor $D\to\infty$ in the exponential part, which can be estimated using the saddle point method. Before doing so, we introduce the following order parameters using the Fourier representation of the delta-function:

$$1 = \int dQ_{\alpha\beta}d\hat{Q}_{\alpha\beta}e^{\hat{Q}_{\alpha\beta}(PQ_{\alpha\beta} - \eta_h^\alpha\eta_h^\beta)}, \tag{B.38}$$

$$1 = \int dR_{\alpha\beta}d\hat{R}_{\alpha\beta}e^{\hat{R}_{\alpha\beta}(DR_{\alpha\beta} - \lambda_i^\alpha\lambda_i^\beta)}. \tag{B.39}$$

This allows to rewrite the prefactor only in terms of Q, R: for example,

$$\mu_1^2\psi_1\lambda_i^1\lambda_i^2 + \mu_\star^2\eta_i^1\eta_i^2 = \psi_1 D(\mu_1^2 R^{12} + \mu_\star^2 Q^{12}).$$

To do this, there are two key quantities we need to calculate: $\lambda G_X^{-1}\lambda$ and $\eta G_W^{-1}\eta$. To calculate both, we note that G_X ang G_W are both of the form $\mathbf{I} + \mathbf{X}$, therefore there inverse may be calculated using their series representation. The result is:

$$[M_X]^{\alpha\beta} := \frac{1}{D}\lambda^\alpha G_X^{-1}\lambda^\beta = \left[R(I + \mu_1^2\psi_1 R)^{-1}\right]^{\alpha\beta}, \tag{B.40}$$

$$[M_W]^{\alpha\beta} := \frac{1}{P}\eta^\alpha (G_W^{-1})\eta^\beta = \left[Q(I + \mu_\star^2\psi_1 AQ)^{-1}\right]^{\alpha\beta}. \tag{B.41}$$

Using the above, we deduce:

$$\lambda^1 H_W \lambda^2 = DM_X^{12} + P\mu_1^2\mu_\star^2\psi_1 \left[M_X M_W M_X\right]^{12}, \tag{B.42}$$

$$\lambda^1 S_W \eta^2 = P\left[M_X M_W\right]^{12}. \tag{B.43}$$

The integrals over η, λ become simple Gaussian integrals with covariance matrices given by $\hat{Q}, \hat{R}$, yielding:

$$1 = \int dQ_{\alpha\beta} d\hat{Q}_{\alpha\beta} e^{-\frac{\psi_1 D}{2}\left(\log\det\hat{Q} - 2\mathrm{Tr}Q\hat{Q}\right)}, \tag{B.44}$$

$$1 = \int dR_{\alpha\beta} d\hat{R}_{\alpha\beta} e^{-\frac{D}{2}\left(\log\det\hat{R} - 2\mathrm{Tr}R\hat{R}\right)}. \tag{B.45}$$

The next step is to take the saddle point with respect to the auxiliary variables $\hat{Q}$ and $\hat{R}$ in order to eliminate them:

$$\frac{\partial S^v}{\partial \hat{Q}_{\alpha\beta}} = \psi_1\left(\hat{Q}^{-1} - 2Q\right) = 0 \Rightarrow \hat{Q} = \frac{1}{2}Q^{-1}, \tag{B.46}$$

$$\frac{\partial S^v}{\partial \hat{R}_{\alpha\beta}} = \left(\hat{R}^{-1} - 2R\right) = 0 \Rightarrow \hat{R} = \frac{1}{2}R^{-1}. \tag{B.47}$$

One finally obtains that:

$$\Psi_3^v = \int \left(\prod dQ\right)\left(\prod dR\right) P_{\Psi_3^v}[Q,R] \exp\left(-\frac{D}{2}S^v[Q,R]\right), \tag{B.48}$$

With:

$$\begin{aligned} P_{\Psi_3^v}[Q,R] =& D\psi_1^2\psi_2(\mu_1^2 R^{12} + \mu_\star^2 Q^{12})\left[\mu_\star^2\left[M_W^v\right]^{12} + \mu_1^2\left(M_X^{12} + \mu_1^2\mu_\star^2\psi_1^2\left[M_X M_W^v M_X\right]^{12}\right) + 2\mu_1^2\mu_\star^2\psi_1\left[M_X M_W^v\right]^{12}\right], \\ S^v[Q,R] =& \psi_2\log\det(G_X) + \psi_2\log\det(G_W) + \psi_1^2\psi_2\lambda\mathrm{Tr}Q + \mathrm{Tr}\left(RQ^{-1}\right) + (1-\psi_1)\log\det Q - \log\det R. \end{aligned} \tag{B.49}$$

Saddle point equations

The aim is now to use the saddle point method in order to evaluate the integrals over the order parameters. Thus, one looks for R and Q solutions to the equations:

$$\frac{\partial S^v}{\partial Q_{\alpha\beta}} = 0, \quad \frac{\partial S^v}{\partial R_{\alpha\beta}} = 0 \ \ \forall\alpha,\beta = 1,\cdots,2n.$$

To solve the above, it is common to make a *replica symmetric ansatz*. In this case, we assume that the solutions to the saddle points equations take the form:

$$Q = \begin{bmatrix} q & \tilde{q} & \cdots & \tilde{q} \\ \tilde{q} & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \tilde{q} \\ \tilde{q} & \cdots & \tilde{q} & q \end{bmatrix}, \quad R = \begin{bmatrix} r & \tilde{r} & \cdots & \tilde{r} \\ \tilde{r} & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \tilde{r} \\ \tilde{r} & \cdots & \tilde{r} & r \end{bmatrix} \tag{B.50}$$

The action takes the following form:

$$\begin{aligned} S^v(q, r, \tilde{q}, \tilde{r}) &= 2n \left(S_0(q, r) + S_1^v(q, r, \tilde{q}, \tilde{r}) \right) \\ S_0(q, r) &= \lambda \psi_1^2 \psi_2 q + \psi_2 \log \left(\frac{\mu_\star^2 \psi_1 q}{\mu_1^2 \psi_1 r + 1} + 1 \right) + \frac{r}{q} + (1 - \psi_1) \log(q) + \psi_2 \log \left(\mu_1^2 \psi_1 r + 1 \right) - \log(r) \\ S_1^v(q, r, \tilde{q}, \tilde{r}) &= f^v(q, r) \tilde{q} + g^v(q, r) \tilde{r} \\ f^v(q, r) &= \lambda \psi_1^2 \psi_2 + \frac{\mu_\star^2 \psi_1 \psi_2}{\mu_\star^2 \psi_1 q + \mu_1^2 \psi_1 r + 1} + \frac{1 - \psi_1}{q} - \frac{r}{q^2} \\ g^v(q, r) &= -\frac{\mu_\star^2 \mu_1^2 \psi_1^2 \psi_2 q}{\left(\mu_1^2 \psi_1 r + 1 \right) \left(\mu_\star^2 \psi_1 q + \mu_1^2 \psi_1 r + 1 \right)} + \frac{\mu_1^2 \psi_1 \psi_2}{\mu_1^2 \psi_1 r + 1} + \frac{1}{q} - \frac{1}{r}. \end{aligned} \tag{B.51}$$

Fluctuations around the saddle point

We introduce the following notations:

$$\begin{aligned} \left[\nabla_{\boldsymbol{T}} F(\boldsymbol{T}_\star) \right]_{\alpha\beta} &= \frac{\partial F}{\partial \boldsymbol{T}_{\alpha\beta}} |_{\boldsymbol{T}_\star}, \\ \left[H_{\boldsymbol{T}} F(\boldsymbol{T}_\star) \right]_{\alpha\beta,\gamma\delta} &= \left[\begin{matrix} \frac{\partial^2 F}{\partial Q_{\alpha\beta} \partial Q_{\gamma\delta}} & \frac{\partial^2 F}{\partial Q_{\alpha\beta} \partial R_{\gamma\delta}} \\ \frac{\partial^2 F}{\partial R_{\alpha\beta} \partial Q_{\gamma\delta}} & \frac{\partial^2 F}{\partial R_{\alpha\beta} \partial R_{\gamma\delta}} \end{matrix} \right] \Bigg|_{\boldsymbol{T}_\star}, \\ H\left[F\right] &= \left[\begin{matrix} \frac{\partial F}{\partial q \partial q} & \frac{\partial F}{\partial q \partial r} & \frac{\partial F}{\partial q \partial \tilde{q}} & \frac{\partial F}{\partial q \partial \tilde{r}} \\ \frac{\partial F}{\partial q \partial r} & \frac{\partial F}{\partial r \partial r} & \frac{\partial F}{\partial r \partial \tilde{q}} & \frac{\partial F}{\partial r \partial \tilde{r}} \\ \frac{\partial F}{\partial q \partial \tilde{q}} & \frac{\partial F}{\partial r \partial \tilde{q}} & \frac{\partial F}{\partial \tilde{q} \partial \tilde{q}} & \frac{\partial F}{\partial \tilde{q} \partial \tilde{r}} \\ \frac{\partial F}{\partial q \partial \tilde{r}} & \frac{\partial F}{\partial r \partial \tilde{r}} & \frac{\partial F}{\partial \tilde{q} \partial \tilde{r}} & \frac{\partial F}{\partial \tilde{r} \partial \tilde{r}} \end{matrix} \right] \Bigg|_{\substack{q=q^* \\ r=r^* \\ \tilde{r}=0 \\ \tilde{q}=0}}. \end{aligned}$$

Proposition Let $q^\star$ and $r^\star$ be the solutions of the fixed point equation for the function $S_0 : (q, r) \mapsto \mathbb{R}$ defined in (B.51):

$$\begin{cases} \frac{\partial S_0(q,r)}{\partial q} = 0 \\ \frac{\partial S_0(q,r)}{\partial r} = 0. \end{cases}$$

Then we have that

$$\Psi_3^v = \frac{1}{D} \operatorname{Tr} \left[H\left[S^v\right]^{-1} H \left[P_{\Psi_3^v} \right] \right]. \tag{B.52}$$

Sketch of proof Solving the saddle point equations:

$$\begin{cases} \frac{\partial S^v(q,r,\tilde{q},\tilde{r})}{\partial q} = 0 \\ \frac{\partial S^v(q,r,\tilde{q},\tilde{r})}{\partial q} = 0 \\ \frac{\partial S^v(q,r,\tilde{q},\tilde{r})}{\partial \tilde{q}} = 0 \\ \frac{\partial S^v(q,r,\tilde{q},\tilde{r})}{\partial \tilde{r}} = 0 \end{cases},$$

one finds $\tilde{q} = \tilde{r} = 0$, which is problematic because the prefactor vanishes: $P_{\Psi_3} \propto \mu_1^2\tilde{q} + \mu_\star^2\tilde{r}$. Therefore we must go beyond the saddle point contribution to obtain a non zero result, i.e. we have to examine the quadratic fluctuations around the saddle point. To do so we preform a second-order expansion of the action (B.49) as a function of Q and R:

$$\begin{aligned} P_{\Psi_3^v}(\boldsymbol{T}) &\approx P_{\Psi_3^v}(\boldsymbol{T}_\star) + (\boldsymbol{T}-\boldsymbol{T}_\star)^\top \nabla P_{\Psi_3^v}(\boldsymbol{T}_\star) + \frac{1}{2}(\boldsymbol{T}-\boldsymbol{T}_\star)^\top H_{\boldsymbol{T}}\left[P_{\Psi_3^v}(\boldsymbol{T}_\star)\right](\boldsymbol{T}-\boldsymbol{T}_\star), \\ S^v(\boldsymbol{T}) &\approx S_0(\boldsymbol{T}_\star) + \frac{1}{2}(\boldsymbol{T}-\boldsymbol{T}_\star)^\top H_{\boldsymbol{T}}\left[S^v(\boldsymbol{T}_\star)\right](\boldsymbol{T}-\boldsymbol{T}_\star). \end{aligned}$$

Computing the second derivative of (B.49), it is easy to show that:

$$\begin{aligned} \left[H_{\boldsymbol{T}}\left[S^v(\boldsymbol{T})\right]\right]_{\alpha\beta,\gamma\delta} &= \left[H_{\boldsymbol{T}}\left[S^v(\boldsymbol{T})\right]\right]_{\alpha\beta}\left(\delta_{\alpha\gamma}\delta_{\beta\delta} + \delta_{\alpha\delta}\delta_{\beta\gamma}\right), \\ \left[H_{\boldsymbol{T}}\left[P_{\Psi_3^v}(\boldsymbol{T})\right]\right]_{\alpha\beta,\gamma\delta} &= \left[H_{\boldsymbol{T}}\left[P_{\Psi_3^v}(\boldsymbol{T})\right]\right]_{\alpha\beta}\left(\delta_{\alpha\gamma}\delta_{\beta\delta} + \delta_{\alpha\delta}\delta_{\beta\gamma}\right), \end{aligned}$$

where

$$\begin{aligned} H_{\boldsymbol{T}}\left[F\right]_{\alpha\beta} &= \begin{bmatrix} \frac{\partial^2 F}{\partial Q_{\alpha\beta}\partial Q_{\alpha\beta}} & \frac{\partial^2 F}{\partial Q_{\alpha\beta}\partial R_{\alpha\beta}} \\ \frac{\partial^2 F}{\partial R_{\alpha\beta}\partial Q_{\alpha\beta}} & \frac{\partial^2 F}{\partial R_{\alpha\beta}\partial R_{\alpha\beta}} \end{bmatrix} \\ &= \frac{1}{2n}\delta_{\alpha\beta}\begin{bmatrix} \frac{\partial^2 F}{\partial q\partial q} & \frac{\partial^2 F}{\partial q\partial r} \\ \frac{\partial^2 F}{\partial r\partial q} & \frac{\partial^2 F}{\partial r\partial r} \end{bmatrix} + \frac{2}{2n(2n-1)}(1-\delta_{\alpha\beta})\begin{bmatrix} \frac{\partial^2 F}{\partial \tilde{q}\partial \tilde{q}} & \frac{\partial^2 F}{\partial \tilde{q}\partial \tilde{r}} \\ \frac{\partial^2 F}{\partial \tilde{r}\partial \tilde{q}} & \frac{\partial^2 F}{\partial \tilde{r}\partial \tilde{r}} \end{bmatrix}. \end{aligned}$$

Hence,

$$\begin{aligned} \Psi_3^v &= \lim_{n\to 0}\int d\boldsymbol{T} P_{\Psi_3^v}(\boldsymbol{T}) \exp^{-\frac{D}{2}S^v(\boldsymbol{T})}, \\ &= \lim_{n\to 0} \underbrace{P_{\Psi_3^v}(\boldsymbol{T}_\star)}_{0} + \nabla P_{\Psi_3^v}(\boldsymbol{T}_\star)_{\alpha\beta} \underbrace{\int d\boldsymbol{T}(\boldsymbol{T}-\boldsymbol{T}_\star)_{\alpha\beta}\exp\left(-\frac{D}{2}S^v(\boldsymbol{T})\right)}_{0} \\ &\quad + \frac{1}{2}H_{\boldsymbol{T}}\left[P_{\Psi_3^v}(\boldsymbol{T}_\star)\right]_{\alpha\beta}\int d\boldsymbol{T}(\boldsymbol{T}-\boldsymbol{T}_\star)^2_{\alpha\beta}\exp\left(-\frac{D}{2}S^v(\boldsymbol{T})\right) \\ &= \lim_{n\to 0}\frac{1}{2}H_{\boldsymbol{T}}\left[P_{\Psi_3^v}(\boldsymbol{T}_\star)\right]_{\alpha\beta} e^{-\frac{D}{2}S^v(\boldsymbol{T}_\star)}\int d\boldsymbol{T}(\boldsymbol{T}-\boldsymbol{T}_\star)^2_{\alpha\beta} e^{-\frac{D}{2}\sum_{\alpha\beta}(\boldsymbol{T}-\boldsymbol{T}_\star)^2_{\alpha\beta}H_{\boldsymbol{T}}[S^v(\boldsymbol{T}_\star)]_{\alpha\beta}} \\ &= \lim_{n\to 0}\frac{1}{2}\frac{(2\pi)^n}{\det H_{\boldsymbol{T}}\left[S^v(\boldsymbol{T}_\star)\right]} e^{-\frac{D}{2}S^v(\boldsymbol{T}_\star)}\frac{2}{D}H_{\boldsymbol{T}}\left[P_{\Psi_3^v}(\boldsymbol{T}_\star)\right]_{\alpha\beta} H_{\boldsymbol{T}}\left[S^v(\boldsymbol{T}_\star)\right]^{-1}_{\alpha\beta} \\ &= \frac{1}{D}\operatorname{Tr}\left[H\left[S^v\right]^{-1}H\left[P_{\Psi_3^v}\right]\right] \end{aligned}$$

In the last step, we used the fact that:

$$\lim_{n\to 0} e^{-\frac{D}{2}S^v(\boldsymbol{T}_\star)} = \lim_{n\to 0} e^{-nDS_0(q_\star,r_\star)} = 1,$$
$$\lim_{n\to 0} \det H_{\boldsymbol{T}}\left[S^v(\boldsymbol{T}_\star)\right] = 1.$$

The last equality follows from the fact that for a matrix of size $n\times n$ of the form $M_{\alpha\beta} = a\delta_{\alpha\beta}+b(1-\delta_{\alpha\beta})$, we have

$$\det M = (a-b)^n\left(1+\frac{nb}{a-b}\right) \xrightarrow[n\to 0]{} 1.$$

Expression of the vanilla terms

Using the above procedure, one can compute the terms $\Psi_1, \Psi_2^v, \Psi_3^v$ of (B.23): for each of these terms, the action is the same as in (B.49), and the prefactors can be obtained as:

$$P_{\Psi_1}[Q,R] = \mu_1^2\psi_1\psi_2\left(M_X^{11} + \mu_1^2\mu_\star^2\psi_1^2\left(M_X M_W M_X\right)^{11} + \mu_\star^2\mu_1\psi_1\left(M_X M_W\right)^{11}\right),$$
$$P_{\Psi_2^v}[Q,R] = D\psi_1^2\psi_2\left(\mu_1^2R^{12}+\mu_\star^2Q^{12}\right)\left(\mu_1^2\psi_2 P_{XX} - 2\mu_1^2\mu_\star^2\psi_1\psi_2 P_{XW} + \mu_\star^2 P_{WW}\right),$$
$$P_{\Psi_3^v}[Q,R] = D\psi_1^2\psi_2(\mu_1^2R^{12}+\mu_\star^2Q^{12})\left[\mu_\star^2\left[M_W^v\right]^{12} + \mu_1^2\left(M_X^{12}+\mu_1^2\mu_\star^2\psi_1^2\left[M_X M_W^v M_X\right]^{12}\right) + 2\mu_1^2\mu_\star^2\psi_1\left[M_X M_W^v\right]^{12}\right],$$
$$P_{XX} = N_X^{12} + \frac{1}{\psi_2}M_X^{12} + 2\left(\mu_1\mu_\star\psi_1\right)^2\left[N_X M_W M_X\right]^{12} + \frac{\left(\mu_1\mu_\star\psi_1\right)^2}{\psi_2}\left[M_X M_W M_X\right]^{12} + \left(\mu_1\mu_\star\psi_1\right)^4\left[M_X M_W N_X M_W M_X\right]^{12},$$
$$P_{XW} = \left[N_X M_W\right]^{12} + \frac{1}{\psi_2}\left[M_X M_W\right]^{12} + \left(\mu_1\mu_\star\psi_1\right)^2\left[M_X M_W N_X M_W\right]^{12},$$
$$P_{WW} = M_W^{12} + \psi_2\left(\mu_1\mu_\star\psi_1\right)^2\left[M_W N_X M_W\right]^{12}.$$

Where a new term appears:

$$\left[N_X\right]^{\alpha\beta} = \left[R(I+\mu_1^2\psi_1 R)^{-2}\right]^{\alpha\beta}. \tag{B.53}$$

B.2.4 Computation of the *ensembling* terms

Expression of the action and the prefactor

In the *ensembling* terms, the two inverse matrices are different, hence one has to introduce two distinct replica variables. We distinguish them by the use of an extra index $a\in\{1,2\}$, denoted in brackets in order not to be confused with the replica indices α.

$$\left[M^{(1)}\right]^{-1}_{ij}\left[M^{(2)}\right]^{-1}_{kl} = \lim_{n\to 0}\int\left(\prod_{\alpha=1}^{n}\prod_{a=1}^{2}d\eta^{\alpha(a)}\right)\eta_i^{1(1)}\eta_j^{1(1)}\eta_k^{1(2)}\eta_l^{1(2)}\exp\left(-\frac{1}{2}\sum_{(a)}\eta^{\alpha(a)}M_{ij}^{(a)}\eta^{\alpha(a)}\right). \tag{B.54}$$

Calculations of the Gaussian integrals follow through in a very similar way as for the *vanilla* terms. The matrices appearing in the process are:

$$(G_X^e)_{ii'} = \delta_{ii'} + \frac{\mu_1^2\psi_1}{D}\sum_{(a)\alpha}\lambda_i^{\alpha(a)}\lambda_{i'}^{\alpha(a)}, \tag{B.55}$$

$$(J_X^e)_i = \frac{\mu_1\mu_\star\sqrt{\psi_1}}{D^2}\sum_{\alpha a}\lambda_i^{\alpha(a)}\eta_h^{\alpha(a)}W_h^{(a)}, \tag{B.56}$$

$$(G_W^e)_{hh'}^{(ab)} = \delta_{hh'}^{ab} + \frac{\mu_\star^2}{D}\sum_{\alpha\beta}\eta_h^{\alpha(a)}A_e^{\alpha\beta,ab}\eta_{h'}^{\beta(b)}, \tag{B.57}$$

$$(J_W^e)_h = 0 \tag{B.58}$$

$$(G_\Theta^e)_{hh',ii'}^{(ab)} = \delta_{hh',ii'}^{ab} \tag{B.59}$$

$$(J_\Theta^e)_{hi}^{(a)} = -i\sum_\alpha \hat{\lambda}_i^{\alpha(a)}\eta_h^{\alpha(a)}, \tag{B.60}$$

$$(G_{\hat{\lambda}}^e)_{ii'}^{\alpha\beta,(ab)} = 2\delta^{ab}\delta^{ii'}\eta_h^{\alpha(a)}\eta_h^{\beta(b)}, \tag{B.61}$$

$$(J_{\hat{\lambda}}^e)_i^{\alpha(a)} = i\sqrt{P}\lambda_i^{\alpha(a)}. \tag{B.62}$$

$$\tag{B.63}$$

with

$$A_e^{\alpha\beta,(ab)} = \delta_{ab}\delta_{\alpha\beta} - \mu_1^2\psi_1\frac{1}{D}\sum_{i,j}\lambda_i^{\alpha(a)}[G_X^{e-1}]_{ij}\lambda_j^{\beta(b)}, \tag{B.64}$$

$$[H_W^e]_{ii'} = \left[G_X^{e-1}\right]_{ii'} + \sum_{\alpha\beta,ab}\left[G_X^{e-1}\lambda^{\alpha(a)}\right]_i\left[G_X^{e-1}\lambda^{\beta(b)}\right]_{i'}\left[\eta^{\alpha(a)}\left[G_W^{e-1}\right]^{(ab)}\eta^{\beta(b)}\right], \tag{B.65}$$

$$[S_W^e]_{ih} = \frac{1}{D}\sum_{\alpha,a}\left[G_X^{e-1}\lambda^{\alpha(a)}\right]_i\left[G_W^{e-1}\eta^{\alpha(a)}\right]_h. \tag{B.66}$$

Starting with the computation of Ψ_3^e in order to illustrate the method used, the prefactor $P_{\Psi_3^e}$ and the action are S^e are given by:

$$P_{\Psi_3^e}[\eta,\lambda] := \frac{\mu_1^2\psi_1\psi_2}{D}\lambda_i^{1(1)}\lambda_i^{1(2)}\left[\mu_\star^2\left[\eta^{1(1)}(G_W^{e-1})^{(12)}\eta^{1(2)}\right] + \mu_1^2\psi_1\left[\lambda^{1(1)}H_W^e\lambda^{1(2)}\right] + 2\mu_1^2\mu_\star^2\psi_1[\lambda^{1(1)}S_W^e\eta^{1(2)}]\right], \tag{B.67}$$

$$S^e[\eta,\lambda] := \psi_2\log\det(G_X^e) + \psi_2\log\det(G_W^e) + \log\det(G_{\hat{\lambda}}^e) + \frac{1}{D}\psi_1\psi_2\lambda\sum(\eta_h^{\alpha(a)})^2 + \frac{1}{2D}\left(\lambda_i^{\alpha(a)}(G_{\hat{\lambda}}^{e-1})_{ii'}^{\alpha\beta,ab}\lambda_{i'}^{\beta(b)}\right). \tag{B.68}$$

Expression of the action and the prefactor in terms of order parameters

This time, because of the two different systems, the order parameters carry an additional index a, which turns them into 2×2 block matrices:

$$1 = \int dQ_{\alpha\beta}^{(ab)}d\hat{Q}_{\alpha\beta}^{(ab)}e^{\hat{Q}_{\alpha\beta}^{(ab)}(PQ_{\alpha\beta}^{(ab)} - \eta_h^{\alpha(a)}\eta_h^{\beta(b)})}, \tag{B.69}$$

$$1 = \int dR_{\alpha\beta}^{(ab)}d\hat{R}_{\alpha\beta}^{(ab)}e^{\hat{R}_{\alpha\beta}^{(ab)}(dR_{\alpha\beta}^{(ab)} - \lambda_i^{\alpha(a)}\lambda_i^{\beta(b)})}. \tag{B.70}$$

The systems being decoupled, we make the following ansatz for the order parameters:

$$Q = \left[\begin{array}{ccc|ccc} q & \cdots & 0 & \tilde{q} & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & q & 0 & \cdots & \tilde{q} \\ \hline \tilde{q} & \cdots & 0 & q & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & \tilde{q} & 0 & \cdots & q \end{array}\right], \quad R = \left[\begin{array}{ccc|ccc} r & \cdots & 0 & \tilde{r} & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & r & 0 & \cdots & \tilde{r} \\ \hline \tilde{r} & \cdots & 0 & r & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & \tilde{r} & 0 & \cdots & r \end{array}\right]. \tag{B.71}$$

In virtue of the simple structure of the above matrices, the replica indices α trivialize and we may replace the matrices Q and R by the 2×2 matrices:

$$Q = \begin{bmatrix} q & \tilde{q} \\ \tilde{q} & q \end{bmatrix}, \quad R = \begin{bmatrix} r & \tilde{r} \\ \tilde{r} & r \end{bmatrix}.$$

Define:

$$[M_X^e]_{(ab)} \equiv \frac{1}{D} \lambda_i^{(a)} \left[G_X^{e-1}\right]_{ij} \lambda_j^{(b)} = \left[R(I + \mu_1^2 \psi_1 R)^{-1}\right]_{(ab)}, \tag{B.72}$$

$$[M_W^e]_{(ab)} \equiv \frac{1}{P} \eta^{(a)} \left[G_W^{e-1}\right]^{(ab)} \eta^{(b)} = \left[Q(I + \psi_1 A^e Q)^{-1}\right]_{(ab)}, \tag{B.73}$$

where products are now over 2×2 matrices. Then, one has:

$$\begin{aligned} P_{\Psi_3^e}[Q,R] :=& D\mu_1^2\psi_1^2\psi_2 R^{(12)} \left[\mu_\star^2 M_W^{e(12)} + \mu_1^2 \left(M_X^{e(12)} + \mu_1^2\mu_\star^2\psi_1^2 \left[M_X^e M_W^e M_X^e\right]^{(12)}\right) + 2\mu_1^2\mu_\star^2\psi_1 \left[M_X^e M_W^e\right]^{(12)}\right], \\ S^e[Q,R] :=& \psi_2 \log\det(G_X^e) + \psi_2 \log\det(G_W^e) + \sum_a \left[(\psi_1^2\psi_2\lambda)\mathrm{Tr} Q^{(a)(a)} + \mathrm{Tr}\left(R^{(a)(a)}(Q^{-1})^{(a)(a)}\right) + \log\det Q^{(a)(a)}\right] \\ & - \psi_1 \log\det Q - \log\det R. \end{aligned}$$

Where we have:

$$M_X^e = \frac{1}{(1+\mu_1^2\psi_1 r)^2 - (\mu_1^2\psi_1\tilde{r})^2} \begin{bmatrix} r + \mu_1^2\psi_1(r^2 - \tilde{r}^2) & \tilde{r} \\ \tilde{r} & r + \mu_1^2\psi_1(r^2 - \tilde{r}^2) \end{bmatrix}, \tag{B.74}$$

$$A_{(ab)}^e = \delta_{(ab)} - \mu_1^2\psi_1 \left[M_X^e\right]_{(ab)}, \tag{B.75}$$

$$[M_W^e]_{(ab)} = q\delta_{(ab)} - \mu_\star^2\psi_1 \left[A^e(I + \mu_\star^2\psi_1 q A^e)^{-1}\right]_{(ab)}, \tag{B.76}$$

$$\det(G_X^e) = \det(\delta_{(ab)} + \psi_1\mu_1^2 R_{(ab)}), \tag{B.77}$$

$$\det(G_W^e) = \det(\delta_{ab} + \psi_1 q A_{(ab)}^e). \tag{B.78}$$

Finally, we are left with:

$$\begin{aligned} S^e(q,r,\tilde{q},\tilde{r}) =& n\left(S_0(q,r) + S_1^e(q,r,\tilde{q},\tilde{r})\right), \\ S_1^e(q,r,\tilde{q},\tilde{r}) =& \tilde{r}^2 f^e(q,r) + \tilde{q}^2 g^e(q,r), \\ f^e(q,r) =& \frac{2r\mu_1^2\psi_1\left(1 + q\mu_\star^2\psi_1\right) + \left(1 + q\mu_\star^2\psi_1\right)^2 - r^2\mu_1^4\psi_1^2(-1+\psi_2)}{2r^2\left(1 + r\mu_1^2\psi_1 + q\mu_\star^2\psi_1\right)^2}, \\ g^e(q,r) =& \frac{\psi_1}{2q^2}. \end{aligned}$$

Where S_0 was defined in (B.51).

Expression of the ensembling terms

Evaluating the fluctuations around the saddle point follows through in the same way as for the vanilla terms, with the following expressions of the prefactors:

$$P_{\Psi_2^e}[Q,R] = D\psi_1^2\psi_2\mu_1^2\tilde{r}\left[\mu_1^2 P^e_{XX} - 2\mu_1^2\mu_\star^2\psi_1 P^e_{WX} + \mu_\star^2 P^e_{WW}\right], \tag{B.79}$$

$$P_{\Psi_3^e}[Q,R] = D\mu_1^2\psi_1^2\psi_2 R^{(12)}\left[\mu_\star^2 M_W^{e(12)} + \mu_1^2\left(M_X^{e(12)} + \mu_1^2\mu_\star^2\psi_1^2\left[M_X^e M_W^e M_X^e\right]^{(12)}\right) + 2\mu_1^2\mu_\star^2\psi_1\left[M_X^e M_W^e\right]^{(12)}\right], \tag{B.80}$$

$$\begin{aligned} P^e_{XX} &= \psi_2 N_X^{e12} + M_X^{e12} + 2\psi_2(\mu_1\mu_\star\psi_1)^2\left[M_X^e N_X^e M_W^e\right]^{12} + (\mu_1\mu_\star\psi_1)^2\left[M_X^e M_W^e M_X^e\right]^{12} \\ &\quad + \psi_2(\mu_1\mu_\star\psi_1)^4\left[M_X^e M_W^e N_X^e M_W^e M_X^e\right]^{12}, \end{aligned} \tag{B.81}$$

$$P^e_{WX} = \psi_2\left[N_X^e M_W^e\right]^{12} + \left[M_X^e M_W^e\right]^{12} + \psi_2(\mu_1\mu_\star\psi_1)^2\left[M_X^e M_W^e N_X^e M_W^e\right]^{12}, \tag{B.82}$$

$$P^e_{WW} = \left[M_W^e\right]^{12} + \psi_2(\mu_1\mu_\star\psi_1)^2\left[M_W^e N_X^e M_W^e\right]^{12}. \tag{B.83}$$

B.2.5 Computation of the *bagging* term

Here, we are interested in computing the term Ψ_2^b. This term differs from the previous ones in that there are now two independent data matrices $\boldsymbol{X}^{(1)}$ and $\boldsymbol{X}^{(2)}$. The calculations for the action and the prefactor are very similar to calculations performed for the *ensembling* terms Ψ_2^e, Ψ_3^e, with the addition that $\boldsymbol{X}$ now also carries an index (a).
Firstly let us write Ψ_2^b as a trace over random matrices:

$$\Psi_2^b = \frac{\mu_1^2}{d^2}\operatorname{Tr}\left[\boldsymbol{X}^{(1)}\boldsymbol{Z}^{(1)}\boldsymbol{B}^{(1)-1}\boldsymbol{\Theta}^{(1)}\boldsymbol{\Theta}^{(2)}\boldsymbol{B}^{(2)-1}\boldsymbol{Z}^{(2)}\boldsymbol{X}^{(2)}\right]. \tag{B.84}$$

Calculations follow through in the same way as in the previous sections. Using the replica formula (B.54), and performing the integrals over the Gaussian variables the following quantities appear:

$$(G_X^b)_{ii'}^{(ab)} = \delta_{ii'} + \frac{\mu_1^2\psi_1}{D}\sum_\alpha \lambda_i^{\alpha(a)}\lambda_{i'}^{\alpha(a)}, \tag{B.85}$$

$$(J_X^d)_i^{(a)} = \frac{\mu_1\mu_\star\sqrt{\psi_1}}{D^2}\sum_\alpha \lambda_i^{\alpha(a)}\eta_h^{\alpha(a)}W_h^{(a)}, \tag{B.86}$$

$$(G_W^b)_{hh'}^{(ab)} = \delta_{hh'}^{ab} + \frac{\mu_\star^2}{D}\sum_{\alpha\beta}\eta_h^{\alpha(a)}A^{\alpha\beta,ab}\eta_{h'}^{\beta(b)}, \tag{B.87}$$

$$(J_W^b)_h = 0, \tag{B.88}$$

$$(G_\Theta^b)_{hh',ii'}^{(ab)} = \delta_{hh',ii'}^{ab}, \tag{B.89}$$

$$(J_\Theta^b)_{hi}^{(a)} = -i\sum_\alpha \hat{\lambda}_i^{\alpha(a)}\eta_h^{\alpha(a)}, \tag{B.90}$$

$$(G_{\hat{\lambda}}^b)_{ii'}^{\alpha\beta,(ab)} = 2\delta^{ab}\delta^{ii'}\eta_h^{\alpha(a)}\eta_h^{\beta(b)}, \tag{B.91}$$

$$(J_{\hat{\lambda}}^b)_i^{\alpha(a)} = i\sqrt{P}\lambda_i^{\alpha(a)}. \tag{B.92}$$

$$\tag{B.93}$$

The saddle point ansatz for Q and R is the same as the one for the ensembling terms (see (B.71)). The procedure to evaluate Ψ_2^b is also the same as the one for Ψ_2^e except the Hessian is taken with respect to

S^b. The final result is given below.

$$
\begin{aligned}
P_{\Psi_2^b}\left[Q,R\right] =& D\mu_1^2\psi_1\psi_2^2\tilde{r}\left[\psi_1\mu_1^2 P_{XX} + 2\mu_\star^2\mu_1^2\psi_1^2 P_{WX} + \mu_\star^2\psi_1 P_{WW}\right], \\
P_{XX} =& \left(N_X^{11} + 2(\mu_1\mu_\star\psi_1)^2\left[N_X M_W M_X\right]^{11} + (\mu_1\mu_\star\psi_1)^4\left[M_X M_W N_X M_W M_X\right]^{11}\right), \\
P_{WX} =& \left[N_X M_W\right]^{11} + (\mu_1\mu_\star\psi_1)^2\left[M_X M_W N_X M_W\right]^{11}, \\
P_{WW} =& (\mu_1\mu_\star\psi_1)^2\left[M_W N_X M_W\right]^{11}, \\
S^b[q,r,\tilde{q},\tilde{r}] =& n\left(S_0(q,r) + S_1^b(q,r,\tilde{q},\tilde{r})\right), \\
S_1^b(q,r,\tilde{q},\tilde{r}) =& \frac{\tilde{r}^2}{r^2} + \frac{\psi_1\tilde{q}^2}{q^2}.
\end{aligned}
$$

With:

$$
\begin{aligned}
M_X &= \frac{r}{1+\mu_1^2\psi_1 r}, \\
A &= 1 - \mu_1^2\psi_1 M_X, \\
M_W &= \frac{q}{1+\mu_\star^2\psi_1 q A}, \\
N_X &= \frac{\tilde{r}}{(1+\mu_1^2\psi_1 r)^2}.
\end{aligned}
$$

Appendix C

Triple Descent and the Two Kinds of Overfitting: When and Why do they Occur?

C.1 Effect of signal-to-noise ratio and nonlinearity

C.1.1 RF model

In the RF model, varying r can easily be achieved analytically and yields interesting results, as shown in Fig. C.1[1].
In the top panel, we see that the parameter-wise profile exhibits double descent for all degrees of linearity r and signal-to-noise ratio SNR, except in the linear case $r = 1$ which is monotonously deceasing. Increasing the degree of nonlinearity (decreasing r) and the noise (decreasing the SNR) simply makes the nonlinear peak stronger.
In the bottom panel, we see that the sample-wise profile is more complex. In the linear case $r = 1$, only the linear peak appears (except in the noiseless case). In the nonlinear case $r < 1$, the nonlinear peak appears is always visible; as for the linear peak, it is regularized away, except in the strong noise regime $\mathrm{SNR} > 1$ when the degree of nonlinearity is small ($r > 0.8$), where we observe the triple descent.
Notice that both in the parameter-wise and sample-wise profiles, the test loss profiles change smoothly with r, except near $r = 1$ where the behavior abruptly changes, particularly at low SNR.
One can also mimick these results numerically by considering, as in [149], the following family of piecewise linear functions:

$$\sigma_\alpha(x) = \frac{[x]_+ + \alpha[-x]_+ - \frac{1+\alpha}{\sqrt{2\pi}}}{\sqrt{\frac{1}{2}(1+\alpha^2) - \frac{1}{2\pi}(1+\alpha)^2}}, \tag{C.1}$$

for which

$$r_\alpha = \frac{(1-\alpha)^2}{2(1+\alpha^2) - \frac{2}{\pi}(1+\alpha)^2}. \tag{C.2}$$

[1]We focus here on the practically relevant setup $N/D \gg 1$. Note from the (P, N) phase-space that things can be more complex at $N/D \lesssim 1$).

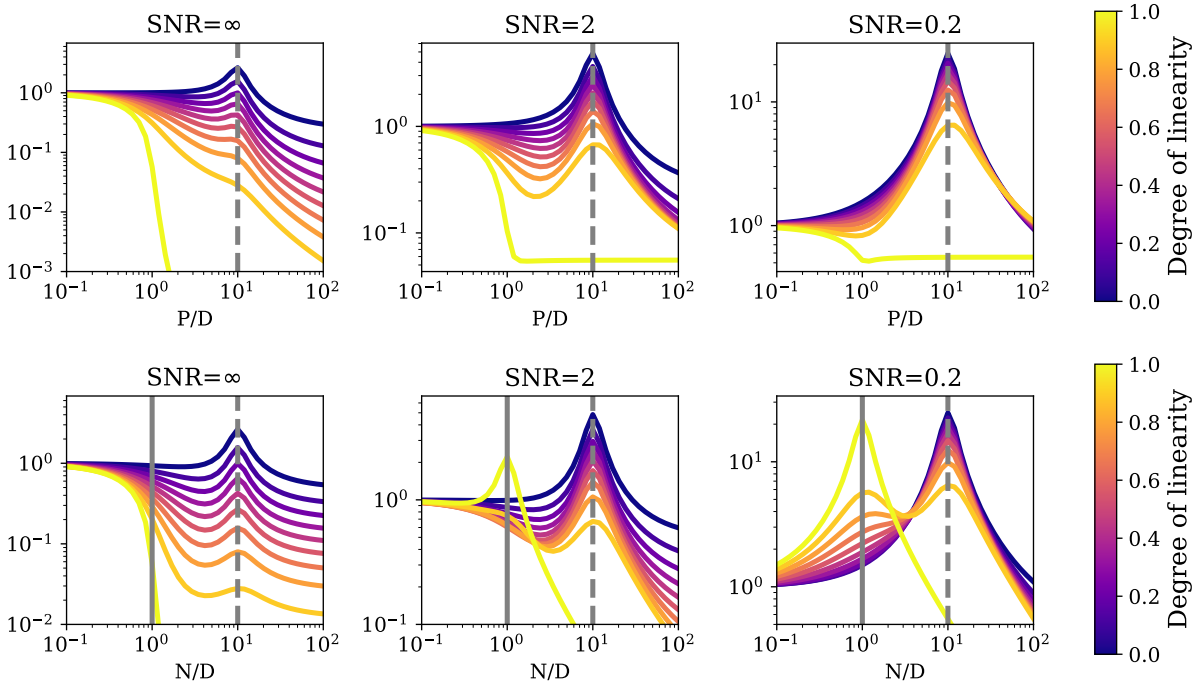


Figure C.1: Analytical parameter-wise (**top**, $N/D = 10$) and sample-wise (**bottom**, $P/D = 10$) test loss profiles of the RF model. **Left**: noiseless case, $\mathrm{SNR} = \infty$. **Center**: low noise, $\mathrm{SNR} = 2$. **Right**: high noise, $\mathrm{SNR} = 0.2$. We set $\gamma = 10^{-1}$.

Here, α parametrizes the ratio of the slope of the negative part to the positive part and allows to adjust the value of r continuously. $\alpha = -1$ ($r = 1$) will correspond to a (shifted) absolute value, $\alpha = 1$ ($r = 0$) will correspond to a linear function, $\alpha = 0$ will correspond to a (shifted) ReLU. In Fig. C.2, we show the effect of sweeping α uniformly from 1 to -1 (which causes r to range from 0 to 1). As expected, we see the linear peak become stronger and the nonlinear peak become weaker.

C.1.2 NN model

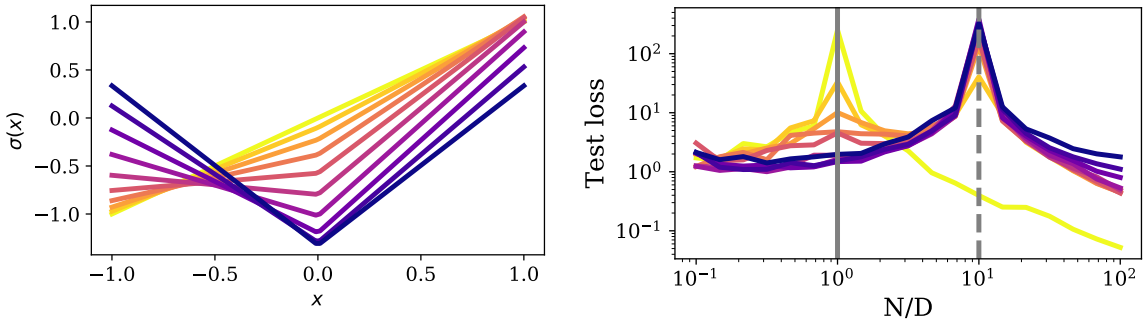
In Fig. C.3, we show the effect of replacing Tanh ($r \sim 0.92$) by ReLU ($r = 0.5$). We still distinguish the two peaks of triple descent, but the linear peak is much weaker in the case of ReLU, as expected from the stronger degree of nonlinearity.

C.2 Origin of the linear peak

In this section, we follow the lines of [122], where the test loss is decomposed in the following way (Eq. D.6):

$$\mathcal{L}_g = \rho + Q - 2M \tag{C.3}$$

$$\rho = \frac{1}{D}\|\boldsymbol{\beta}\|^2, \quad M = \frac{\sqrt{\zeta}}{D}\boldsymbol{b}\cdot\boldsymbol{\beta}, \quad Q = \frac{\zeta}{D}\|\boldsymbol{b}\|^2 + \frac{\eta-\zeta}{P}\|\boldsymbol{a}\|^2, \quad \boldsymbol{b} = \boldsymbol{\Theta}\boldsymbol{a} \tag{C.4}$$



(a) Nonlinearities used

(b) Corresponding test loss

Figure C.2: Moving from a purely nonlinear function to a purely linear function (dark to light colors) strengthens the linear peak and weakens the nonlinear peak.

As before, $\boldsymbol{\beta}$ denotes the linear teacher vector and $\boldsymbol{\Theta}, \boldsymbol{a}$ respectively denote the (fixed) first and (learnt) second layer of the student. This insightful expression shows that the loss only depends on the norm of the second layer $\|\boldsymbol{a}\|$, the norm of the linearized network $\|\boldsymbol{b}\|$, and its overlap with the teacher $\boldsymbol{b} \cdot \boldsymbol{\beta}$. We plot these three terms in Fig. C.4, focusing on the triple descent scenario $\mathrm{SNR} < 1$. In the left panel, we see that the overlap of the student with the teacher is monotically increasing, and reaches its maximal value at a certain point which increases from D to P as we decrease r from 1 to 0. In the central panel, we see that $\|\boldsymbol{a}\|$ peaks at $N = P$, causing the nonlinear peak as expected, but nothing special happens at $N = D$ (except for $r = 1$). However, in the right panel, we see that the norm of the linearized network peaks at $N = D$, where we know from the spectral analysis that the gap of the linear part of the spectrum is minimal. This is the origin of the linear peak.

C.3 Structured datasets

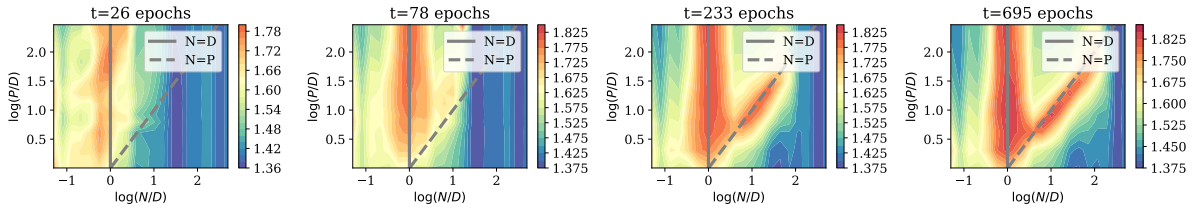
In this section, we examine how our results are affected by considering the realistic case of correlated data. To do so, we replace the Gaussian i.i.d. data by MNIST data, downsampled to 10×10 images for the RF model ($D = 100$) and 14×14 images for the NN model ($D = 196$).

C.3.1 RF model: data structure does not matter in the lazy regime

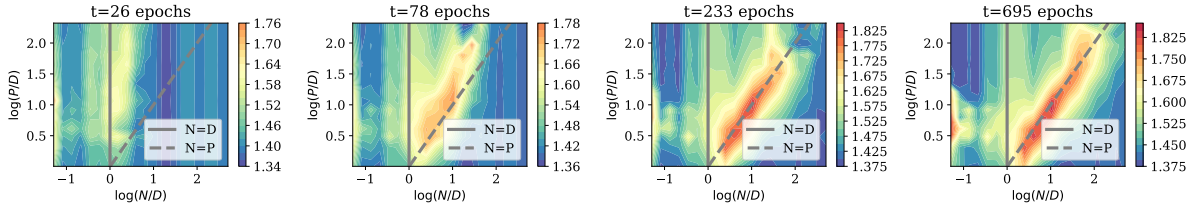
We refer to the results in Fig C.5. Interestingly, the triple descent profile is weakly affected by the correlated structure of this realistic dataset: the linear peak and nonlinear peaks still appear, respectively at $N = D$ and $N = P$. However, the spectral properties of $\boldsymbol{\Sigma} = \frac{1}{N}\boldsymbol{Z}^\top \boldsymbol{Z}$ are changed in an interesting manner: the two parts of the spectrum are now contiguous, there is no gap between the linear part and the nonlinear part.

C.3.2 NN model: the effect of feature learning

As shown in Fig. C.6, the NN model behaves very differently on structured data like CIFAR10. At late times, three main differences with respect to the case of random data can be observed in the low SNR setup:



(a) Tanh



(b) ReLU

Figure C.3: Dynamics of the test loss phase space, with weight decay $\gamma = 0.05$. **Top**: Tanh. **Bottom**: ReLU.

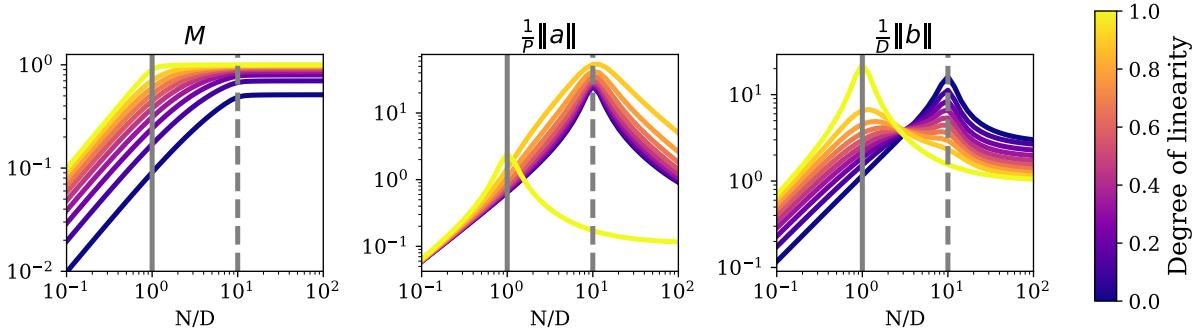
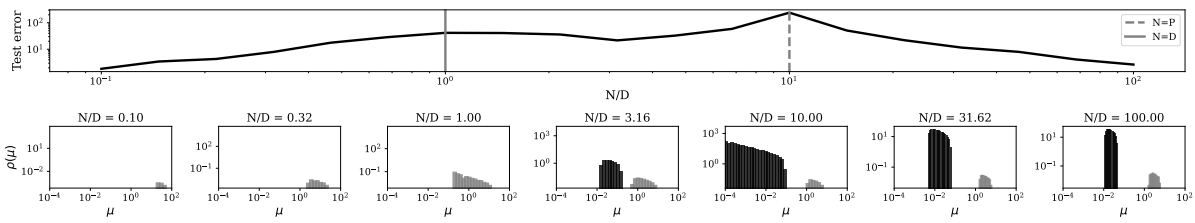


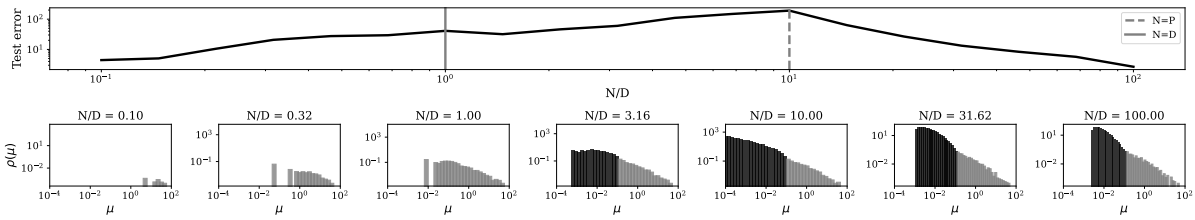
Figure C.4: Terms entering Eq. C.4, plotted at SNR $= 0.2$, $\gamma = 10^{-1}$.

- Instead of a clearly defined linear peak at $N = D$, we observe a large overfitting regions at $N < D$.
- The nonlinear peak shifts to higher values of N during time, but always scales sublinearly with P, i.e. it is located at $N \sim P^{\alpha}$ with $\alpha < 1$

Behavior of the linear peaks As emphasized by [147], a non-trivial structure of the covariance of the input data can strongly affect the number and location of linear peaks. For example, in the context of linear regression, [146] considers gaussian data drawn from a diagonal covariance matrix $\Sigma \in \mathbb{R}^{30\times 30}$ with two blocks of different strengths: $\Sigma_i, i = 10$ for $i \leq 15$ and $\Sigma_i, i = 1$ for $i \geq 15$. In this situation, two linear peaks are observed: one at $N_1 = 15$, and one at $N_2 = 30$. In the setup of structured data such as CIFAR10, where the covariance is evidently much more complicated, one can expect to see a multiple peaks, all located at $N \leq D$.
This is indeed what we observe: in the case of Tanh, where the linear peaks are strong, two linear peaks



(a) Random data

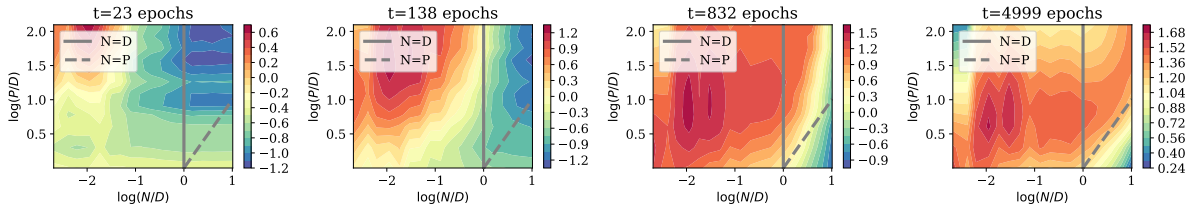


(b) MNIST

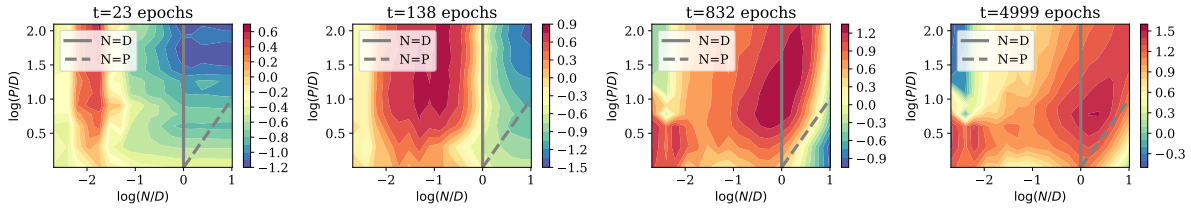
Figure C.5: Spectrum of the covariance of the projected features $\boldsymbol{\Sigma} = \frac{1}{N}\boldsymbol{Z}^\top \boldsymbol{Z}$ at various values of N/D, with the corresponding loss curve shown above. We set $\sigma = \mathrm{Tanh}$, $\gamma = 10^{-5}$.

are particularly at late times, at $N_1 = 10^{-1.5}D$, and the other at $N_2 = 10^{-2}D$. In the case of ReLU, they are obscured at late times by the strength of the nonlinear peak.

Behavior of the nonlinear peak We observe that during training, the nonlinear peak shifts towards higher values of N. This phenomenon was already observed in [33], where it is explained through the notion of effective model complexity. The intuition goes as follows: increasing the training time increases the expressivity of a given model, and therefore increases the number of training examples it can overfit.
We argue that this sublinear scaling is a consequence of the fact that structured data is easier to memorize than random data [461]: as the dataset grows large, each new example becomes easier to classify since the network has learnt the underlying rule (in contrast to the case of random data where each new example makes the task harder by the same amount).



(a) CIFAR10, Tanh



(b) CIFAR10, ReLU

Figure C.6: Dynamics of test loss phase space on CIFAR10 with SNR $= 0.2$, for different activation functions.

Appendix D

On the interplay between Loss Function and Data Structure

D.1 Phase spaces

Comparing square and logistic In Figs. D.1 and D.2, we show how the various observables of interest evolve in the (N, P) phase space, respectively from square loss and logistic loss. To show that our results generalize to different activation functions and regularization levels, we choose $\sigma = \mathrm{ReLU}$ and $\lambda = 0.1$. We make the following observations:

1. **Test error**: the phase-space is almost symmetric in the isotropic setup, with the double descent peak clearly visible at $N = P$ (dashed grey line) for square loss but strongly attenuated for logistic loss. Interestingly, for logistic loss, the peak appears at $N > P$ in the noiseless setup as observed in [122], but shifts to $N = P$ in presence of noise. In the anisotropic setup, the phase space dissymetrizes, with a wide overfitting region emerging in the overparametrized regime around $N = D$ (solid grey line), as observed in [119] for isotropic regression tasks. This overfitting is strongly regularized for logistic loss, explaining its superiority on structured datasets

2. **Train loss**: Here a strong difference appears between square loss and logistic loss. For square loss, the overparametrized region $P > N > D$ reaches zero training loss, and the interpolation threshold clearly appears at $P = N$. For logistic loss, the interpolation threshold depends more strongly on the data structure: it shifts up when we increase the label noise and shifts down when we increase the anisotropy.

3. **Order parameter** Q: recall that this observable represents the variance of the student's outputs. Here the phase space looks completely different for the two loss functions. For square loss, the phase space is almost symmetric in the noiseless isotropic setup and dissymetrizes in presence of noise or anisotropy, with a peak appearing at $N = D$, forming the "linear peak" in the test error [119]. For logistic loss, the phase space is not symmetric: Q becomes very large in the overparametrized regime, leading to overconfidence, as observed in Fig. 5.5 of the main text.

4. **Order parameter** M: recall that this observable represents the covariance between the student's outputs and the teacher's outputs, i.e. the dot product of their corresponding vectors. Again, behavior is very different for square loss and logistic loss. For square loss, the covariance increases symmetrically when increasing P or N, reaching its maximal value respectively at $P = D$ and $N = D$ in the isotropic setup, or earlier in the aligned setup. For logistic loss, the phase space is more complex due to the fact that the norm of the student diverges in the overparametrized regime.

Varying the anisotropy Fig. D.3 illustrates the modification of the phase space of the random features model trained on the strong and weak features model as the saliency r_x of the $\phi_1 D = 0.01D$ relevant features ($r_\beta = 1000$) is gradually increased. At $r_x = 1$ (panel (a)), the data is isotropic and the phase space is symmetric. When $r_x \to \infty$, all the variance goes in the salient subspace and we are left with an isotropic task of dimensionality $\phi_1 D$: the phase space is symmetric again (panel (d)). In between these two extreme scenarios, we see the asymmetry appear in the phase space under the form of an overfitting peak at $N = D$, as the irrelevant features come into play (panels (b) and (c)).

D.2 Analytical derivations

The following sections present the analytical derivations of Chap. 5. First, we provide an outline of the computation of the test error. Second, we describe our extension of the Gaussian Equivalence Theorem to anisotropic data, a key ingredient to handle the strong and weak features model. The three following sections develop the steps of the replica computation, from the Gibbs formulation of the learning objective, to the set of self-consistent equations to obtain the order parameters. Lastly, we explain how to also obtain the asymptotic training loss from the output of the replica computation.

D.2.1 Outline

In the main text, we study the strong and weak features scenario with two blocks, and only for classification tasks, but our derivation will be performed here in full generality.

Setup and notations We recall notations: D is the input dimension, P is the number of random features and N is the number of training examples. We consider the high-dimensional limit where $D, P, N \to \infty$ and $\gamma = \frac{D}{P}$, $\alpha = \frac{N}{P}$ are of order one. The inputs $\boldsymbol{x} \in \mathbb{R}^D$ and the elements of the teacher vectors $\boldsymbol{\beta} \in \mathbb{R}^D$ are sampled from a block-structured covariance matrix:

$$\boldsymbol{x} \sim \mathcal{N}(0, \Sigma_x), \quad \Sigma_x = \begin{bmatrix} \sigma_{x,1}\mathbb{I}_{\phi_1 D} & 0 & 0 \\ 0 & \sigma_{x,2}\mathbb{I}_{\phi_2 D} & 0 \\ 0 & 0 & \mathrm{d}ots \end{bmatrix} \tag{D.1}$$

$$\boldsymbol{\beta} \sim \mathcal{N}(0, \Sigma_\beta), \quad \Sigma_\beta = \begin{bmatrix} \sigma_{\beta,1}\mathbb{I}_{\phi_1 D} & 0 & 0 \\ 0 & \sigma_{\beta,2}\mathbb{I}_{\phi_2 D} & 0 \\ 0 & 0 & \mathrm{d}ots \end{bmatrix} \tag{D.2}$$

$$\tag{D.3}$$

The output of the random feature model is given by:

$$\hat{y}_\mu = \hat{f}\left(\sum_{i=1}^{P} \boldsymbol{a}_i \sigma\left(\frac{\boldsymbol{\Theta}_i \cdot \boldsymbol{x}_\mu}{\sqrt{D}}\right)\right) \equiv \hat{f}\left(\boldsymbol{a} \cdot \boldsymbol{z}_\mu\right), \tag{D.4}$$

where $\sigma(\cdot)$ is a pointwise activation function, $\hat{f}$ is an output function and we introduced the notation $\boldsymbol{z}_\mu \in \mathbb{R}^P$. Elements of $\boldsymbol{\Theta}$ are drawn i.i.d from $\mathcal{N}(0,1)$. The labels are given by a linear ground truth, modulated by an output function f^0 and possibly corrupted by noise through a probabilistic channel $\mathcal{P}$:

$$y_\mu \sim \mathcal{P}\left(\cdot \,\middle|\, f^0\left(\frac{\boldsymbol{\beta} \cdot \boldsymbol{x}_\mu}{\sqrt{D}}\right)\right). \tag{D.5}$$

The second layer weights, i.e. the elements of $\boldsymbol{a} \in \mathbb{R}^P$, are trained by minimizing an ℓ_2-regularized loss on N training examples $\{\boldsymbol{x}_\mu \in \mathbb{R}^D\}_{\mu=1\dots N}$:

$$\hat{\boldsymbol{a}} = \underset{\boldsymbol{a}}{\operatorname{argmin}}\left[\mathcal{L}(\boldsymbol{a})\right], \quad \mathcal{L}(\boldsymbol{a}) = \sum_{\mu=1}^{N} \ell\left(y^\mu, \boldsymbol{a} \cdot \boldsymbol{z}_\mu\right) + \frac{\lambda}{2}\|\boldsymbol{a}\|_2^2. \tag{D.6}$$

We consider both regression tasks, where $\hat{f} = f^0 = \mathrm{id}$, and binary classification tasks, $\hat{f} = f^0 = \mathrm{sign}$. In the first case, ℓ is the square loss and the noise is additive. In the second case, ℓ can be any type of loss (square, hinge, logistic) and the noise amounts to random label flipping.
The test error is given by

$$\epsilon_g = \frac{1}{2^k}\mathbb{E}_{\boldsymbol{x},y}\left[\left(\hat{y}\left(\boldsymbol{x}\right) - y\right)^2\right] \tag{D.7}$$

with $k = 1$ for regression tasks (mean-square error) and $k = 2$ for binary classification tasks (zero-one error).
In the following, we denote as $\boldsymbol{x}|_i, \boldsymbol{\beta}|_i$ the orthogonal projection of $\boldsymbol{x}$ and $\boldsymbol{\beta}$ onto the subspace i of $\mathbb{R}^D$. For example, $\boldsymbol{x}|_1$ amounts to the first $\phi_1 D$ components of $\boldsymbol{x}$.

Steps of the derivation of the test error The key observation is that the test error can be rewritten, in the high-dimensional limit, in terms of so-called *order parameters*:

$$\epsilon_g = \frac{1}{2^k}\mathbb{E}_{\boldsymbol{x}^{\mathrm{new}},y^{\mathrm{new}}}\left(y^{\mathrm{new}} - \hat{f}\left(\frac{1}{\sqrt{P}}\sigma\left(\frac{\boldsymbol{\Theta}^\top \boldsymbol{x}^{\mathrm{new}}}{\sqrt{D}}\right)\cdot\hat{\boldsymbol{a}}\right)\right)^2 \tag{D.8}$$

$$= \frac{1}{2^k}\int \mathrm{d}\nu \int \mathrm{d}\lambda P(\nu,\lambda)(f^0(\nu) - \hat{f}(\lambda))^2 \tag{D.9}$$

where we defined

$$\lambda = \frac{1}{\sqrt{P}}\sigma\left(\frac{\boldsymbol{\Theta}^\top \boldsymbol{x}^{\mathrm{new}}}{\sqrt{D}}\right)\cdot\hat{\boldsymbol{a}} \in \mathbb{R}, \quad \nu = \frac{1}{\sqrt{D}}\boldsymbol{x}^{\mathrm{new}}\cdot\boldsymbol{\beta} \in \mathbb{R} \tag{D.10}$$

and the joint probability distribution

$$P(\nu,\lambda) = \mathbb{E}_{\boldsymbol{x}^{\mathrm{new}}}\left[\delta\left(\nu - \frac{1}{\sqrt{D}}\boldsymbol{x}^{\mathrm{new}}\cdot\boldsymbol{\beta}\right)\delta\left(\lambda - \frac{1}{\sqrt{P}}\sigma\left(\frac{\boldsymbol{\Theta}^\top \boldsymbol{x}^{\mathrm{new}}}{\sqrt{D}}\right)\cdot\hat{\boldsymbol{a}}\right)\right]. \tag{D.11}$$

The key objective to calculate the test error is therefore to obtain the joint distribution $P(\nu,\lambda)$. To do so, our derivation is organized as follows.

1. In Sec. D.2.2, we adapt the Gaussian equivalence theorem [82, 128, 164, 181] to the case of anisotropic data. The latter shows that $P(\nu,\lambda)$ is a joint gaussian distribution whose covariance depends only on the following *order parameters*:

$$m_{s,i} = \frac{1}{D}\boldsymbol{s}|_i\cdot\boldsymbol{\beta}|_i, \quad q_{s,i} = \frac{1}{D}\boldsymbol{s}|_i\cdot\boldsymbol{s}|_i, \quad q_a = \frac{1}{P}\hat{\boldsymbol{a}}\cdot\hat{\boldsymbol{a}},$$

   where we denoted $\boldsymbol{s} = \frac{1}{\sqrt{P}}\boldsymbol{\Theta}\hat{\boldsymbol{a}} \in \mathbb{R}^D$ and the index i refers to the subspace of $\mathbb{R}^D$ the vectors are projected onto.

2. In Sec. D.2.3, we recast the optimization problem as a Gibbs measure over the weights, from which one can sample the average value of the order parameters m_s, q_s, q_a. Thanks to the convexity of the problem, this measure concentrates around the solution of the optimization problem in the limit of high temperature, see [122].

3. In Sec. D.2.4, we leverage tools from random matrix theory to derive the saddle-point equations allowing the obtain the values of the order parameters
4. In Sec. D.2.5, we give the explicit expressions of the saddle-point equations for the two cases studied in the main text: square loss regression and classification
5. In Sec. D.2.6, we also show how to obtain the train error from the order parameters.

Once the order parameters are known, the joint law of λ, ν is determined as explained in the main text:

$$P(\lambda, \nu) = \mathcal{N}(0, \Sigma), \quad \Sigma = \begin{pmatrix} \rho & M \\ M & Q \end{pmatrix}. \tag{D.12}$$

It is then easy to perform the Gaussian integral giving the test error.
For the regression problem where $\hat{f} = f^0 = \mathrm{id}$, we have:

$$\epsilon_g = \frac{1}{2}\left(\rho + Q - 2M\right)$$

For the classification problem where $\hat{f} = f^0 = \mathrm{sgn}$, we have:

$$\epsilon_g = \frac{1}{\pi}\cos^{-1}\left(\frac{M}{\sqrt{\rho Q}}\right)$$

.

D.2.2 The anisotropic Gaussian Equivalence Theorem

We now present the derivation of the anisotropic Gaussian Equivalence Theorem. This is a key ingredient for the replica analysis presented in the next section.
Define, for a Gaussian input vector $\boldsymbol{x} \in \mathbb{R}^D$, the family of vectors indexed by $a = 1 \dots r$:

$$\lambda^a = \frac{1}{\sqrt{P}}\boldsymbol{a}^a \cdot \sigma\left(\frac{\boldsymbol{\Theta x}}{\sqrt{D}}\right) \in \mathbb{R}, \quad \nu = \frac{1}{\sqrt{D}}\boldsymbol{x}\cdot\boldsymbol{\beta} \in \mathbb{R} \tag{D.13}$$

Using the Gaussian equivalence principle, we can obtain expectancies for this family in the high-dimensional limit:

$$(\nu, \lambda^a) \sim \mathcal{N}(0, \Sigma), \quad \Sigma^{ab} = \begin{pmatrix} \rho & M^a \\ M^a & Q^{ab} \end{pmatrix} \in \mathbb{R}^{(r+1)\times(r+1)} \tag{D.14}$$

$$\rho = \sum_i \phi_i \beta|_i \sigma_{x,i}, \quad M^a = \mu_1 \sum_i \sigma_{x,i} m^a_{s,i}, \quad Q^{ab} = \mu_1^2 \sum_i \sigma_{x,i} q^{ab}_{s,i} + \mu_\star^2 q^{ab}_a \tag{D.15}$$

$$m^a_{s,i} = \frac{1}{D}\boldsymbol{s}^a|_i \cdot \boldsymbol{\beta}|_i, \quad q^{ab}_{s,i} = \frac{1}{D}\boldsymbol{s}^a|_i \cdot \boldsymbol{s}^b|_i, \quad q^{ab}_a = \frac{1}{P}\boldsymbol{a}^a \cdot \boldsymbol{a}^b \tag{D.16}$$

where we have:

$$\boldsymbol{s}^a = \frac{1}{\sqrt{P}}\boldsymbol{\Theta a}^a \in \mathbb{R}^D, \quad \boldsymbol{s}^a_i = P_{E_i}\boldsymbol{s}^a, \quad \boldsymbol{\beta}|_i = P_{E_i}\boldsymbol{\beta} \tag{D.17}$$

and denoting $\psi = \sum_i \phi_i \sigma_{x,i}$ and $z \sim \mathcal{N}(0, \psi)$, we defined

$$\mu_0 = \mathbb{E}_z[\sigma(z)], \quad \mu_1 = \frac{1}{\psi}\mathbb{E}_z[z\sigma(z)]], \quad \mu_\star = \sqrt{\mathbb{E}_z\left[\sigma(z)^2\right] - \mu_0^2 - \psi\mu_1^2}. \tag{D.18}$$

Isotropic setup Let us start in the setup where we only have one block, of unit variance.
By rotational invariance, we can write:

$$\mathbb{E}_{\boldsymbol{x}}\left[\sigma\left(\frac{\boldsymbol{x}\cdot\boldsymbol{\Theta}_\mu}{\sqrt{D}}\right)\sigma\left(\frac{\boldsymbol{x}\cdot\boldsymbol{\Theta}_\nu}{\sqrt{D}}\right)\right] = f\left(\frac{\boldsymbol{\Theta}_\mu\cdot\boldsymbol{\Theta}_\nu}{D}\right) \equiv f(q_{\mu\nu}) \tag{D.19}$$

$$= \delta_{\mu\nu}f(1) + (1-\delta_{\mu\nu})\left(f(0) + f'(0)q_{\mu\nu}\right) + o\left(q_{\mu\nu}\right) \tag{D.20}$$

$$= f(0) + f'(0)q_{\mu\nu} + \delta_{\mu\nu}(f(1) - f(0) - f'(0)) \tag{D.21}$$

Therefore, the expectancy is the same as the "Gaussian equivalent model" where we replace

$$\sigma\left(\frac{\boldsymbol{x}\cdot\boldsymbol{\Theta}_\mu}{\sqrt{D}}\right) \to \sqrt{f(0)} + \sqrt{f'(0)}\left(\frac{\boldsymbol{x}\cdot\boldsymbol{\Theta}_\mu}{\sqrt{D}}\right) + \sqrt{(f(1) - f(0) - f'(0)}\xi_\mu, \quad \xi_\mu \sim \mathcal{N}(0,1) \tag{D.22}$$

What remains is to find the expression of $f(0), f(1)$ and $f'(0)$. If $q_{\mu\nu} = 1$, then $\boldsymbol{\Theta}_\mu$ and $\boldsymbol{\Theta}_\nu$ are the same vector and clearly $f(1) = \mathbb{E}_z\left[\sigma(z)^2\right]$.
Otherwise, we use the fact that $x_\mu = \frac{\boldsymbol{x}\cdot\boldsymbol{\Theta}_\mu}{\sqrt{D}}$ and $x_\nu = \frac{\boldsymbol{x}\cdot\boldsymbol{\Theta}_\nu}{\sqrt{D}}$ are correlated gaussian variables:

$$\mathbb{E}\left[x_\mu^2\right] = 1, \qquad \mathbb{E}\left[x_\mu x_\nu\right] = \frac{\boldsymbol{\Theta}_\mu\cdot\boldsymbol{\Theta}_\nu}{D} \equiv q_{\mu\nu} \sim O\left(\frac{1}{\sqrt{D}}\right) \tag{D.23}$$

Therefore we can parametrize as follows,

$$x_\mu = \eta_\mu\sqrt{r - q_{\mu\nu}} + z\sqrt{q_{\mu\nu}} \sim \eta_\mu(1 - \frac{1}{2}q_{\mu\nu}) + z\sqrt{q_{\mu\nu}} \tag{D.24}$$

$$x_\nu = \eta_\nu\sqrt{r - q_{\mu\nu}} + z\sqrt{q_{\mu\nu}} \sim \eta_\mu(1 - \frac{1}{2}q_{\mu\nu}) + z\sqrt{q_{\mu\nu}} \tag{D.25}$$

$$\tag{D.26}$$

where $\eta_\mu, \eta_\nu, z \sim \mathcal{N}(0,1)$. To calculate $f(0), f'(0)$, we can expand the nonlinearity,

$$f(q_{\mu\nu}) \equiv f(0) + f'(0)q_{\mu\nu} + o(q_{\mu\nu}^2) \tag{D.27}$$

$$= \mathbb{E}\left[\left(\sigma(\eta_\mu) + \sigma'(\eta_\mu)\left(-\frac{q}{2}\eta_\mu + z\sqrt{q}_{\mu\nu}\right) + \frac{\sigma''(\eta_\mu)}{2}\left(z^2 q_{\mu\nu}\right)\right)\right. \tag{D.28}$$

$$\left.\left(\sigma(\eta_\nu) + \sigma'(\eta_\nu)\left(-\frac{q}{2}\eta_\nu + z\sqrt{q}_{\mu\nu}\right) + \frac{\sigma''(\eta_\nu)}{2}\left(z^2 q_{\mu\nu}\right)\right)\right] \tag{D.29}$$

$$= \mathbb{E}_\eta\left[\sigma(\eta)\right]^2 + q_{\mu\nu}\left(\mathbb{E}[\sigma'(\eta)]^2 + \mathbb{E}[\sigma''(\eta)]\mathbb{E}[\sigma(\eta)] - \mathbb{E}[\sigma'(\eta)\eta]\mathbb{E}[\sigma(\eta)]\right) + o(q_{\mu\nu}^2) \tag{D.30}$$

$$\tag{D.31}$$

But since $\mathbb{E}[\sigma(\eta)\eta] = \mathbb{E}[\sigma'(\eta)]$, the two last terms cancel out and we are left with

$$f(0) = \mathbb{E}[\sigma(z)]^2, \qquad f'(0) = \mathbb{E}[\sigma(z)\eta]^2 \tag{D.32}$$

$$\mathbb{E}_{\boldsymbol{x}}\left[\sigma\left(\frac{\boldsymbol{x}\cdot\boldsymbol{\Theta}_\mu}{\sqrt{D}}\right)\sigma\left(\frac{\boldsymbol{x}\cdot\boldsymbol{\Theta}_\nu}{\sqrt{D}}\right)\right] = \mu_0^2 + \mu_1^2\frac{\boldsymbol{\Theta}_\mu\cdot\boldsymbol{\Theta}_\nu}{D} + \mu_\star^2\delta_{\mu\nu} \tag{D.33}$$

Anisotropic setup Now in the block setup,

$$\mathbb{E}\left[x_\mu^2\right] = \sum_i \sigma_{x,i}\phi_i \equiv r, \qquad \mathbb{E}\left[x_\mu x_\nu\right] = \sum_i \sigma_{x,i}\frac{\boldsymbol{\Theta}_\mu^i \cdot \boldsymbol{\Theta}_\nu^i}{D} = \sum_i \sigma_{x,i} q_{\mu\nu}^i \equiv q_{\mu\nu} \sim O\left(\frac{1}{\sqrt{D}}\right) \tag{D.34}$$

Therefore we can parametrize as follows,

$$x_\mu = \eta_\mu\sqrt{1-\frac{q_{\mu\nu}}{r}} + z\sqrt{\frac{q_{\mu\nu}}{r}} \sim \eta_\mu(1-\frac{1}{2}\frac{q_{\mu\nu}}{r}) + z\sqrt{\frac{q_{\mu\nu}}{r}} \tag{D.35}$$

$$x_\nu = \eta_\nu\sqrt{1-\frac{q_{\mu\nu}}{r}} + z\sqrt{\frac{q_{\mu\nu}}{r}} \sim \eta_\mu(1-\frac{1}{2}\frac{q_{\mu\nu}}{r}) + z\sqrt{\frac{q_{\mu\nu}}{r}} \tag{D.36}$$

$$\tag{D.37}$$

where $\eta_\mu, \eta_\nu, z \sim \mathcal{N}(0, r)$. As before,

$$f(q_{\mu\nu}) = \mathbb{E}_\eta\left[\sigma(\eta)\right]^2 + \frac{q_{\mu\nu}}{r}\left(\mathbb{E}[\sigma'(\eta)]^2\mathbb{E}[z^2] + \mathbb{E}[\sigma''(\eta)]\mathbb{E}[\sigma(\eta)]\mathbb{E}[z^2] - \mathbb{E}[\sigma'(\eta)\eta]\mathbb{E}[\sigma(\eta)]\right) + o(q_{\mu\nu}^2) \tag{D.38}$$

$$\tag{D.39}$$

Now since $\mathbb{E}[z^2] = r$ and $\mathbb{E}[\sigma(\eta)\eta] = r\mathbb{E}[\sigma'(\eta)]$, the two last terms do not cancel out like before and we have:

$$f(0) = \mathbb{E}[\sigma(z)]^2, \qquad f'(0) = \frac{1}{r}\mathbb{E}[\sigma(z)\eta]^2\mathbb{E}[z^2] = \frac{1}{r^2}\mathbb{E}[\sigma(z)z]^2 \tag{D.40}$$

Finally we have

$$\mathbb{E}_{\boldsymbol{x}}\left[\sigma\left(\frac{\boldsymbol{x}\cdot\boldsymbol{\Theta}_\mu}{\sqrt{D}}\right)\sigma\left(\frac{\boldsymbol{x}\cdot\boldsymbol{\Theta}_\nu}{\sqrt{D}}\right)\right] = \mu_0^2 + \mu_1^2\sum_i \sigma_{x,i}\frac{\boldsymbol{\Theta}_\mu^i\cdot\boldsymbol{\Theta}_\nu^i}{D} + \mu_\star^2\delta_{\mu\nu} \tag{D.41}$$

$$\mu_0 = \mathop{\mathbb{E}}_{z\sim\mathcal{N}(0,r)}[\sigma(z)], \quad \mu_1 = \frac{1}{r}\mathop{\mathbb{E}}_{z\sim\mathcal{N}(0,r)}[z\sigma(z)]], \quad \mu_\star = \sqrt{\mathop{\mathbb{E}}_{z\sim\mathcal{N}(0,r)}[\sigma(z)^2] - \mu_0^2 - r\mu_1^2} \tag{D.42}$$

$$\tag{D.43}$$

D.2.3 Gibbs formulation of the problem

To obtain the test error, we need to find the typical value of the order parameters m_s, q_s, q_a. To do so, we formulate the optimization problem for the second layer weights as a Gibbs measure over the weights as in [122]:

$$\mu_\beta\left(\boldsymbol{a} \mid \{\boldsymbol{x}^\mu, y^\mu\}\right) = \frac{1}{\mathcal{Z}_\beta}e^{-\beta\left[\sum_{\mu=1}^N \ell(y^\mu, \boldsymbol{x}^\mu\cdot\boldsymbol{a}) + \frac{\lambda}{2}\|\boldsymbol{a}\|_2^2\right]} = \frac{1}{\mathcal{Z}_\beta}\underbrace{\prod_{\mu=1}^N e^{-\beta\ell(y^\mu,\boldsymbol{x}^\mu\cdot\boldsymbol{a})}}_{\equiv P_y(\boldsymbol{y}|\boldsymbol{a}\cdot\boldsymbol{x}^\mu)}\underbrace{\prod_{i=1}^P e^{-\frac{\beta\lambda}{2}w_i^2}}_{\equiv P_a(\boldsymbol{a})} \tag{D.44}$$

Of key interest is the behavior of the free energy density, which is self-averaging in the high-dimensional limit and whose minimization gives the optimal value of the overlaps:

$$f_\beta = -\lim_{P\to\infty}\frac{1}{P}\mathbb{E}_{\{\boldsymbol{x}^\mu, y^\mu\}}\log\mathcal{Z}_\beta \tag{D.45}$$

To calculate the latter, we use the Replica Trick from statistical physics:

$$\mathbb{E}_{\{\boldsymbol{x}^\mu,y^\mu\}}\log\mathcal{Z}_\beta=\lim_{r\to 0}\frac{D}{Dr}\mathbb{E}_{\{\boldsymbol{x}^\mu,y^\mu\}}\mathcal{Z}_\beta^r \tag{D.46}$$

The right hand side can be written in terms of the *order parameters* :

$$\mathbb{E}_{\{\boldsymbol{x}^\mu,y^\mu\}}\mathcal{Z}_\beta^r=\int\frac{\mathrm{d}\rho\,\mathrm{d}\hat{\rho}}{2\pi}\int\prod_{a=1}^r\frac{\mathrm{d}m_s^a\,\mathrm{d}\hat{m}_s^a}{2\pi}\int\prod_{1\le a\le b\le r}\frac{\mathrm{d}q_s^{ab}\,\mathrm{d}\hat{q}_s^{ab}}{2\pi}\frac{\mathrm{d}q_a^{ab}\,\mathrm{d}\hat{q}_a^{ab}}{2\pi}e^{P\Phi^{(r)}} \tag{D.47}$$

$$\Phi^{(r)}=-\gamma\rho\hat{\rho}-\gamma\sum_{a=1}^r\sum_i m_{s,i}^a\hat{m}_{s,i}^a-\sum_{1\le a\le b\le r}\left(\gamma\sum_i q_{s,i}^{ab}\hat{q}_{s,i}^{ab}+q_a\hat{q}_a\right) \tag{D.48}$$

$$+\alpha\Psi_y^{(r)}\left(\rho,m_s^a,q_s^{ab},q_a^{ab}\right)+\Psi_a^{(r)}\left(\hat{\rho},\hat{m}_s^a,\hat{q}_s^{ab},\hat{q}_a^{ab}\right) \tag{D.49}$$

Where we introduced and *energetic* part Ψ_y which corresponds to the likelihood of the order parameters taking a certain value based on the data, and an *entropic* part Ψ_a which corresponds to the volume compatible with those order parameters :

$$\begin{aligned}\Psi_y^{(r)}&=\log\int\mathrm{d}y\int\mathrm{d}\nu P_y^0(y\mid\nu)\int\textstyle\prod_{a=1}^r\left[\mathrm{d}\lambda^a P_y\left(y\mid\lambda^a\right)\right]P\left(\nu,\{\lambda^a\}\right)\\ \Psi_a^{(r)}&=\tfrac{1}{P}\log\int\mathrm{d}\boldsymbol{\beta}P_\beta\left(\boldsymbol{\beta}\right)e^{-\hat{\rho}\|\beta\|^2}\int\textstyle\prod_{a=1}^r\mathrm{d}\boldsymbol{a}^a P_a\left(\boldsymbol{a}^a\right)e^{\sum_{1\le a\le b\le r}\left[\hat{q}_a^{ab}\boldsymbol{a}^a\cdot\boldsymbol{a}^b+\sum_i\hat{q}_{s,i}^{ab}\boldsymbol{s}^a|_i\cdot\boldsymbol{s}^b|_i\right]-\sum_{a=1}^r\sum_i\hat{m}_{s,i}^a\boldsymbol{s}^a|_i\cdot\boldsymbol{\beta}|_i}\end{aligned} \tag{D.50}$$

The calculation of Ψ_y is exactly the same as in [122] : we defer the reader to the latter for details. As for Ψ_a, considering that $P_a(\boldsymbol{a})=e^{-\frac{\lambda}{2}\|\boldsymbol{a}\|^2}$ we have (denoting the block indices as i):

$$\Psi_a=\lim_{P\to\infty}\mathbb{E}_{\boldsymbol{\beta},\xi,\eta}\left[\frac{1}{2}\frac{\eta^2\hat{q}_a}{\beta\lambda+\hat{V}_a}-\frac{1}{2}\log\left(\beta\lambda+\hat{V}_a\right)-\frac{1}{2P}\operatorname{tr}\log\left(\mathrm{I}_D+\hat{V}_s\Sigma\right)-\frac{1}{2P}\boldsymbol{\mu}^\top\Sigma^{-1}\boldsymbol{\mu}\right. \tag{D.51}$$

$$\left.+\frac{1}{2P}\left(\boldsymbol{b}+\Sigma^{-1}\boldsymbol{\mu}\right)^\top\hat{V}_s^{-1}\left(\boldsymbol{b}+\Sigma^{-1}\boldsymbol{\mu}\right)-\frac{1}{2P}\left(\boldsymbol{b}+\Sigma^{-1}\mu\right)^\top\hat{V}_s^{-1}\left(\mathrm{I}_D+\hat{V}_s\Sigma\right)^{-1}\left(\boldsymbol{b}+\Sigma^{-1}\boldsymbol{\mu}\right)\right] \tag{D.52}$$

where $\hat{V}_s$ is a block diagonal matrix where the block diagonals read $(\hat{V}_{s,1},\hat{V}_{s,2})$, and

$$\begin{aligned}\boldsymbol{b}&=\left(\sqrt{\hat{q}_{s,1}}\xi_1\mathbf{1}_{\phi_1 D}+\hat{m}_{s,1}\boldsymbol{\beta}_1,\sqrt{\hat{q}_{s,2}}\xi_2\mathbf{1}_{\phi_2 D}+\hat{m}_{s,2}\boldsymbol{\beta}_2\right)\\ \boldsymbol{\mu}&=\frac{\sqrt{\hat{q}_a}\eta}{\beta\lambda+\hat{W}_a}\frac{\boldsymbol{\Theta}\mathbf{1}_P}{\sqrt{P}}\\ \Sigma&=\frac{1}{\beta\lambda+\hat{V}_a}\frac{\Theta\Theta^\top}{P}\end{aligned}$$

and λ here is the coefficient in the ℓ_2 regularization. Then, we have

$$\begin{aligned}\mathbb{E}_\eta\left[\boldsymbol{\mu}^\top\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}\right]&=\tfrac{1}{P}\tfrac{\hat{q}_a}{\left(\beta\lambda+\hat{V}_a\right)^2}\left(\boldsymbol{\Theta}\mathbf{1}_P\right)^\top\boldsymbol{\Sigma}^{-1}\left(\boldsymbol{\Theta}\mathbf{1}_P\right)=D\tfrac{\hat{q}_a}{\beta\lambda+\hat{V}_a}\\ \mathbb{E}_{\eta,\xi,\boldsymbol{\beta}}\left\|\boldsymbol{b}+\Sigma^{-1}\boldsymbol{\mu}\right\|^2&=\textstyle\sum_i\phi_i D\left(\hat{m}_{s,i}^2+\hat{q}_{s,i}\right)+\tfrac{1}{P}\hat{q}_a\operatorname{tr}(\Theta\Theta^\top)^{-1}\\ \mathbb{E}_{\eta,\xi,\boldsymbol{\beta}}\left(\boldsymbol{b}+\Sigma^{-1}\boldsymbol{\mu}\right)^\top\left(\mathrm{I}_D+\hat{V}_s\Sigma\right)^{-1}\left(\boldsymbol{b}+\Sigma^{-1}\boldsymbol{\mu}\right)&=\tfrac{1}{P}\hat{q}_a\operatorname{tr}\left[\Theta\Theta^\top\left(\mathrm{I}_D+\hat{V}_s\Sigma\right)^{-1}\right]\\ &+\textstyle\sum_i\left(\hat{m}_{s,i}^2+\hat{q}_{s,i}\right)\operatorname{tr}\left(\mathrm{I}_D+\hat{V}_{s_i}\Sigma_i\right)^{-1}\end{aligned} \tag{D.53}$$

$$\Psi_a = -\frac{1}{2}\log\left(\beta\lambda + \hat{V}_a\right) - \frac{1}{2}\lim_{P\to\infty}\frac{1}{P}\operatorname{tr}\log\left(\mathrm{I}_D + \frac{\hat{V}_s}{\beta\lambda+\hat{V}_a}\frac{\Theta\Theta^\top}{P}\right) \tag{D.54}$$

$$+\sum_i \frac{\hat{m}_{s,i}^2 + \hat{q}_{s,i}}{2\hat{V}_{s,i}}\left[\phi_i\gamma\right] - \lim_{P\to\infty}\frac{1}{P}\operatorname{tr}\left((\hat{m}_s^2+\hat{q}_s)\left(\mathrm{I}_D + \frac{\hat{V}_s}{\beta\lambda+\hat{V}_a}\frac{\Theta\Theta^\top}{P}\right)^{-1}\right) \tag{D.55}$$

$$+\frac{1}{2}\frac{\hat{q}_a}{\beta\lambda+\hat{V}_a}\left[1-\gamma\right] + \frac{\hat{q}_a}{2}\lim_{P\to\infty}\frac{1}{P}\left[\operatorname{tr}\left(\hat{V}_s^{-1}(\Theta\Theta^\top)^{-1}\right) - \operatorname{tr}\left((\hat{V}_s\Theta\Theta^\top)^{-1}\left(\mathrm{I}_D + \frac{\hat{V}_s}{\beta\lambda+\hat{V}_a}\frac{\Theta\Theta^\top}{P}\right)^{-1}\right)\right] \tag{D.56}$$

where $(\hat{m}_s^2+\hat{q}_s)$ and $\hat{V}_s$ in the trace operator are to be understood as diagonal matrices here. Using the following simplification

$$\operatorname{tr}\left(V_s^{-1}(\Theta\Theta^\top)^{-1}\right) - \operatorname{tr}\left((V_s\Theta\Theta^\top)^{-1}\left(\mathrm{I}_D + \frac{\hat{V}_s}{\beta\lambda+\hat{V}_a}\frac{\Theta\Theta^\top}{P}\right)^{-1}\right) \tag{D.57}$$

$$= \frac{1}{P(\beta\lambda+\hat{V}_a)}\operatorname{tr}\left(\mathrm{I}_d + \frac{\hat{V}_s}{\beta\lambda+\hat{V}_a}\frac{\Theta\Theta^\top}{P}\right), \tag{D.58}$$

we finally obtain

$$\Psi_a = -\frac{1}{2}\log\left(\beta\lambda + \hat{V}_a\right) - \frac{1}{2}\lim_{P\to\infty}\frac{1}{P}\operatorname{tr}\log\left(\mathrm{I}_D + \frac{\hat{V}_s}{\beta\lambda+\hat{V}_a}\frac{\Theta\Theta^\top}{P}\right) \tag{D.59}$$

$$+\sum_i \frac{\hat{m}_{s,i}^2 + \hat{q}_{s,i}}{2\hat{V}_{s,i}}\left[\phi_i\gamma\right] - \lim_{P\to\infty}\frac{1}{P}\operatorname{tr}\left((\hat{m}_s^2+\hat{q}_s)\left(\mathrm{I}_D + \frac{\hat{V}_s}{\beta\lambda+\hat{V}_a}\frac{\Theta\Theta^\top}{P}\right)^{-1}\right) \tag{D.60}$$

$$+\frac{1}{2}\frac{\hat{q}_a}{\beta\lambda+\hat{V}_a}\left[1-\gamma+\lim_{P\to\infty}\frac{1}{P}\operatorname{tr}\left(\left(\mathrm{I}_D + \frac{\hat{V}_s}{\beta\lambda+\hat{V}_a}\frac{\Theta\Theta^\top}{P}\right)^{-1}\right)\right] \tag{D.61}$$

.

Define $M = \frac{1}{P}\hat{V}_s\boldsymbol{\Theta}\boldsymbol{\Theta}^\top$. Ψ_a involves the following term :

$$\frac{1}{P}\operatorname{tr}\left(\mathrm{I}_D + \frac{\hat{V}_s}{\beta\lambda+\hat{V}_a}M\right)^{-1} = \gamma(\beta\lambda+\hat{V}_a)g(-\beta\lambda+\hat{V}_a) \tag{D.62}$$

Where g is the Stieljes transform of M : $g(z) = \frac{1}{D}\operatorname{Tr}(z-M)^{-1}$.

D.2.4 Some random matix theory for block matrices

To calculate the desired Stieljes transform, we specialize to the case of Gaussian random feature matrices and use again tools from Statistical Physics.

Isotropic setup We first consider the isotropic setup where $\hat{V}_s$ is a scalar. Then we have

$$g(z) = \frac{1}{D}\operatorname{Tr}(z-M)^{-1} \tag{D.63}$$

$$= -\frac{1}{D}\frac{d}{dz}\operatorname{Tr}\log(z-M) \tag{D.64}$$

$$= -\frac{1}{D}\frac{d}{dz}\log\det(z-M) \tag{D.65}$$

$$= -\frac{2}{D}\frac{d}{dz}\left\langle \log\int dy e^{-\frac{1}{2}y(z-M)y^\top}\right\rangle \tag{D.66}$$

where $\langle . \rangle$ stands for the average over disorder, here the matrix $\mathbf{\Theta}$. Then we use the replica trick,

$$\langle \log Z\rangle \to_{n\to 0} \frac{1}{n}\log\langle Z^n\rangle \tag{D.67}$$

Therefore we need to calculate Z^n:

$$\langle Z^n\rangle = \int\prod_{a=1}^{n} d\vec{y}_a e^{-\frac{1}{2}z\sum_a \vec{y}_a^2}\int d\mathbf{\Theta} e^{-\frac{1}{2}\Theta \mathrm{L}\Theta^\top} \tag{D.68}$$

$$= \int\prod_{a=1}^{n} d\vec{y}_a e^{-\frac{1}{2}z\sum_a \vec{y}_a^2}(\det L)^{-P/2} \tag{D.69}$$

where $L = \mathbb{I}_d - \frac{\gamma\hat{V}_s}{D}\sum_a y_a\vec{y}_a^\top$. Here we decompose the vectors $\vec{y}$ into two parts. Then we use that $\det L = \det\tilde{L}$, where $\tilde{L}_{ab} = \delta_{ab} - \frac{\gamma\hat{V}_s}{D}\vec{y}_a\cdot\vec{y}_b$. Then,

$$\langle Z^n\rangle = \int\prod_a d\vec{y}_a e^{\frac{1}{2}\sum_a \vec{y}_a\cdot\vec{y}_a}\det\left[\mathbb{I}_d - \frac{\gamma\hat{V}_s}{D}Y^\top Y\right]^{-\frac{P}{2}} \tag{D.70}$$

where $Y\in\mathbb{R}^{d\times n}$ with $Y_{ia} = y_i^a$. We introduce $1 = \int dQ_{ab}\delta(dQ_{ab} - \vec{y}_a\cdot\vec{y}_b)$, and use the Fourier representation of the delta function, yielding

$$\langle Z^n\rangle = \int dQ e^{-\frac{D}{2}z\operatorname{Tr}Q}\left(\det\left[\mathbb{I}-\hat{V}_s Q\right]\right)^{-\frac{P}{2}}\int d\hat{Q}_{ab}e^{\sum_{ab}dQ_{ab}\hat{Q}_{ab} - \hat{Q}_{ab}\vec{y}_a\vec{y}_b} \tag{D.71}$$

$$= \int dQ d\hat{Q} e^{-dS[Q,\hat{Q}]} \tag{D.72}$$

where

$$S[Q,\hat{Q}] = \frac{1}{2}z\operatorname{Tr}Q + \frac{1}{2\gamma}\log\det\left(1-\hat{V}_s Q\right) + \frac{1}{2}\log\det\left(2\hat{Q}\right) - \operatorname{Tr}\left(Q\hat{Q}\right) \tag{D.73}$$

A saddle-point on $\hat{Q}$ gives $Q = (2\hat{Q})^{-1}$. Therefore we can replace in S,

$$S[Q] = \frac{1}{2}z\operatorname{Tr}Q + \frac{1}{2\gamma}\log\det\left(1-\hat{V}_s Q\right) - \frac{1}{2}\log\det\left(Q\right) \tag{D.74}$$

In the RS ansatz $Q_{ab} = q\delta_{ab}$, this yields

$$S[q]/n = \frac{1}{2}zq + \frac{1}{2\gamma}\log\left(1-\hat{V}_s q\right) - \frac{1}{2}\log q \tag{D.75}$$

Now we may apply a saddle point method to write

$$\langle Z^n \rangle = e^{-dS[q^\star]} \tag{D.76}$$

where $q^\star$ minimizes the action, i.e. $\frac{dS}{dq}|_{q^\star} = 0$:

$$z - \frac{\hat{V}_s}{1 - \hat{V}_s q^\star} - \frac{1}{q^\star} = 0 \tag{D.77}$$

Therefore,

$$g(z) = -\frac{2}{D}\frac{d}{dz}(-DS[q^\star]) = q^\star(z) \tag{D.78}$$

Anisotropic setup Now we consider that V is a black diagonal matrix, with blocks of size $\phi_i d$ with values $\hat{V}_{s,i}$. We need to adapt the calculation by decomposing the auxiliary fields y along the different blocks, then define separately the overlaps of the blocks q_i. Then we obtain the following action:

$$S[\{q_i\}]/n = \frac{1}{2} z \sum_i \phi_i q_i + \frac{1}{2\gamma} \log\left(1 - \sum_i \phi_i \hat{V}_{s,i} q_i\right) - \sum_i \frac{\phi_i}{2} \log(q_i) \tag{D.79}$$

To obtain the Stieljes transform, we need to solve a system of coupled equations :

$$g(z) = \sum_i \phi_i q_i^\star \tag{D.80}$$

$$\phi_i z \Omega q_i^\star - \phi_i \hat{V}_i q_i^\star - \phi_i \Omega = 0 \tag{D.81}$$

$$\Omega = 1 - \sum_i \phi_i \hat{V}_{s,i} q_i^\star \tag{D.82}$$

We therefore conclude that

$$\lim_{P\to\infty} \frac{1}{P} \operatorname{tr}\left(\left(\mathrm{I}_D + \frac{\hat{V}_s}{\beta\lambda + \hat{V}_a}\frac{\Theta\Theta^\top}{P}\right)^{-1}\right) = \gamma(\beta\lambda + \hat{V}_a)\sum_i \phi_i q_i^\star \tag{D.83}$$

$$\lim_{P\to\infty} \frac{1}{P} \operatorname{tr}\left((\hat{m}_s^2 + \hat{q}_s)\left(\mathrm{I}_D + \frac{\hat{V}_s}{\beta\lambda + \hat{V}_a}\frac{\Theta\Theta^\top}{P}\right)^{-1}\right) = \gamma(\beta\lambda + \hat{V}_a)\sum_i (\hat{m}_{s,i}^2 + \hat{q}_{s,i})\phi_i q_i^\star \tag{D.84}$$

D.2.5 Obtaining the saddle-point equations

The saddle-point equations of [122] become the following in the anisotropic setup :

$$\begin{cases} \hat{r}_{s,i} = -2\sigma_{x,i}\frac{\alpha}{\gamma}\partial_{r_{s,i}}\Psi_y(R,Q,M) & r_{s,i} = -\frac{2}{\gamma}\partial_{\hat{r}_{s,i}}\Psi_a\left(\hat{r}_{s,i},\hat{q}_{s,i},\hat{m}_{s,i},\hat{r}_a,\hat{q}_a\right) \\ \hat{q}_{s,i} = -2\sigma_{x,i}\frac{\alpha}{\gamma}\partial_{q_{s,i}}\Psi_y(R,Q,M) & q_{s,i} = -\frac{2}{\gamma}\partial_{\hat{q}_{s,i}}\Psi_a\left(\hat{r}_{s,i},\hat{q}_{s,i},\hat{m}_{s,i},\hat{r}_a,\hat{q}_a\right) \\ \hat{m}_{s,i} = \sigma_{x,i}\frac{\alpha}{\gamma}\partial_{m_{s,i}}\Psi_y(R,Q,M) & m_{s,i} = \frac{1}{\gamma}\partial_{\hat{m}_{s,i}}\Psi_a\left(\hat{r}_{s,i},\hat{q}_{s,i},\hat{m}_{s,i},\hat{r}_a,\hat{q}_a\right) \\ \hat{r}_a = -2\alpha\partial_{r_a}\Psi_y(R,Q,M) & r_a = -2\partial_{\hat{r}_a}\Psi_a\left(\hat{r}_{s,i},\hat{q}_{s,i},\hat{m}_{s,i},\hat{r}_a,\hat{q}_a\right) \\ \hat{q}_a = -2\alpha\partial_{q_a}\Psi_y(R,Q,M) & q_a = -2\partial_{\hat{q}_a}\Psi_a\left(\hat{r}_{s,i},\hat{q}_{s,i},\hat{m}_{s,i},\hat{r}_a,\hat{q}_a\right) \end{cases} \tag{D.85}$$

The saddle point equations corresponding to Ψ_a do not depend on the learning task, and can be simplified in full generality to the following set of equations :

$$\begin{cases} V_{s,i} = \frac{1}{\hat{V}_{s,i}} \left(\phi_i - z_i g_\mu(-z_i)\right) \\ q_{s,i} = \frac{\sigma_{\beta,i}\hat{m}_{s,i}^2 + \hat{q}_{s,i}}{\hat{V}_{s,i}^2} \left[\phi_i - 2z_i g_\mu(-z_i) + z_i^2 g'_\mu(-z_i)\right] - \frac{\hat{q}_a}{\left(\beta\lambda + \hat{V}_a\right)\hat{V}_{s,i}} \left[-z_i g_\mu(-z_i) + z_i^2 g'_\mu(-z_i)\right] \\ m_{s,i} = \frac{\sigma_{\beta,i}\hat{m}_{s,i}}{\hat{V}_{s,i}} \left(\phi_i - z_i g_\mu(-z_i)\right) \\ V_a = \sum_i \frac{\gamma}{\beta\lambda + \hat{V}_a} \left[\frac{1}{\gamma} - 1 + z_i g_\mu(-z_i)\right] \\ q_a = \sum_i \frac{\sigma_{\beta,i}\hat{m}_s^2 + \hat{q}_{s,i}}{\left(\beta\lambda + \hat{V}_a\right)\hat{V}_{s,i}} \left[-z_i g_\mu(-z_i) + z_i^2 g'_\mu(-z_i)\right] + \gamma \frac{\hat{q}_a}{\left(\beta\lambda + \hat{V}_a\right)^2} \left[\frac{1}{\gamma} - 1 + z_i^2 g'_\mu(-z_i)\right] \end{cases} \tag{D.86}$$

As for the saddle point equations corresponding to Ψ_y, they depend on the learning task. We can solve analytically for the two setups below.
The solution of the saddle-point equations allow to obtain the order parameter values that determine the covariance of ν and λ (for one given replica, say $a = 1$) and hence to compute the test error as explained at the beginning of the appendix.

Square loss regression Now we specialize to our mean-square regression case study where the teacher is a Gaussian additive channel :

$$\mathcal{P}(x|y) = \frac{1}{\sqrt{2\pi\Delta}} e^{-\frac{(x-y)^2}{2\Delta}} \tag{D.87}$$

$$\ell(y, x) = \frac{1}{2}(x - y)^2 \tag{D.88}$$

In this case, the saddle-point equations simplify to :

$$\begin{cases} \hat{V}_{s,i}^\infty = \sigma_{x,i}\frac{\alpha}{\gamma}\frac{\mu_{1,i}^2}{1+V^\infty} \\ \hat{q}_{s,i}^0 = \sigma_{x,i}\alpha\mu_{1,i}^2 \frac{\phi_i\sigma_{\beta,i} + \Delta + Q^\infty - 2M^\infty}{\gamma(1+V^\infty)^2} \\ \hat{m}_{s,i} = \sigma_{x,i}\frac{\alpha}{\gamma}\frac{\mu_{1,i}}{1+V^\infty} \\ \hat{V}_a^\infty = \frac{\alpha\sum_i \phi_i\mu_{\star,i}^2}{1+V^\infty} \\ \hat{q}_a^\infty = \alpha\sum_i \phi_i\mu_{\star,i}^2 \frac{1+\Delta+Q^\infty-2M^\infty}{(1+V^\infty)^2} \end{cases} \tag{D.89}$$

Square loss classification Next we examine the classification setup where the teacher gives binary labels with a sign flip probability of Δ, and the student learns them through the mean-square loss:

$$\mathcal{P}(x|y) = (1-\Delta)\delta(x - sign(y)) + \Delta\delta(x + sign(y)), \quad \Delta \in [0, 1] \tag{D.90}$$

$$\ell(y, x) = \frac{1}{2}(x - y)^2 \tag{D.91}$$

In this case, the equations simplify to :

$$\begin{cases} \hat{V}_{s,i}^{\infty} = \sigma_{x,i}\frac{\alpha}{\gamma}\frac{\mu_{1,i}^2}{1+V^{\infty}} \\ \hat{q}_{s,i}^{0} = \sigma_{x,i}\alpha\mu_{1,i}^2\frac{\phi_i\sigma_{\beta,i}+Q^{\infty}-2\frac{(1-2\Delta)\sqrt{2}}{\sqrt{\pi}}M^{\infty}}{\gamma(1+V^{\infty})^2} \\ \hat{m}_{s,i} = \sigma_{x,i}\frac{\alpha}{\gamma}\frac{\mu_{1,i}}{1+V^{\infty}} \\ \hat{V}_{a}^{\infty} = \frac{\alpha\sum_i\phi_i\mu_{\star,i}^2}{1+V^{\infty}} \\ \hat{q}_{a}^{\infty} = \alpha\sum_i\phi_i\mu_{\star,i}^2\frac{1+Q^{\infty}-2\frac{(1-2\Delta)\sqrt{2}}{\sqrt{\pi}}M^{\infty}}{(1+V^{\infty})^2} \end{cases} \tag{D.92}$$

Cross-entropy loss classification For general loss functions such as the cross-entropy loss, $\ell(x,y) = \log(1+e^{-xy})$, the saddle-point equations for Ψ_y do not simplify and one needs to evaluate the integral over ξ numerically. This case is outside the scope of this paper.

D.2.6 Training loss

To calculate the training loss, we remove the regularization term:

$$\epsilon_t = \frac{1}{N}\mathbb{E}_{\{\boldsymbol{x}^{\mu},y^{\mu}\}}\left[\sum_{\mu=1}^{N}\ell\left(y^{\mu},\boldsymbol{x}^{\mu}\cdot\hat{\boldsymbol{w}}\right)\right] \tag{D.93}$$

As explained in [122], the latter can be written as

$$\epsilon_t = \mathbb{E}_{\xi}\left[\int_{\mathbb{R}}\mathrm{d}y\mathcal{Z}_y^0\left(y,\omega_0,V_0\right)\ell\left(y,\eta\left(y,\omega_1\right)\right)\right], \quad \xi\sim\mathcal{N}(0,1) \tag{D.94}$$

where $\mathcal{P}$ is the teacher channel defined in (D.5), and we have:

$$\omega_0 = M/\sqrt{Q}\xi \tag{D.95}$$
$$V_0 = \rho - M^2/Q \tag{D.96}$$
$$\omega_1 = \sqrt{Q}\xi \tag{D.97}$$
$$\eta(y,\omega) = \arg\min_x\frac{(x-\omega)^2}{2V} + \ell(y,x) \tag{D.98}$$
$$\mathcal{Z}_y^0\left(y,\omega\right) = \int\frac{\mathrm{d}x}{\sqrt{2\pi V_0}}e^{-\frac{1}{2V_0}(x-\omega)^2}\mathcal{P}(y|x) \tag{D.99}$$

Square loss regression Now we specialize to our regression case study where the teacher is a Gaussian additive channel $\mathcal{P}(x|y) = \frac{1}{\sqrt{2\pi\Delta}}e^{-\frac{(x-y)^2}{2\Delta}}$ and the loss is $\ell(y,x) = \frac{1}{2}(x-y)^2$. These assumptions imply

$$\mathcal{Z}_y^0\left(y,\omega\right) = \frac{1}{\sqrt{2\pi(V_0+\Delta)}}e^{-\frac{(y-\omega)^2}{2(V_0+\Delta)}}, \tag{D.100}$$
$$\eta(y,\omega) = \frac{\omega+Vy}{1+V}, \tag{D.101}$$

which yields the simple formula for the training error

$$\mathcal{L} = \frac{1}{2(1+V)^2}\mathbb{E}_\xi\left[\int \mathrm{d}y \mathcal{Z}_y^0(y,\omega)(y-\omega_1)^2\right] \tag{D.102}$$

$$= \frac{1}{2(1+V)^2}\mathbb{E}_\xi\left[V_0 + \Delta + (\omega_0-\omega_1)^2\right] \tag{D.103}$$

$$= \frac{1}{2(1+V)^2}(\rho + Q - 2M + \Delta) \tag{D.104}$$

$$= \frac{\epsilon_g + \Delta}{(1+V)^2}. \tag{D.105}$$

Square loss classification Next we specialize to our classification case study where the teacher is a Gaussian additive channel $\mathcal{P}(x|y) = (1-\Delta)\delta(x - sign(y)) + \Delta\delta(x + sign(y))$ and the loss is $\ell(y,x) = \frac{1}{2}(x-y)^2$. These assumptions imply

$$\mathcal{Z}_y^0(y,\omega) = \frac{1}{\sqrt{2\pi V_0}}\left((1-\Delta)e^{-\frac{(y-\omega)^2}{2V_0}} + \Delta e^{-\frac{(-y-\omega)^2}{2V_0}}\right), \tag{D.106}$$

$$\eta(y,\omega) = \frac{\omega + Vy}{1+V}, \tag{D.107}$$

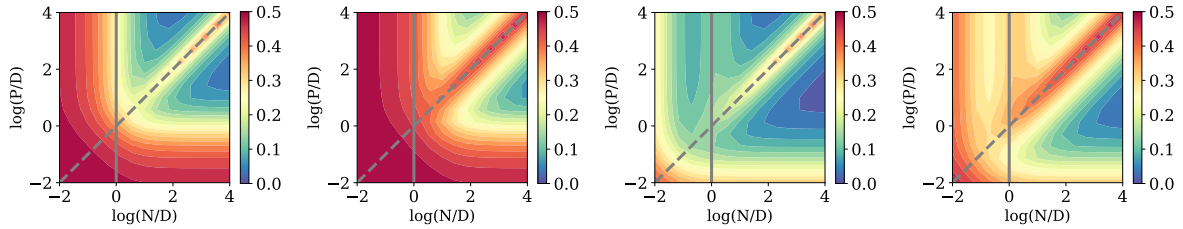
which yields the simple formula for the training error

$$\mathcal{L} = \frac{1}{2(1+V)^2}\mathbb{E}_\xi\left[V_0 + (1-\Delta)(\omega_0-\omega_1)^2 + \Delta(\omega_0+\omega_1)^2\right] \tag{D.108}$$

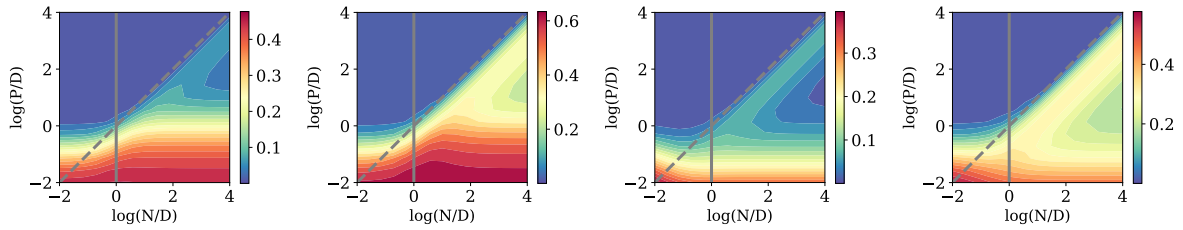
$$= \frac{1}{2(1+V)^2}(\rho + Q - 2(1-2\Delta)M) \tag{D.109}$$

$$\tag{D.110}$$

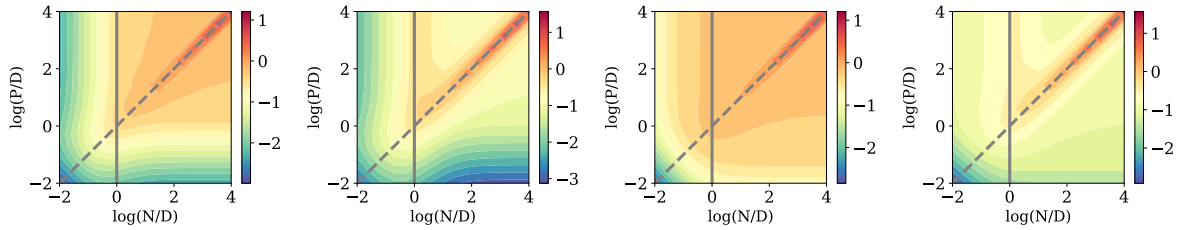
This expression is similar to the one obtained in the regression setup, except for the role of the noise, which reflects label flipping instead of additive noise. As a sanity check, note that flipping all the labels, i.e. $\Delta = 1$, is equivalent to the transformation $M \to -M$, as one could expect.



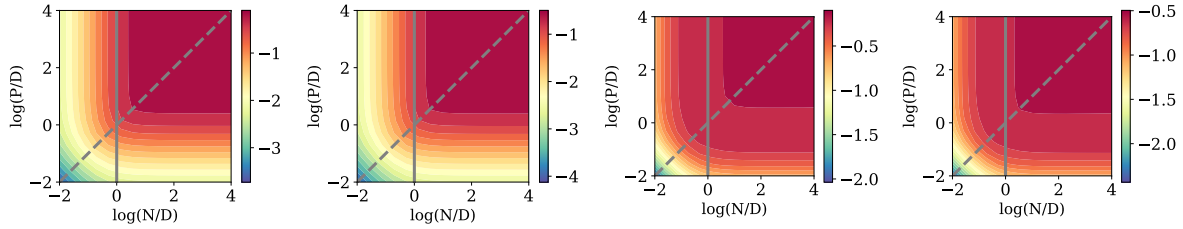
(a) Isotropic, noiseless (b) Isotropic, $\Delta = 0.4$ (c) Aligned, $\Delta = 0$ (d) Aligned, $\Delta = 0.4$



(e) Isotropic, noiseless (f) Isotropic, $\Delta = 0.4$ (g) Aligned, $\Delta = 0$ (h) Aligned, $\Delta = 0.4$

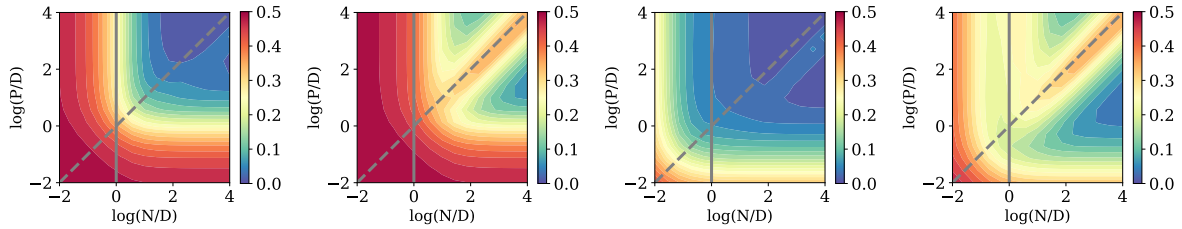


(i) Isotropic, noiseless (j) Isotropic, $\Delta = 0.4$ (k) Aligned, $\Delta = 0$ (l) Aligned, $\Delta = 0.4$

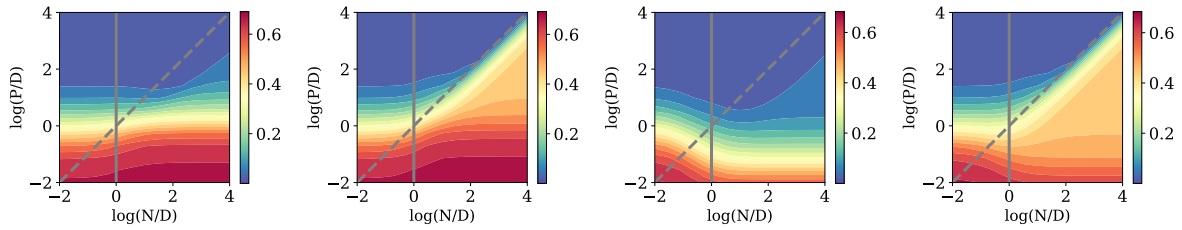


(m) Isotropic, noiseless (n) Isotropic, $\Delta = 0.4$ (o) Aligned, $\Delta = 0$ (p) Aligned, $\Delta = 0.4$

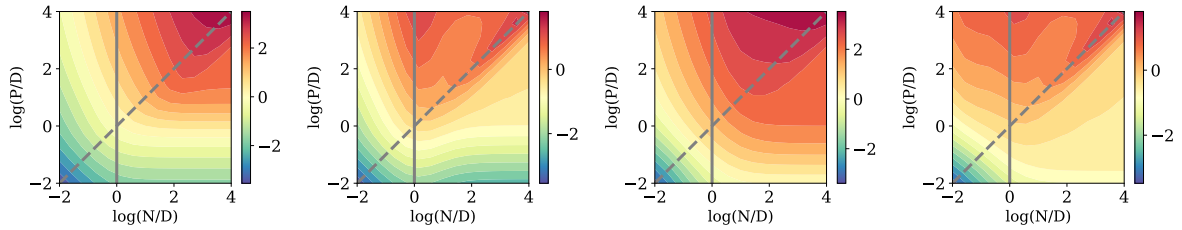
Figure D.1: **Square loss phase spaces.** We studied the classification task for $\sigma = \mathrm{ReLU}$, $\lambda = 0.1$. In the anisotropic phase spaces we set $\phi_1 = 0.01$, $r_x = 100$, $r_\beta = 100$. *First row:* test error. *Second row:* train error. *Third row:* Q. *Fourth row:* M. The solid and dashed grey lines represent the $N = D$ and $N = P$ lines, where one can find overfitting peaks [119]. For Q and M, the colormaps are logarithmic.



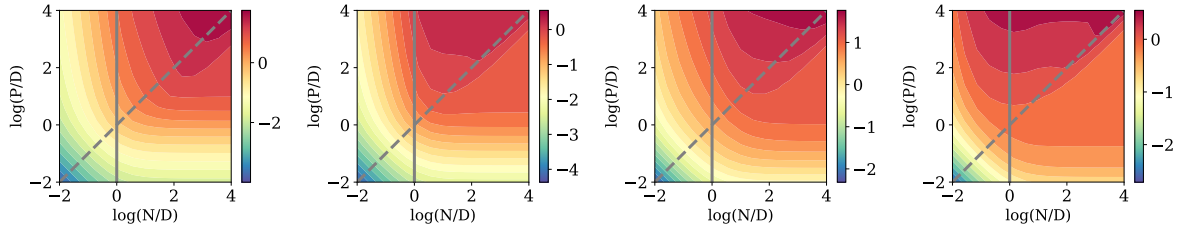
(a) Isotropic, noiseless (b) Isotropic, $\Delta = 0.4$ (c) Aligned, $\Delta = 0$ (d) Aligned, $\Delta = 0.4$



(e) Isotropic, noiseless (f) Isotropic, $\Delta = 0.4$ (g) Aligned, $\Delta = 0$ (h) Aligned, $\Delta = 0.4$

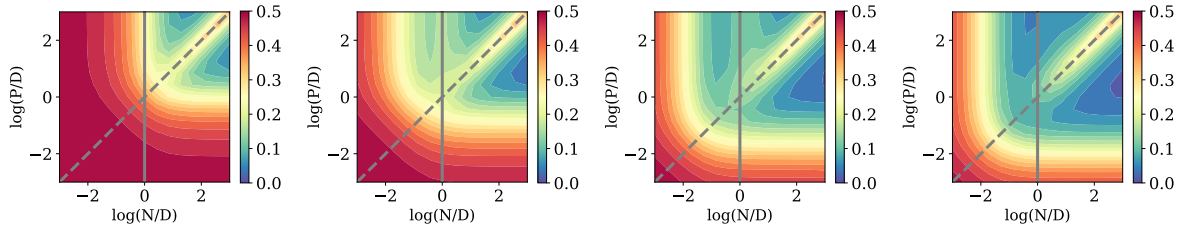


(i) Isotropic, noiseless (j) Isotropic, $\Delta = 0.4$ (k) Aligned, $\Delta = 0$ (l) Aligned, $\Delta = 0.4$



(m) Isotropic, noiseless (n) Isotropic, $\Delta = 0.4$ (o) Aligned, $\Delta = 0$ (p) Aligned, $\Delta = 0.4$

Figure D.2: **Logistic loss phase spaces.** We studied the classification task for $\sigma = \mathrm{ReLU}$, $\lambda = 0.1$. In the Aligned phase spaces we set $\phi_1 = 0.01$, $r_x = 100$, $r_\beta = 100$. *First row:* test error. *Second row:* train error. *Third row:* Q. *Fourth row:* M. The solid and dashed grey lines represent the $N = D$ and $N = P$ lines, where one can find overfitting peaks [119]. For Q and M, the colormaps are logarithmic.



(a) $r_x = 1$ (b) $r_x = 10$ (c) $r_x = 100$ (d) $r_x = 1000$

Figure D.3: Increasing the saliency of the first subspace from 1 to ∞, the asymmetry forms then vanishes as the data becomes effectively isotropic in smaller dimension. We study the classification task for $\sigma = \mathrm{ReLU}$, $\lambda = 0.1$, $\phi_1 = 0.01$, $r_\beta = 1000$ and $\Delta = 0$.

Appendix E

Finding the Needle in the Haystack: When do Convolutional Constraints help?

E.1 Visualizing the embedding

In Fig. E.1, we provide an illustration of the mapping from CNN to eFCN. Denoting as k, s, p the filter size, stride and padding of the convolution, we have the following:

$$\begin{aligned} d_{in} &= 4 \\ (k, s, p) &= (3, 1, 0) \\ d_{out} &= \frac{d_{in} + 2p - k}{s} + 1 = 2 \end{aligned}$$

The eFCN layer is of size $(c_{in} \times d_{in} \times d_{in}, c_{out} \times d_{out} \times d_{out}) = (4, 16)$ since $c_{in} = c_{out} = 1$ here. In Fig. E.2, we show the typical structure of the eFCN weight matrices observed in practice.

E.2 Results with AlexNet on CIFAR-100

In this section, we show that the ideas we presented in the main text hold for various classes of data, architecture and optimizer. Namely, we show that our results hold when switching from SGD to Adam on CIFAR10, and for AlexNet [1] on the CIFAR-100 dataset. Each subsection contains figures which are counterparts of the ones of the main text: performance and training dynamics of the eFCNs in Fig. E.3, deviation from CNN subspace in Fig. E.4, role of off-local blocks in learning in Fig. E.5.

E.3 Interpolating between CNNs and eFCNs

Another way to understand the dynamics of the eFCNs is to examine the paths that connect them to the CNN they stemmed from in the FCN weight space. Interpolating in the weight space has received some attention in recent literature, in papers such as [202, 203], where it has been shown that contrary to previous beliefs the bottom of the landscapes of deep neural networks resembles a flat, connected level set since one can always find a path of low energy connecting minima.

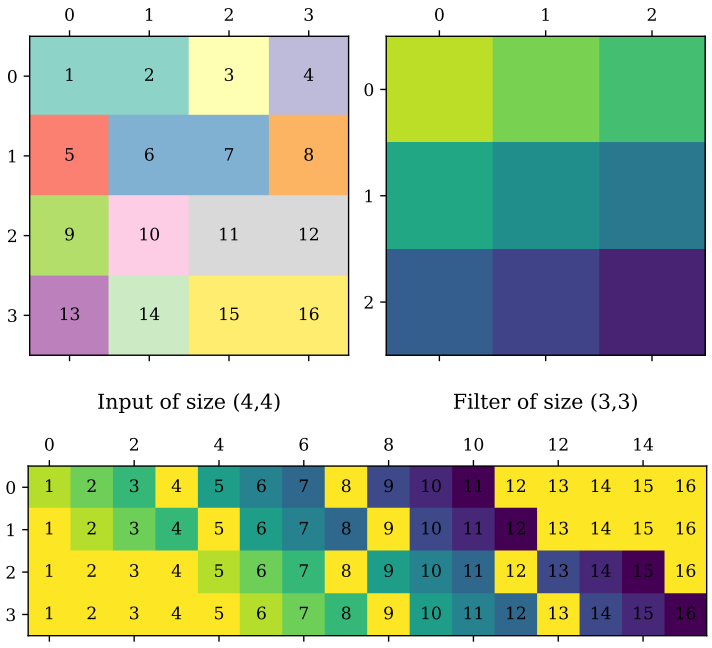


Figure E.1: eFCN wight matrix (**bottom**) obtained when acting on an input of size of size (4,4) (**top left**) with a filter of size (3,3) (**top right**). The colors of the eFCN weight matrix show where they stem from in the filter (the off-local blocks, in yellow, are set to zero at initialization).

Here we use two interpolation methods in weight space. The first method, labeled "linear", consists in sampling n equally spaced points along the linear path connecting the weights. Of course, the interpolated points generally have higher training loss than the endpoints.
The second method, labeled "string", consists in starting from the linear interpolation path, and letting the interpolated points fall down the landscape following gradient descent, while ensuring that they stay close enough together by adding an elastic term in the loss:

$$\mathcal{L}_{elastic} = \frac{1}{2} k \sum_{i=1}^{n-1} (\mathbf{x_{i+1}} - \mathbf{x_i})^2 \tag{E.1}$$

By adjusting the stiffness constant k we can control how straight the string is: at high k we recover the linear interpolation, whereas at low k the points decouple and reach the bottom of the landscape, but are far apart and don't give us an actual path. Note that this method is a simpler form of the one used in [203], where we don't use the "nudging" trick.
For comparison, we also show the performance obtained when interpolating directly in output space (as done in ensembling methods).
Results are shown in figure E.6, with the x-axis representing the interpolation parameter $\alpha \in [0, 1]$. We see that for both the linear and string interpolations, the training loss profile displays a barrier, except at late t_w where the the eFCN has not escaped far from the CNN subspace. Although the string method fails to find a path without a barrier, this is not sufficient to conclude that no paths exist.

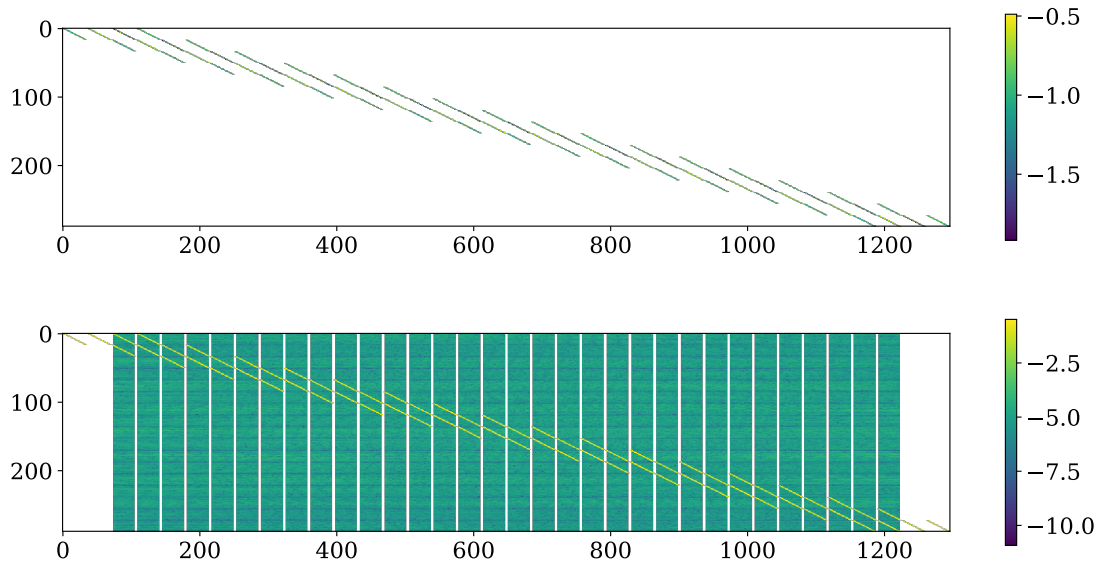


Figure E.2: **Top**: Heatmap of a block of weights corresponding to the first input channel and the first output channel of the first layer of the eFCN just after its initialization from the converged VanillaCNN. The colorscale indicates the natural logarithm of the absolute value of the weights. The highly sparse and self-repeating structure of the weight matrix is due to the locality and weight sharing constraints. **Bottom**: Same after training the eFCN for 100 epochs. The off-local blocks appear in blue: their weights are several orders of magnitude smaller in absolute value than those of the local blocks, in yellow. Note that due to the padding many weights stay at zero even after relaxing the constraints. When unflattened, the first row of this heatmap gives rise to the images shown in Fig. E.5.

However, the behavior of test accuracy is much more surprising. In all cases, despite the increase in training loss, the interpolated paths reach higher test accuracies than the endpoints, even at early t_w when the eFCN and the CNN are quite far from each other. This confirms that there is a basin of high generalization around the CNN subspace, and that optimum performance can actually be found somewhere in between the solution found by the CNN and the solution found by the eFCN. This offers yet another procedure to improve the performance in practice. However, in all cases we note that the gain in accuracy is lower than the gain obtained by interpolating in output space.

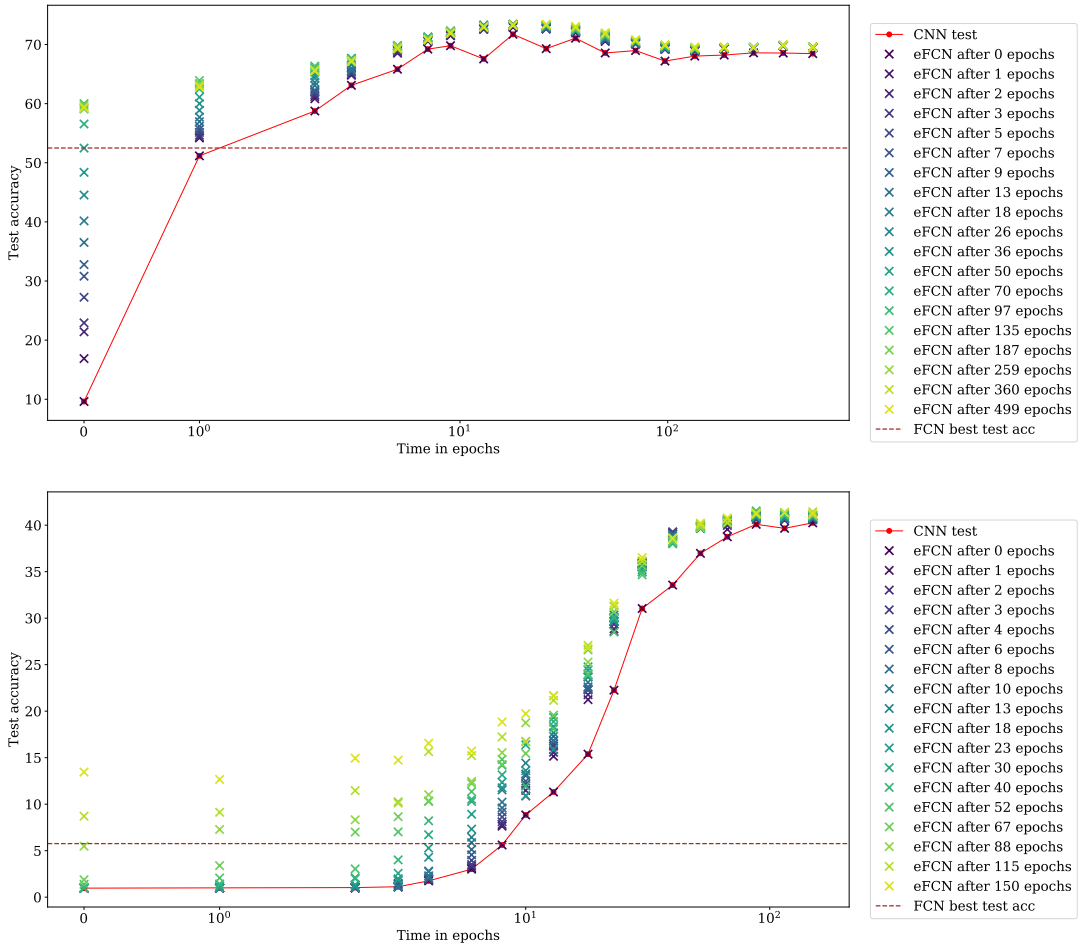


Figure E.3: This figure sums up in a compact way the generalization dynamics of the eFCNs. The red curve represents the test accuracy of the model versus its training time in epochs. Above each point t_w of the training, we depict as crosses the test accuracy history of the eFCN stemmed at relax time t_w, with colors indicating the training time of the eFCN after embedding. For comparison, the best test accuracy reached by a standard FCN of same size is depicted as a brown horizontal dashed line. **Top**: VanillaCNN on CIFAR-10, with Adam optimizer. **Bottom**: Alexnet on CIFAR-100, with SGD optimizer. We note that results are qualitatively similar: the eFCNs always improve after initialization, outperform the standard FCN, and we again observe that for some relax times the eFCNs even exceeds the best test accuracy reached by the CNN.

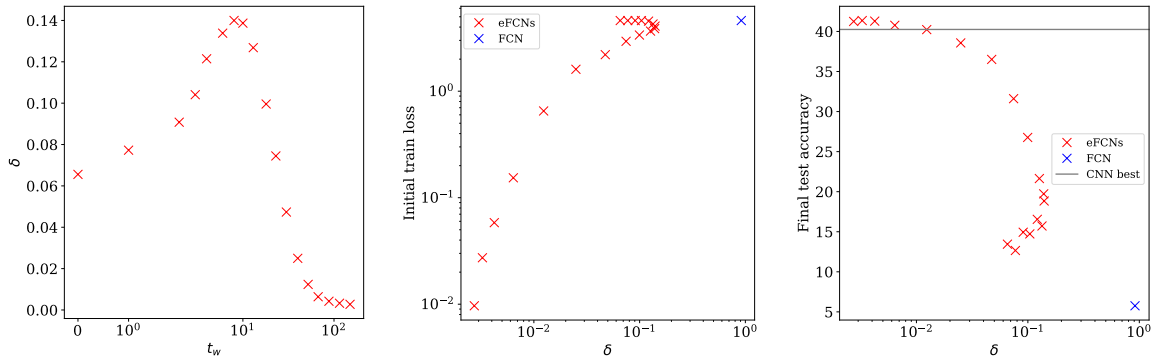


Figure E.4: **Left panel:** relax time t_w of the eFCN vs. δ, the measure of deviation from the CNN subspace through the locality constraint, at the final point of eFCN training. **Middle panel:** δ vs. the initial loss value. **Right panel:** δ vs. final test accuracy of eFCN models. For reference, the blue point in the **middle** and **right** panels indicate the deviation measure for a standard FCN, where $\delta \sim 97\%$.

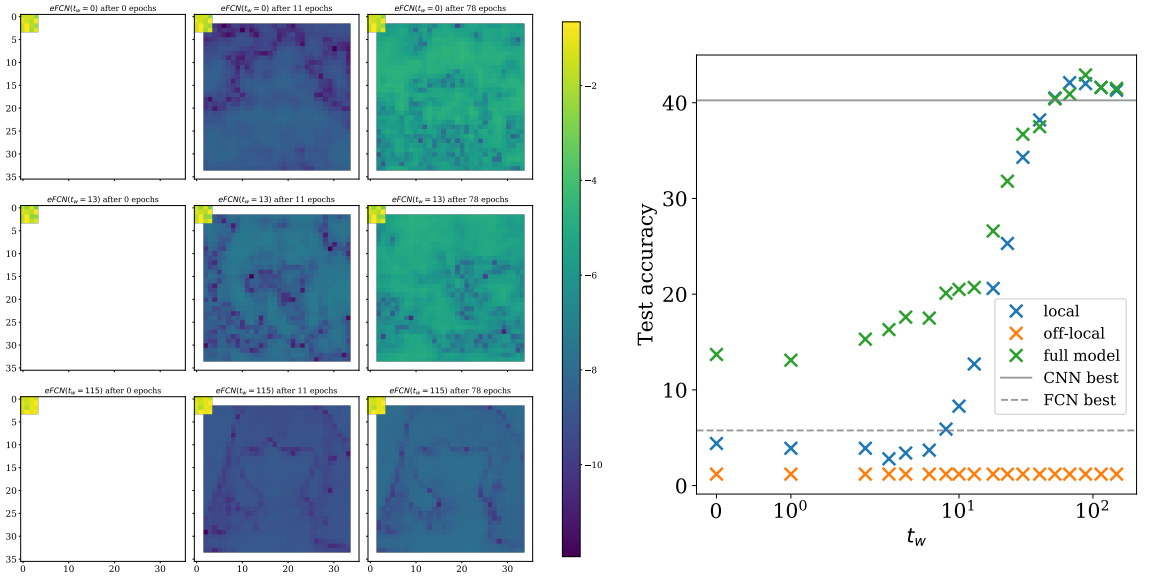
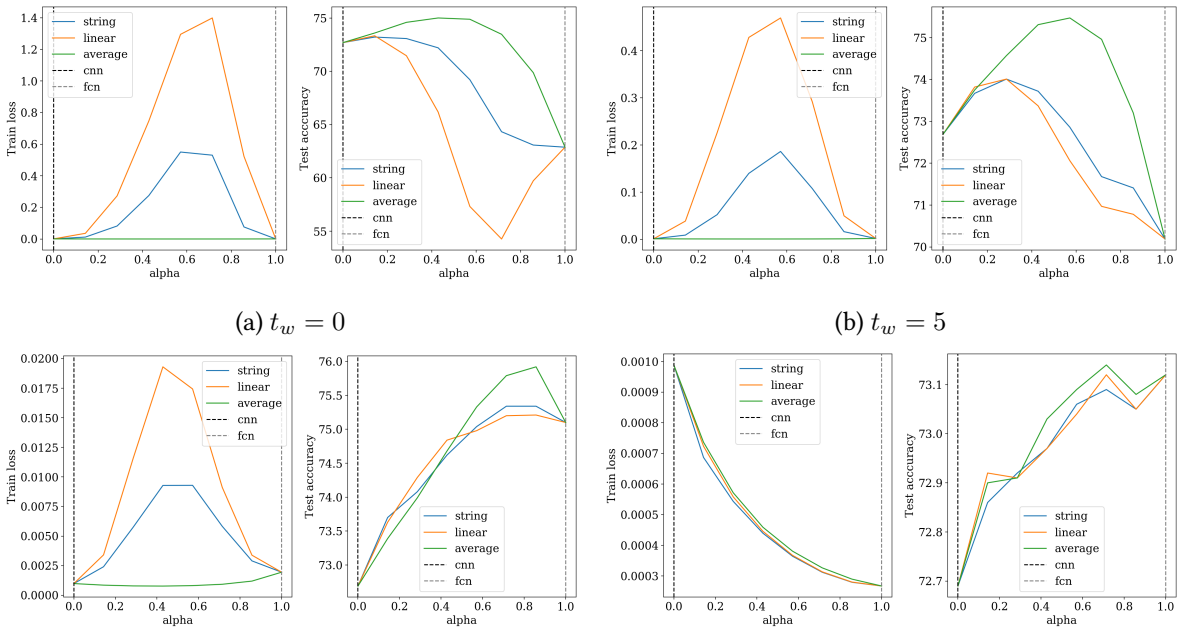


Figure E.5: **Left**: Visualization of an eFCN "filter" from the the first layer just after embedding (left column), after training after 11 epochs (middle column), and training after 78 epochs (right column); where the eFCN is initialized at relax times $t_w = 0$ (top row), $t_w = 13$ (middle row), and $t_w = 115$ (bottom row). The colors indicate the natural logarithm of the absolute value of the weights. **Right**: Contributions to the test accuracy of the local blocks (off-local blocks masked out) and off-local blocks (local blocks masked out).



(a) $t_w = 0$ (b) $t_w = 5$

(c) $t_w = 18$ (d) $t_w = 61$

Figure E.6: Interpolation between the solution reached by the CNN after 100 epochs (interpolation parameter $\alpha = 0$) and the solution found by the eFCN after 100 epochs (interpolation parameter $\alpha = 1$), for four different relax times t_w indicated below the subfigures. In each subfigure, the **left** panel shows train loss, and the **right** panel shows test accuracy. The orange line corresponds to linear interpolation, the blue line corresponds to string method interpolation, and the green line corresponds to interpolation in output space.

Appendix F

Improving Vision Transformers with Soft Convolutional Inductive Biases

F.1 The importance of positional gating

In the main text, we discussed the importance of using GPSA layers instead of the standard PSA layers, where content and positional information are summed before the softmax and lead the attention heads to focus only on the positional information. We give evidence for this claim in Fig. F.1, where we train a ConViT-B for 300 epochs on ImageNet, but replace the GPSA by standard PSA. The convolutional initialization of the PSA still gives the ConViT a large advantage over the DeiT baseline early in training. However, the ConViT stays in the convolutional configuration and ignores the content information, as can be seen by looking at the attention maps (not shown). Later in training, the DeiT catches up and surpasses the performance of the ConViT by utilizing content information.

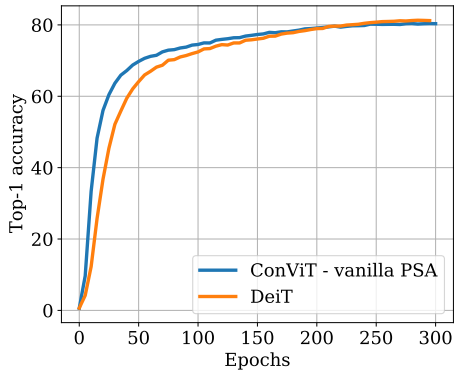


Figure F.1: **Convolutional initialization without GPSA is helfpul during early training but deteriorates final performance.** We trained the ConViT-B along with its DeiT-B counterpart for 300 epochs on ImageNet, replacing the GPSA layers of the ConViT-B by vanilla PSA layers.

F.2 The effect of distillation

Nonlocality In Fig. F.2, we compare the nonlocality curves of Fig. 7.6 of the main text with those obtained when the DeiT is trained via hard distillation from a RegNetY-16GF (84M parameters) [462], as in [217]. In the distillation setup, the nonlocality still drops in the early epochs of training, but increases less at late times compared to without distillation. Hence, the final internal states of the DeiT are more "local" due to the distillation. This suggests that knowledge distillation transfers the locality of the convolutional teacher to the student, in line with the results of [257].

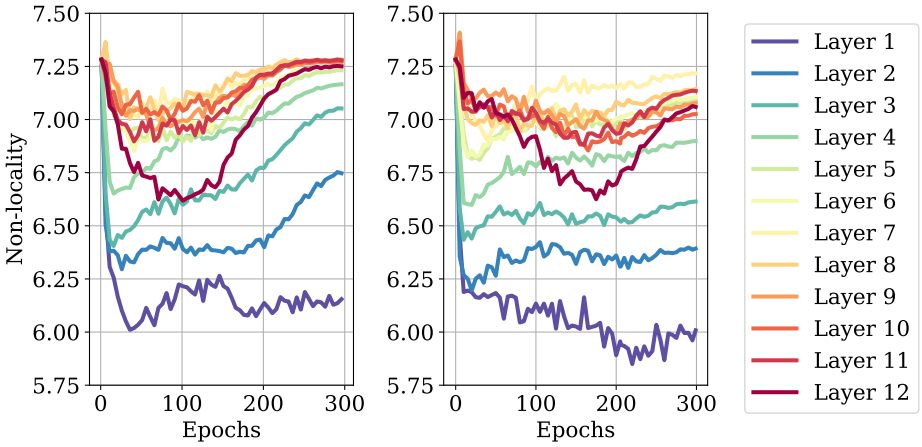


Figure F.2: **Distillation pulls the DeiT towards a more local configuration.** We plotted the nonlocality metric defined in Eq. 7.8 throughout training, for the DeiT-S trained on ImageNet. *Left:* regular training. *Right:* training with hard distillation from a RegNet teacher, by means of the distillation introduced in [217].

Performance The hard distillation introduced in [217] greatly improves the performance of the DeiT. We have verified the complementarity of their distillation methods with our ConViT. In the same way as in the DeiT paper, we used a RegNet-16GF teacher and experimented hard distillation during 300 epochs on ImageNet. The results we obtain are summarized in Tab. F.1.

Method	DeiT-S (22M)	DeiT-B (86M)	ConViT-S+ (48M)
No distillation	79.8	81.8	**82.2**
Hard distillation	80.9	**83.0**	82.9

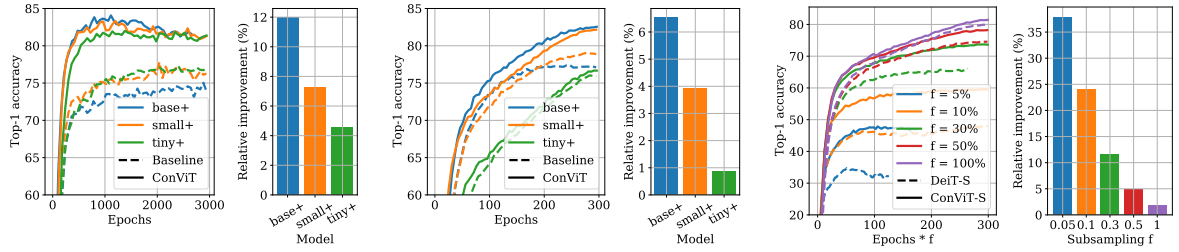
Table F.1: Top-1 accuracies of the ConViT-S+ compared to the DeiT-S and DeiT-B, both trained for 300 epochs on ImageNet.

Just like the DeiT, the ConViT benefits from distillation, albeit somewhat less than the DeiT, as can be seen from the DeiT-B performing less well than the ConViT-S+ without distillation but better with distillation. This hints to the fact that the convolutional inductive bias transferred from the teacher is redundant with its own convolutional prior.

Nevertheless, the performance improvement obtained by the ConViT with hard distillation demonstrates that instantiating soft inductive biases directly in a model can yield benefits on top of those obtained by instantiating such biases indirectly, in this case via distillation.

F.3 Further performance results

In Fig. F.3, we display the time evolution of the top-1 accuracy of our ConViT+ models on CIFAR100, ImageNet and subsampled ImageNet, along with a comparison with the corresponding DeiT+ models. For CIFAR100, we kept all hyperparameters unchanged, but rescaled the images to 224×224 and increased the number of epochs (adapting the learning rate schedule correspondingly) to mimic the ImageNet scenario. After 1000 epochs, the ConViTs shows clear signs of overfitting, but reach impressive performances (82.1% top-1 accuracy with 10M parameters, which is better than the EfficientNets reported in [463]).



(a) CIFAR100 (b) ImageNet-1k (c) Subsampled ImageNet

Figure F.3: **The convolutional inductive bias is particularly useful for large models applied to small datasets.** Each of the three panels displays the top-1 accuracy of the ConViT+ model and their corresponding DeiT+ throughout training, as well as the relative improvement between the best top-1 accuracy reached by the DeiT+ and that reached by the ConViT+. *Left:* tiny, small and base models trained for 3000 epochs on CIFAR100. *Middle:* tiny, small and base models trained for 300 epochs on ImageNet-1k. The relative improvement of the ConViT over the DeiT increases with model size. *Right:* small model trained on a subsampled version of ImageNet-1k, where we only keep a fraction $f \in \{0.05, 0.1, 0.3, 0.5, 1\}$ of the images of each class. The relative improvement of the ConViT over the DeiT increases as the dataset becomes smaller.

In Fig. F.4, we study the impact of the various ingredients of the ConViT (presence and number of GPSA layers, gating parameters, convolutional initialization) on the dynamics of learning.

F.4 Effect of model size

In Fig. F.5, we show the analog of Fig. 7.6 of the main text for the tiny and base models. Results are qualitatively similar to those observed for the small model. Interestingly, the first layers of DeiT-B and ConViT-B reach significantly higher nonlocality than those of the DeiT-Ti and ConViT-Ti.
In Fig. F.6, we show the analog of Fig. 7.7 of the main text for the tiny and base models. Again, results are qualitatively similar: the average weight of the positional attention, $\mathbb{E}_h \sigma(\lambda_h)$, decreases over time, so that more attention goes to the content of the image. Note that in the ConViT-Ti, only the first 4

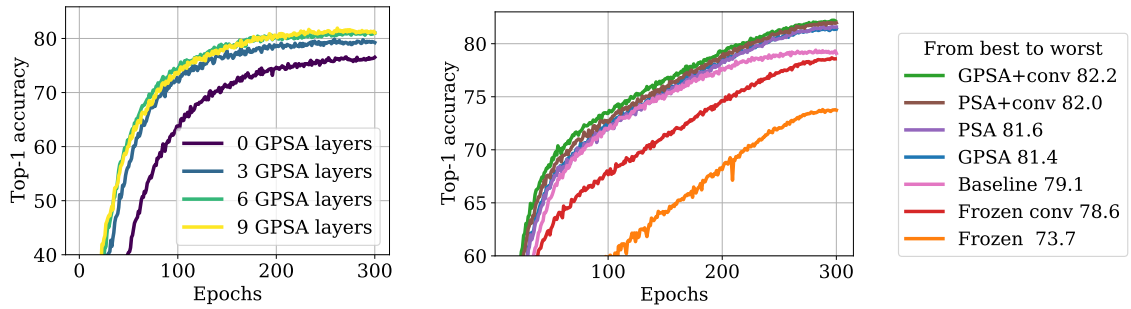
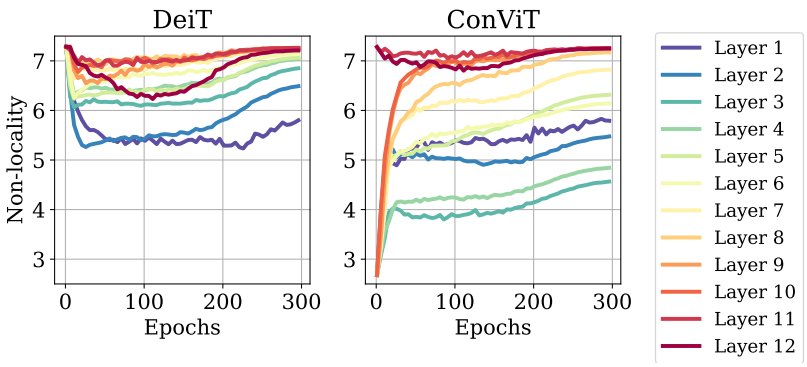
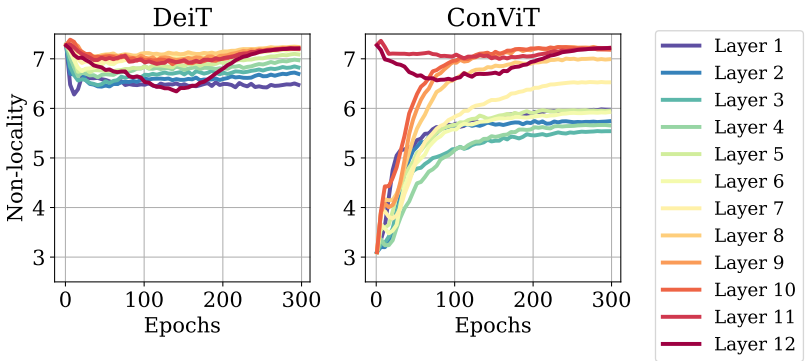


Figure F.4: **Impact of various ingredients of the ConViT on the dynamics of learning.** In both cases, we train the ConViT-S+ for 300 epochs on first 100 classes of ImageNet. *Left:* ablation on number of GPSA layers, as in Fig. 7.9. *Right:* ablation on various ingredients of the ConViT, as in Tab. 7.3. The baseline is the DeiT-S+ (pink). We experimented (i) replacing the 10 first SA layers by GPSA layers ("GPSA") (ii) freezing the gating parameter of the GPSA layers ("frozen gate"); (iii) removing the convolutional initialization ("conv"); (iv) freezing all attention modules in the GPSA layers ("frozen"). The final top-1 accuracy of the various models trained is reported in the legend.

layers still pay attention to position at the end of training (average gating parameter smaller than one), whereas for ConViT-S, the 5 first layers still do, and for the ConViT-B, the 6 first layers still do. This suggests that the larger (i.e. the more underspecified) the model is, the more layers make use of the convolutional prior.

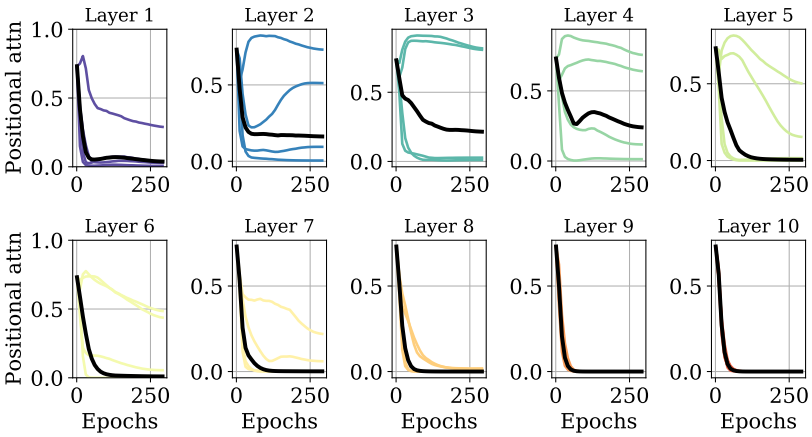


(a) DeiT-Ti and ConViT-Ti

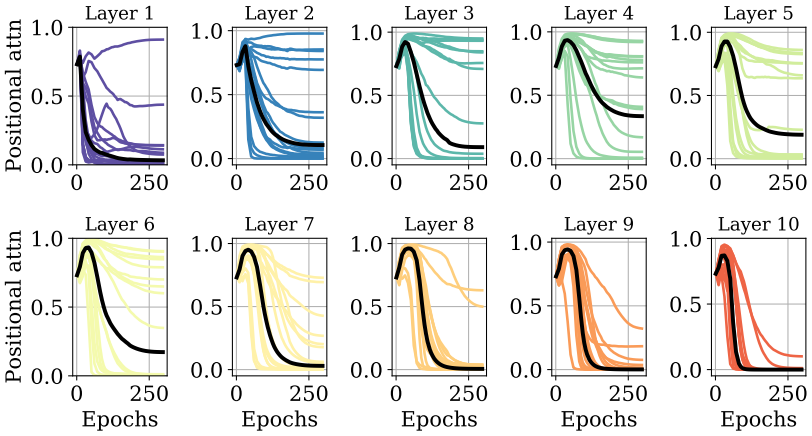


(b) DeiT-B and ConViT-B

Figure F.5: **The bigger the model, the more non-local the attention.** We plotted the nonlocality metric defined in Eq. 7.8 of the main text (the higher, the further the attention heads look from the query pixel) throughout 300 epochs of training on ImageNet-1k.



(a) ConViT-Ti

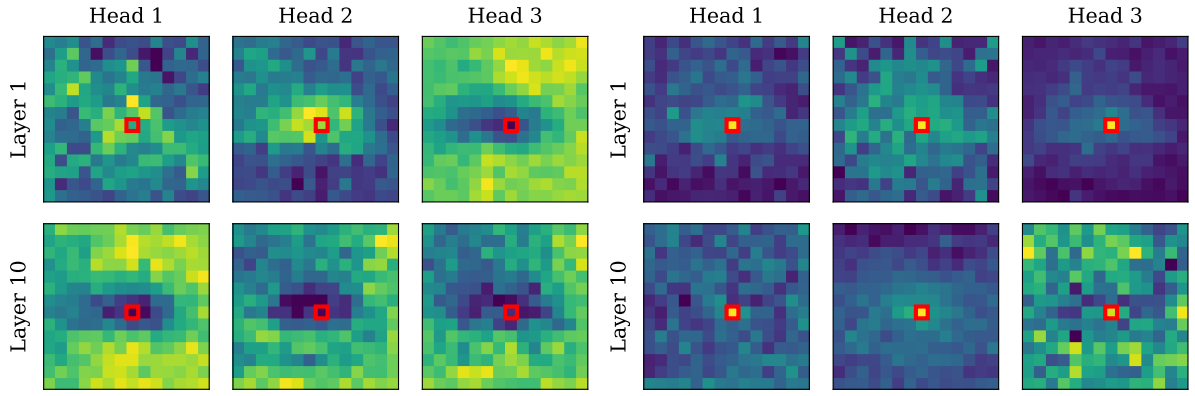


(b) ConViT-B

Figure F.6: **The bigger the model, the more layers pay attention to position.** We plotted the gating parameters of various heads and various layers, as in Fig. 7.7 of the main text (the lower, the less attention is paid to positional information) throughout 300 epochs of training on ImageNet-1k. Note that the ConViT-Ti only has 4 attention heads whereas the ConViT-B has 16, hence the different number of curves.

F.5 Attention maps

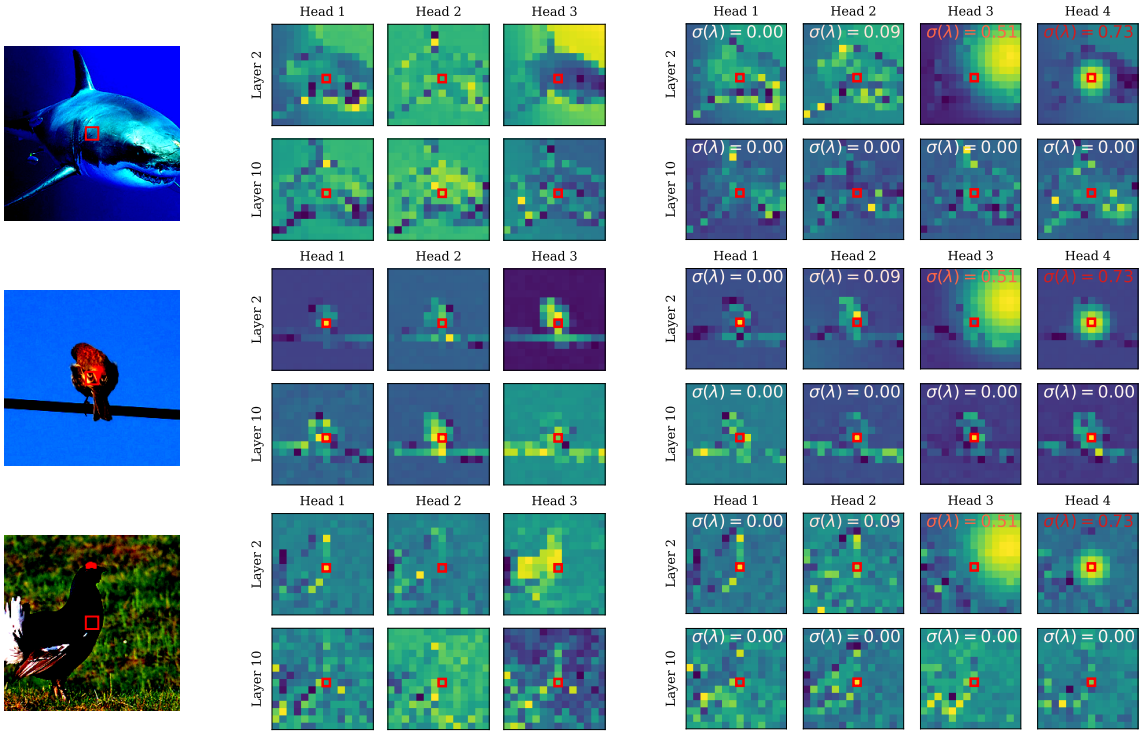
Attention maps of the DeiT reveal locality In Fig. F.7, we give some visual evidence for the fact that vanilla SA layers extract local information by averaging the attention map of the first and tenth layer of the DeiT over 100 images. Before training, the maps look essentially random. After training, however, most of the attention heads of the first layer focus on the query pixel and its immediate surroundings, whereas the attention heads of the tenth layer capture long-range dependencies.



(a) Before training (b) After training

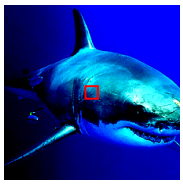
Figure F.7: **The averaged attention maps of the DeiT reveal locality at the end of training.** To better visualise the center of attention, we averaged the attention maps over 100 images. *Top:* before training, the attention patterns exhibit a random structure. *Bottom:* after training, most of the attention is devoted to the query pixel, and the rest is focused on its immediate surroundings.

Attention maps of the ConViT reveal the diversity of the attention heads In Fig. F.8, we show a comparison of the attention maps of Deit-Ti and ConViT-Ti for different images of the ImageNet validation set. In Fig. F.9, we compare the attention maps of DeiT-S and ConViT-S.
In all cases, results are qualitatively similar: the DeiT attention maps look similar across different heads and different layers, whereas those of the ConViT perform very different operations. Notice that in the second layer, the third and forth head focus stay local whereas the first two heads focus on content. In the last layer, all the heads ignore positional information, focusing only on content.

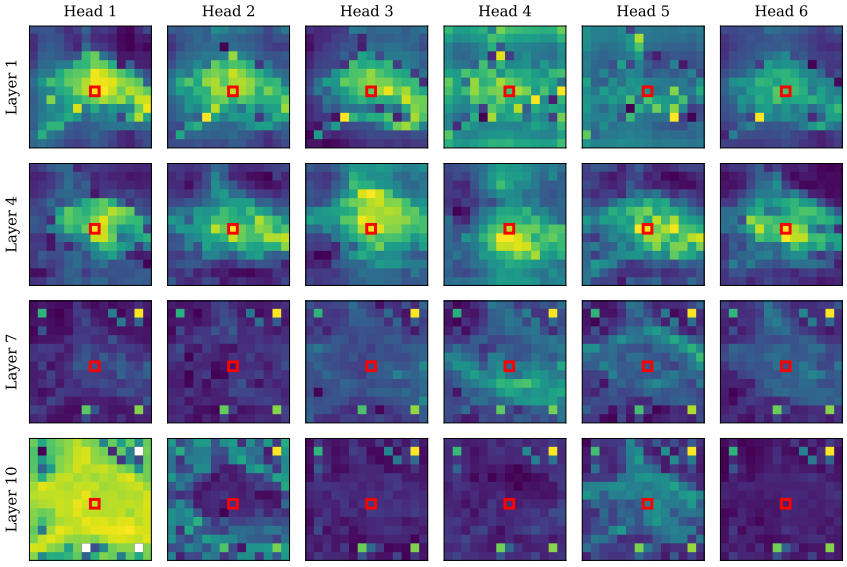


(a) Input images (b) DeiT (c) ConViT

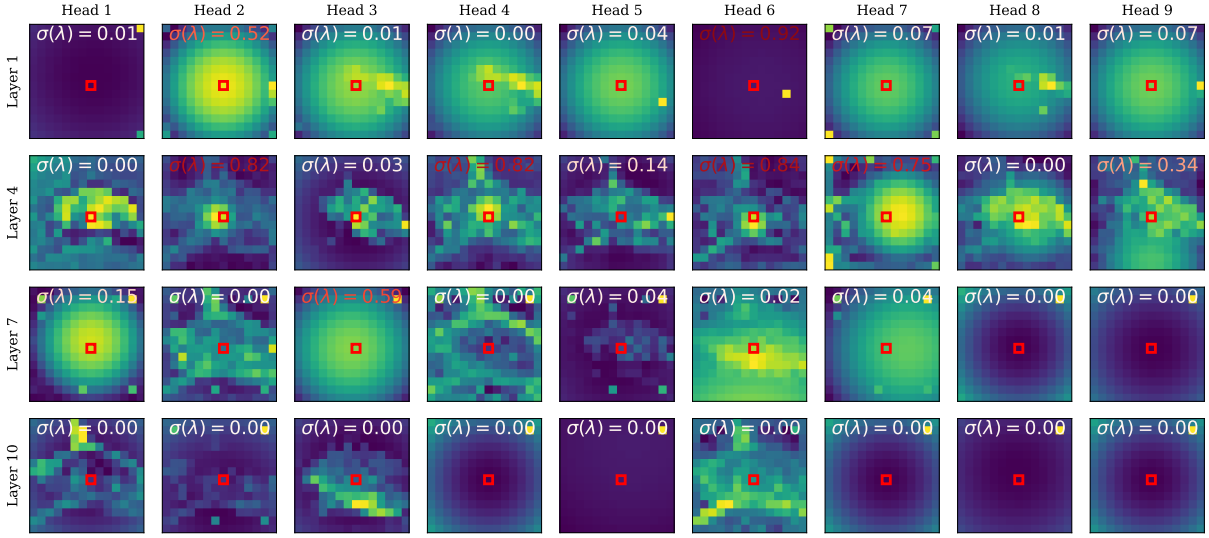
Figure F.8: *Left:* input image which is embedded then fed into the models. The query patch is highlighted by a red box and the colormap is logarithmic to better reveal details. *Center:* attention maps obtained by a DeiT-Ti after 300 epochs of training on ImageNet. *Right:* Same for ConViT-Ti. In each map, we indicated the value of the gating parameter in a color varying from white (for heads paying attention to content) to red (for heads paying attention to position).



(a) Input image



(b) DeiT



(c) ConViT

Figure F.9: Attention maps obtained by a DeiT-S and ConViT-S after 300 epochs of training on ImageNet. In each map, we indicated the value of the gating parameter in a color varying from white (for heads paying attention to content) to red (for heads paying attention to position).

F.6 Further ablations

In this section, we explore masking off various parts of the network to understand which are most crucial.
In Tab. F.2, we explore the importance of the absolute positional embeddings injected to the input in both the DeiT and ConViT. We see that masking them off at test time a mild impact on accuracy for the ConViT, but a significant impact for the DeiT, which is expected as the ConViT already has relative positional information in each of the GPSA layers. This also shows that the absolute positional information contained in the embeddings is not very useful.
In Tab. F.3, we explore the relative importance of the positional and content information by masking them off at test time. To do so, we manually set the gating parameter $\sigma(\lambda)$ to 1 (no content attention) or 0 (no positional attention). In the first GPSA layers, both procedures affect performance similarly, signalling that positional and content information are both useful. However in the last GPSA layers, masking the content information kills performance, whereas masking the positional information does not, confirming that content information is more crucial.

Model	Mask pos embed	No mask
DeiT-Ti	38.3	72.2
ConViT-Ti	67.1	73.1

Table F.2: Performance on ImageNet with the positional embeddings masked off at test time.

# layers masked	Mask content	Mask position	No mask
3	62.3	63.5	73.1
5	35.0	53.1	73.1
10	1.3	46.8	73.1

Table F.3: Performance of ConViT-Ti on ImageNet with positional or content attention masked off at test time.

Appendix G

Transformed CNNs: Recasting Pre-trained Convolutional Networks as Transformers

G.1 Performance table

In Tab. G.1, we display the characteristics and the performance of our T-ResNet-RS models and compare them to the original ResNet-RS models as well as several other strong baselines reported in [274, 275].

G.2 Changing the learning rate

We have shown that the learning dynamics decompose into two phases: the learning rate warmup phase, where the test loss drops, then the learning rate decay phase, where the test loss increases again. This could lead one to think that the maximal learning rate is too high, and the dip could be avoided by choosing a lower learning rate. Yet this is not the case, as shown in Fig. G.1. Reducing the maximal learning rate indeed reduces the dip, but it also slows down the increase in the second phase of learning. This confirms that the model needs to "unlearn" the right amount to find better solutions.

G.3 Changing the test resolution

One advantage of the GPSA layers introduced by [271] is how easily they adapt to different image resolutions. Indeed, the positional embeddings they use are fixed rather than learnt. They simply consist in 3 values for each pair of pixels: their euclidean distance $\|\boldsymbol{\delta}\|$, as well as their coordinate distance $\boldsymbol{\delta}_1, \boldsymbol{\delta}_2$ (see Eq. 7.5). Our implementation automatically adjusts these embeddings to the input image, allowing us to change the test resolution seamlessly.
In Fig. G.2, we show how the top-1 accuracies of our T-ResNet-RS models compares to those of the ResNet-RS models finetuned at same resolution but without SA. At test resolution 416, our T-ResNetRS-350 reaches an impressive top-1 accuracy of 84.9%, beyond those of the best EfficientNets and BotNets [269].

Model	Res.	Params	Speed	Flops	IN-1k	IN-C	IN-A	IN-R	FGSM	PGD
Transformers										
ViT-B/16‡	224	86 M	182	16.9	77.9	52.2	7.0	21.9	30.6	14.3
ViT-L/16‡	224	307 M	55	59.7	76.5	49.3	6.1	17.9	27.8	13.0
ViT-B/16 21k‡	224	86 M	182	16.9	84.0	65.8	26.7	38.0	31.3	10.3
ViT-L/16 21k‡	224	307 M	55	59.7	**85.1**	**70.0**	28.1	40.6	40.5	16.2
DeiT-S†	224	22 M	544	4.6	79.9	55.4	18.9	31.0	40.7	16.7
DeiT-B†	224	87 M	182	17.6	82.0	60.7	27.4	34.6	46.4	21.3
ConViT-S†	224	28 M	296	5.4	81.5	59.5	24.5	34.0	41.0	17.2
ConViT-B†	224	87 M	139	17.7	82.4	61.9	29.0	36.9	51.8	20.8
RVT-S†	224	23.3 M	-	4.7	81.9	-	25.7	47.7	51.8	28.2
RVT-B†	224	91.8 M	-	17.7	82.6	-	28.5	48.7	53.0	29.9
CNNs										
ResNet50‡	224	25 M	736	4.1	76.8	46.1	4.2	21.5	-	-
ResNet101‡	224	45 M	435	7.85	78.0	50.2	6.3	23.0	14.7	2.0
ResNet101x3‡	224	207 M	62	69.6	80.3	53.4	9.1	24.5	23.6	7.3
ResNet152x4‡	224	965 M	18	183.1	80.4	54.5	11.6	25.8	33.3	10.5
ResNet50-RS	160	36 M	938	4.6	78.8	36.8	5.7	39.1	28.7	18.4
ResNet101-RS	192	64 M	674	12.1	80.3	44.1	11.8	44.8	32.9	18.8
ResNet152-RS	256	87 M	304	31.2	81.2	49.9	23.4	45.9	41.6	28.5
ResNet200-RS	256	93 M	225	40.4	82.8	49.3	25.4	48.1	40.4	24.6
ResNet270-RS	256	130 M	152	54.2	83.8	53.6	26.6	48.7	44.7	30.3
ResNet350-RS	288	164 M	89	87.5	84.0	53.9	34.9	49.7	48.3	34.6
Our Transformed CNNs										
T-ResNet50-RS	224	38 M	447	17.6	81.0	48.0	18.7	42.9	47.2	33.9
T-ResNet101-RS	224	66 M	334	25.1	82.4	52.9	27.7	47.8	50.3	34.2
T-ResNet152-RS	320	89 M	128	65.8	83.7	54.5	39.8	50.6	57.3	36.8
T-ResNet200-RS	320	96 M	105	80.2	84.0	57.0	41.2	51.1	58.3	36.4
T-ResNet270-RS	320	133 M	75	107.2	84.3	58.6	43.7	51.4	**59.0**	**36.6**
T-ResNet350-RS	320	167 M	61	130.5	84.5	59.2	**44.8**	**53.8**	53.4	36.4

Table G.1: **Accuracy of our models on various benchmarks.** Throughput is the number of images processed per second on a V100 GPU at batch size 32. For ImageNet-C, we keep a resolution of 224 at test time to avoid distorting the corruptions; this disadvantages our large models, which are trained at higher resolutions. †: reported from [275] (we recalculated ImageNet-C accuracies, as the original paper reports MCE). ‡: reported from [274] (in their setup, PGD uses 8 steps with a stepsize of 1/8).

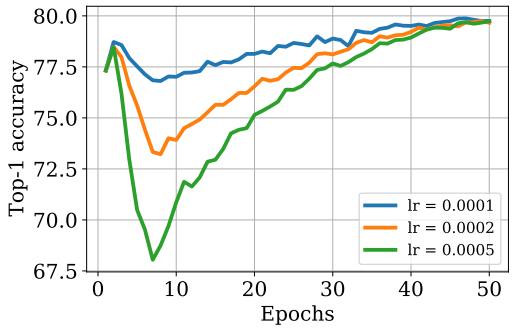


Figure G.1: **The larger the learning rate, the lower the test accuracy dips, but the faster it climbs back up.** We show the dynamics of the ResNet50, fine-tuned for 50 epochs at resolution 224, for three different values of the maximal learning rate.

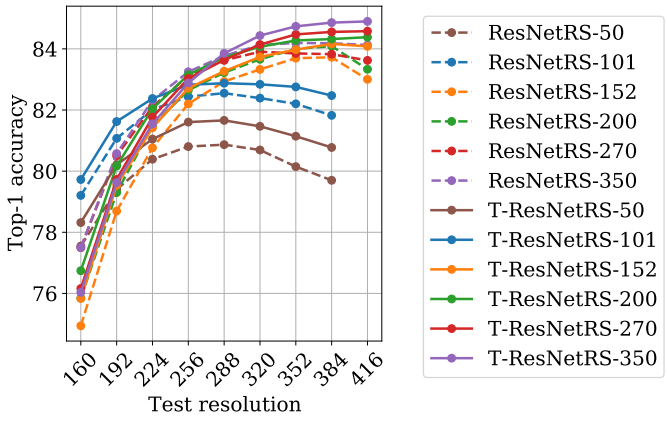


Figure G.2: **Performance at different test-time resolutions, for the finetuned models with and without SA.** The ResNet50-RS and ResNet101-RS models are finetuned at resolution 224, and all other models are finetuned at resolution 320.

G.4 Changing the number of epochs

In Tab. G.2, we show how the top-1 accuracy of the T-ResNet-RS model changes with the number of fine-tuning epochs. As expected, performance increases significantly as we fine-tune for longer, yet we chose to set a maximum of 50 fine-tuning epochs to keep the computational cost of fine-tuning well below that of the original training.

G.5 Changing the architecture

Our framework, which builds on the timm package, makes changing the original CNN architecture very easy. We applied our fine-tuning procedure to the ResNet-D models [464] with the exact same hyperparameters, and observed substantial performance gains, similar to the ones obtained for ResNet-RS, see Tab. G.3. This suggests the wide applicability of our method.

Model	Epochs	Top-1 acc
ResNet50-RS	0	79.91
T-ResNet50-RS	10	80.11
T-ResNet50-RS	20	80.51
T-ResNet50-RS	50	**81.02**
ResNet101-RS	0	81.70
T-ResNet101-RS	10	81.54
T-ResNet101-RS	20	81.90
T-ResNet101-RS	50	**82.39**

Table G.2: **Longer fine-tuning increases final performance.** We report the top-1 accuracies of our models on ImageNet-1k at resolution 224.

Model	Original res.	Original acc.	Fine-tune res.	Fine-tune acc.	Gain
T-ResNet50-D	224	80.6	320	81.6	+1.0
T-ResNet101-D	320	82.3	384	83.1	+0.8
T-ResNet152-D	320	83.1	384	83.8	+0.7
T-ResNet200-D	320	83.2	384	**83.9**	+0.7
T-ResNet50-RS	160	78.8	224	81.0	+2.8
T-ResNet101-RS	192	81.2	224	82.4	+1.2
T-ResNet152-RS	256	83.0	320	83.7	+0.7
T-ResNet200-RS	256	83.4	320	**84.0**	+0.6

Table G.3: **Comparing the performance gains of the ResNet-RS and ResNet-D architectures.** Top-1 accuracy is measured on ImageNet-1k validation set. The pre-trained models are all taken from the timm library [253].

Appendix H

Align, then Memorise: the Dynamics of Learning with Feedback Alignment

H.1 Derivation of the ODE

The derivation of the ODE's that describe the dynamics of the test error for shallow networks closely follows the one of [355] and [357] for back-propagation. Here, we give the main steps to obtain the analytical curves of the main text and refer the reader to their paper for further details.
As we discuss in Sec. 10.2, student and teacher are both two-layer networks with K and M hidden nodes, respectively. For an input $x \in \mathbb{R}^D$, their outputs $\hat{y}$ and y can be written as

$$\hat{y} = \phi_\theta(x) = \sum_{k=1}^{K} W_2^k \, \sigma\left(\lambda^k\right), \\ y = \phi_{\tilde{\theta}}(x) = \sum_{m=1}^{M} \tilde{W}_2^m \, \sigma\left(\nu^m\right), \tag{H.1}$$

where we have introduced the pre-activations $\lambda^k \equiv W_1^k x/\sqrt{D}$ and $\nu^m \equiv \tilde{W}_1^m x/\sqrt{D}$. Evaluating the test error of a student with respect to the teacher under the squared loss leads us to compute the average

$$\epsilon_g\left(\theta, \tilde{\theta}\right) = \frac{1}{2}\mathbb{E}_x \left[\sum_{k=1}^{K} W_2^k \, \sigma\left(\lambda^k\right) - \sum_{m=1}^{M} \tilde{W}_2^m \, \sigma\left(\nu^m\right)\right]^2, \tag{H.2}$$

where the expectation is taken over inputs x for a fixed student and teacher. Since x only enters Eq. (H.2) via the pre-activations $\lambda = (\lambda^k)$ and $\nu = (\nu^m)$, we can replace the high-dimensional average over x by a low-dimensional average over the $K + M$ variables (λ, ν). The pre-activations are jointly Gaussian since the inputs are drawn element-wise i.i.d. from the Gaussian distribution. The mean of (λ, ν) is zero since $\mathbb{E}\, x_i = 0$, so the distribution of (λ, ν) is fully described by the second moments

$$Q^{kl} = \mathbb{E}\, \lambda^k \lambda^l = W_1^k \cdot W_1^l / D, \tag{H.3}$$
$$R^{km} = \mathbb{E}\, \lambda^k \nu^m = W_1^k \cdot \tilde{W}_1^m / D, \tag{H.4}$$
$$T^{mn} = \mathbb{E}\, \nu^m \nu^n = \tilde{W}_1^m \cdot \tilde{W}_1^n / D. \tag{H.5}$$

which are the "order parameters" that we introduced in the main text. We can thus rewrite the generalisation error (10.5) as a function of only the order parameters and the second-layer weights,

$$\lim_{D\to\infty} \epsilon_g(\theta, \tilde{\theta}) = \epsilon_g(Q, R, T, W_2, \tilde{W}_2) \tag{H.6}$$

As we update the weights using SGD, the time-dependent order parameters Q, R, and W_2 evolve in time. By choosing different scalings for the learning rates in the SGD updates (10.4), namely

$$\eta_{W_1} = \eta, \qquad \eta_{W_2} = \eta/D$$

for some constant η, we guarantee that the dynamics of the order parameters can be described by a set of ordinary differential equations, called their "equations of motion". We can obtain these equations in a heuristic manner by squaring the weight update (10.4) and taking inner products with $\tilde{W}_1^m$, to yield the equations of motion for Q and R respectively:

$$\frac{\mathrm{d}R^{km}}{\mathrm{d}\alpha} = -\eta F_1^k \mathbb{E}\left[g'(\lambda^k)\nu^m e\right] \tag{H.7a}$$

$$\frac{\mathrm{d}Q^{k\ell}}{\mathrm{d}\alpha} = -\eta F_1^k \mathbb{E}\left[g'(\lambda^k)\lambda^\ell e\right] - \eta F_1^\ell \mathbb{E}\left[g'(\lambda^\ell)\lambda^k e\right] + \eta^2 F_1^k F_1^\ell \mathbb{E}\left[g'(\lambda^k)g'(\lambda^\ell)e^2\right], \tag{H.7b}$$

$$\frac{\mathrm{d}W_2^k}{\mathrm{d}\alpha} = -\eta \mathbb{E}\left[\sigma(\lambda^k)e\right] \tag{H.7c}$$

where, as in the main text, we introduced the error $e = \phi_\theta(x) - \phi_{\tilde{\theta}}(x)$. In the limit $D\to\infty$, the variable $\alpha = \mu/D$ becomes a continuous time-like variable. The remaining averages over the pre-activations, such as

$$\mathbb{E}\, g'(\lambda^k)\lambda^\ell \sigma(\nu^m),$$

are simple three-dimensional integral over the Gaussian random variables λ^k, λ^ℓ and ν^m and can be evaluated analytically for the choice of $\sigma(x) = \mathrm{erf}(x/\sqrt{2})$ [357] and for linear networks with $\sigma(x) = x$. Furthermore, these averages can be expressed only in term of the order parameters, and so the equations close. We note that the asymptotic exactness of Eqs. H.7 can be proven using the techniques used recently to prove the equations of motion for BP [370].
We provide an integrator for the full system of ODEs for any K and M in the Github repository.

H.2 Detailed analysis of DFA dynamics

In this section, we present a detailed analysis of the ODE dynamics in the matched case $K = M$ for sigmoidal networks ($\sigma(x) = \mathrm{erf}\,(x/\sqrt{2})$).

The Early Stages and Gradient Alignment We now use Eqs. (H.7) to demonstrate that alignment occurs in the early stages of learning, determining from the start the solution DFA will converge to (see Fig. 10.3 which summarises the dynamical evolution of the student's second layer weights).

Assuming zero initial weights for the student and orthogonal first layer weights for the teacher (i.e. T^{nm} is the identity matrix), for small times ($t \ll 1$), one can expand the order parameters in t:

$$\begin{aligned} R^{km}(t) &= t\dot{R}^{km}(0) + \mathcal{O}(t^2), \\ Q^{kl}(t) &= t\dot{Q}^{kl}(0) + \mathcal{O}(t^2), \\ W_2^k(t) &= t\dot{W}_2^k(0) + \mathcal{O}(t^2). \end{aligned} \tag{H.8}$$

where, due to the initial conditions, $R(0) = Q(0) = W_2(0) = 0$. Using Eq. H.7, we can obtain the lowest order term of the above updates:

$$\begin{aligned} \dot{R}^{km}(0) &= \frac{\sqrt{2}}{\pi}\eta\tilde{W}_2^m F_1^k, \\ \dot{Q}^{kl}(0) &= \frac{2}{\pi}\eta^2\left((\tilde{W}_2^k)^2 + (\tilde{W}_2^l)^2\right)F_1^l F_1^k, \\ \dot{W}_2^k(0) &= 0 \end{aligned} \tag{H.9}$$

Since both $\dot{R}(0)$ and $\dot{Q}(0)$ are non-zero, this initial condition is not a fixed point of DFA. To analyse initial alignment, we consider the first order term of $\dot{W}_2$. Using Eq. (H.8) with the derivatives at $t = 0$ (H.9), we obtain to linear order in t:

$$\dot{W}_2^k(t) = \frac{2}{\pi^2}\eta^2||\tilde{W}_2||^2 F_1^k t. \tag{H.10}$$

Crucially, this update is in the direction of the feedback vector F_1. DFA training thus constrains the student to initially grow in the direction of the feedback vector and align with it. This implies gradient alignment between BP and DFA and dictates into which of the many degenerate solutions in the energy landscape the student converges.

Plateau phase After the initial phase of learning with DFA where the test error decreases exponentially, similarly to BP, the student falls into a symmetric fixed point of the Eqs. (H.7) where the weights of a single student node are correlated to the weights of all the teacher nodes ([76, 355, 357]). The test error stays constant while the student is trapped in this fixed point. We can obtain an analytic expression for the order parameters under the assumption that the teacher first-layer weights are orthogonal ($T^{nm} = \delta_{nm}$). We set the teacher's second-layer weights to unity for notational simplicity ($\tilde{W}_2^m = 1$) and restrict to linear order in the learning rate η, since this is the dominant contribution to the learning dynamics at early times and on the plateau [356]. In the case where all components of the feedback vector are positive, the order parameters are of the form $Q^{kl} = q, R^{km} = r, W_2^k = w_2$ with:

$$q = \frac{1}{2K-1}, \quad r = \sqrt{\frac{q}{2}}, \quad w_2 = \sqrt{\frac{1+2q}{q(4+3q)}}. \tag{H.11}$$

If the components of the feedback vector are not all positive, we instead obtain $R^{km} = \mathrm{sgn}(F^k)r$, $W_2^k = \mathrm{sgn}(F^k)w_2$ and $Q^{kl} = \mathrm{sgn}(F^k)\,\mathrm{sgn}(F^l)q$. This shows that on the plateau the student is already in the configuration that maximises its alignment with F_1. Note that in all cases, the value of the test error reached at the plateau is the same for DFA and BP.

Memorisation phase and Asymptotic Fixed Point At the end of the plateau phase, the student converges to its final solution, which is often referred to as the *specialised* phase [76, 355, 357]. The configuration of the order parameters is such that the student reproduces her teacher up to sign changes that guarantee the alignment between W_2 and F_1 is maximal, i.e. $\mathrm{sgn}(W_2^k) = \mathrm{sgn}(F_1^k)$. The final value of the test error of a student trained with DFA is the same as that of a student trained with BP on the same teacher.

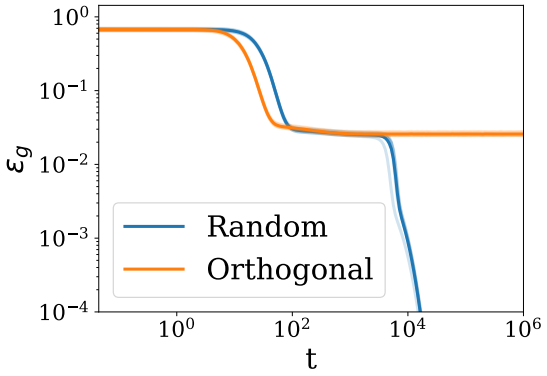


Figure H.1: Test error of a sigmoidal student started with zero initial weights. The feedback vector F_1 is chosen random (blue) and orthogonal to the teacher's second layer weights $\tilde{W}_2$ (orange). *Parameters:* $\eta = 0.1, K = M = 2$.

Choice of the feedback vector In the main text, we saw how a wrong choice of feedback vector F_1 can prevent a ReLU student from learning a task. Here, we show that also for sigmoidal student, a *wrong* choice of feedback vector F_1 is possible. As Fig. H.1 shows, in the case where the F_1 is taken orthogonal to the teacher second layer weights, a student whose weights are initialised to zero remains stuck on the plateau and is unable to learn. In contrast, when the F_1 is chosen with random i.i.d. components drawn from the standard normal distribution, perfect recovery is achieved.

H.3 Derivation of weight alignment

Since the network is linear, the update equations are (consider the first three layers only):

$$\delta W_1 = -\eta (F_1 e) x^T, \tag{H.12}$$

$$\delta W_2 = -\eta (F_2 e)(W_1 x)^\top, \tag{H.13}$$

$$\delta W_3 = -\eta (F_3 e)(W_2 W_1 x)^\top \tag{H.14}$$

First, it is straightforward to see that

$$W_1^t = -\eta \sum_{t'=0}^{t-1} F_1 e_{t'} x_{t'}^\top = F_1 A_1^t \tag{H.15}$$

$$A_1^t = -\eta \sum_{t'=0}^{t-1} e_{t'} x_{t'}^\top \tag{H.16}$$

This allows to calculate the dynamics of W_2^t:

$$\delta W_2^t = -\eta F_2 e_t (A_1^t x_t)^\top F_1^\top \tag{H.17}$$

$$W_2^t = -\eta \sum_{t'=0}^{t-1} F_2 e_t (A_1^{t'} x_{t'})^\top F_1^\top = F_2 A_2^t F_1^\top \tag{H.18}$$

$$A_2^t = -\eta \sum_{t'=0}^{t-1} e_{t'} (A_1^{t'} x_{t'})^\top = \eta^2 \sum_{t'=0}^{t-1} \sum_{t''=0}^{t'-1} (x_{t'} \cdot x_{t''}) e_{t'} e_{t''}^\top. \tag{H.19}$$

Which in turns allows to calculate the dynamics of W_3^t:

$$\delta W_3^t = -\eta F_3 e_t (F_2 A_2^{t'} F_1^\top F_1 A_1^{t'} x_t)^\top \tag{H.20}$$

$$W_3^t = -\eta \sum_{t'=0}^{t-1} F_3 e_{t'} (F_2 A_2^{t'} F_1^\top F_1 A_1^{t'} x_t)^\top = F_3 A_3^t F_2^\top \tag{H.21}$$

$$A_3^t = -\eta \sum_{t'=0}^{t-1} F_3 e_{t'} (A_2^{t'} F_1^\top F_1 A_1^{t'} x_{t'})^\top \tag{H.22}$$

$$= \eta^2 \sum_{t'=0}^{t-1} \sum_{t''=0}^{t'-1} (A_1^{t'} x_{t'}) \cdot (A_1^{t''} x_{t''}) e_{t'} e_{t''}^\top. \tag{H.23}$$

By induction it is easy to show the general expression:

$$A_1^t = -\eta \sum_{t'=0}^{t-1} e_{t'} x_{t'}^\top \tag{H.24}$$

$$A_2^t = \eta^2 \sum_{t'=0}^{t-1} \sum_{t''=0}^{t'-1} (x_{t'} \cdot x_{t''}) e_{t'} e_{t''}^\top \tag{H.25}$$

$$A_{l\geq 3}^t = \eta^2 \sum_{t,t'=0} (A_{l-2}^{t'} \dots A_1^{t'} x_{t'}) \cdot (A_{l-2}^{t''} \dots A_1^{t''} x_{t''}) e_{t'} e_{t''}^\top \tag{H.26}$$

Defining $A_0 \equiv \mathbb{I}_D$, one can rewrite this as in Eq. 10.15

$$A_{l\geq 2}^t = \eta^2 \sum_{t'=0}^{t-1} \sum_{t''=0}^{t'-1} (B_l^{t'} x_{t'}) \cdot (B_l^{t''} x_{t''}) e_{t'} e_{t''}^\top, \tag{H.27}$$

$$B_l = A_{l-2} \cdots A_0. \tag{H.28}$$

H.4 Impact of data structure

To study the impact of data structure on the alignment, the simplest setup to consider is that of Direct Random Target Projection [358]. Indeed, in this case the error vector $e_t = -y_t$ does not depend on the prediction of the network: the dynamics become explicitly solvable in the linear case.
For concreteness, we consider the setup of [347] where the targets are given by a linear teacher, $y = Tx$, and the inputs are i.i.d Gaussian. We denote the input and target correlation matrices as follows:

$$\mathbb{E}\left[xx^\top\right] \equiv \Sigma_x \in \mathbb{R}^{D\times D}, \tag{H.29}$$

$$\mathbb{E}\left[TT^\top\right] \equiv \Sigma_y \in \mathbb{R}^{C\times C} \tag{H.30}$$

If the batch size is large enough, one can write $x_t x_t^\top = \mathbb{E}\left[xx^\top\right] = \Sigma_x$. Hence the dynamics of Eq. 10.9 become:

$$\delta W_1^t = -\eta(F_1 e_t)x_t^T = \eta F_1 T x_t x_t^\top = \eta F_1 T \Sigma_x \tag{H.31}$$

$$\delta W_2^t = -\eta(F_2 e_t)(W_1 x_t)^\top = \eta F_2 T \Sigma_x W_1^\top \tag{H.32}$$

$$= \eta^2 F_2 \left(T\Sigma_x^2 T^\top\right) F_1^\top \tag{H.33}$$

$$\delta W_3^t = -\eta(F_3 e_t)(W_2 W_1 x_t)^\top = \eta F_3 T \Sigma_x W_1^\top W_2^\top \tag{H.34}$$

$$= \eta^3 F_3 \left(T\Sigma_x^2 T^\top\right)\left(T\Sigma_x^2 T^\top\right) F_2^\top \tag{H.35}$$

From which we easily deduce $A_1^t = \eta T \Sigma_x t$, and the expression of the alignment matrices at all times:

$$A_{l\geq 2}^t = \eta^l \left(T\Sigma_x^2 T^\top\right)^{l-1} t \tag{H.36}$$

As we saw, GA depends on how well-conditioned the alignement matrices are, i.e. how different it is from the identity. To examine deviation from identity, we write $\Sigma_x = \mathbb{I}_D + \tilde{\Sigma}_x$ and $\Sigma_y = \mathbb{I}_C + \tilde{\Sigma}_y$, where the tilde matrices are small perturbations. Then to first order,

$$A_{l\geq 2}^t - I_C \propto (l-1)\left(\tilde{\Sigma}_y + 2T\tilde{\Sigma}_x T^\top\right) \tag{H.37}$$

Here we see that GA depends on how well-conditioned the input and target correlation matrices Σ_x and Σ_y are. In other words, if the different components of the inputs or the targets are correlated or of different variances, we expect GA to be hampered, observed in Sec. 10.5. Note that due to the $l-1$ exponent, we expect poor conditioning to have an even more drastic effect in deeper layers.
Notice that in this DRTP setup, the norm of the weights grows linearly with time, which makes DRTP inapplicable to regression tasks, and over-confident in classification tasks. It is clear in this case the the first layer learns the teacher, and the subsequent layers try to passively transmit the signal.

H.5 Details about the experiments

H.5.1 Direct Feedback Alignment implementation

We build on the Pytorch implementation of DFA implemented in [351], accessible at https://github.com/lightonai/dfa-scales-to-modern-deep-learning/tree/master/TinyDFA. Note that we do not use the shared feedback matrix trick introduced in this chapter. We sample the elements of the feedback matrix F_l from a centered uniform distribution of scale $1/\text{width}$.

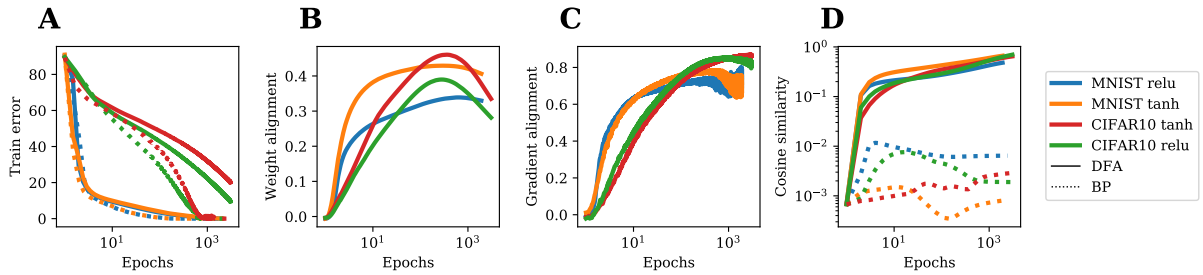
H.5.2 Experiments on realistic datasets

We trained 4-layer MLPs with 100 nodes per layer for 1000 epochs using vanilla SGD, with a batch size of 32 and a learning rate of 10^{-4}. The datasets considered are MNIST and CIFAR10, and the activation functions are Tanh and ReLU.
We initialise the networks using the standard Pytorch initialization scheme. We do not use any momentum, weight decay, dropout, batchnorm or any other bells and whistles. We downscale all images to 14×14 pixels to speed up the experiments. Results are averaged over 10 runs.
For completeness, we show in Fig. H.2 the results in the main text for 4 different levels of label corruption. The transition from Alignment phase to Memorisation phase can clearly be seen in all cases from the drop in weight alignment. Three important remarks can be made:

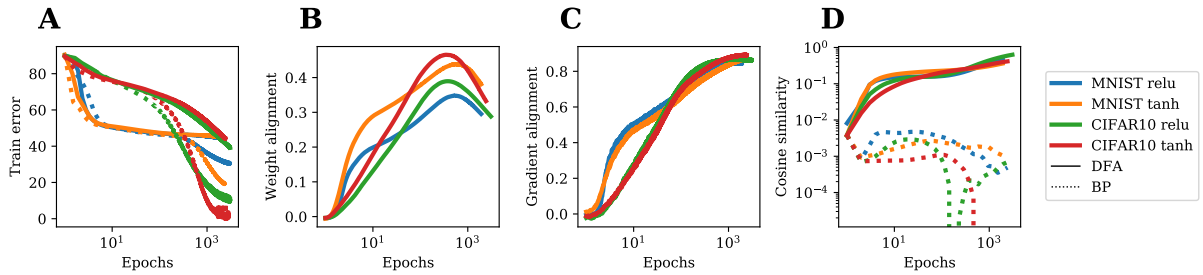
- **Alignment phase**: Increasing label corruption slows down the early increase of weight alignment, as noted in Sec. 10.5.1.
- **Memorization phase**: Increasing label corruption makes the datasets harder to fit. As a consequence, the network needs to give up more weight alignment in the memorization phase, as can be seen from the sharper drop in the weight alignment curves.
- **Transition point**: the transition time between the Alignement and Memorization phases coincides with the time at which the training error starts to decrease sharply (particularly at high label corruption), and is hardly affected by the level of label corruption.

H.5.3 Experiment on the structure of targets

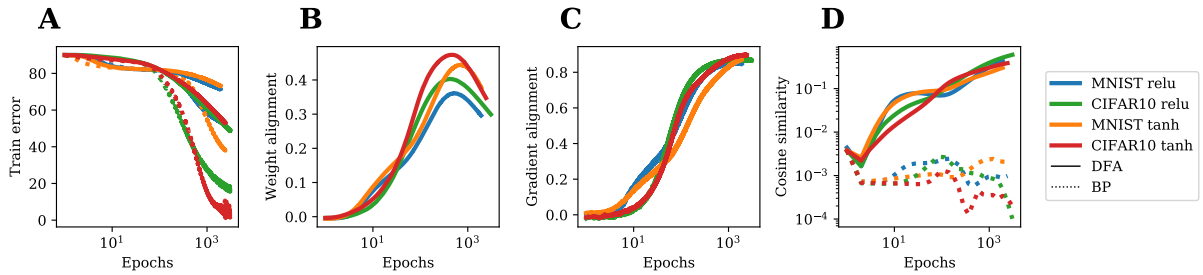
We trained a 3-layer linear MLP of width 100 for 1000 epochs on the synthetic dataset described in the main text, containing 10^4 examples. We used the same hyperparameters as for the experiment on nonlinear networks. We choose 5 values for α and β: 0.2, 0.4, 0.6, 0.8 and 1.
In Fig. H.3, we show the dynamics of weight alignment for both ReLU and Tanh activations. We again see the Align-then-Memorise process distinctly. Notice that decreasing α and β hampers both the mamixmal weight alignment (at the end of the alignment phase) and the final weight alignment (at the end of the memorisation phase).



(a) No label corruption

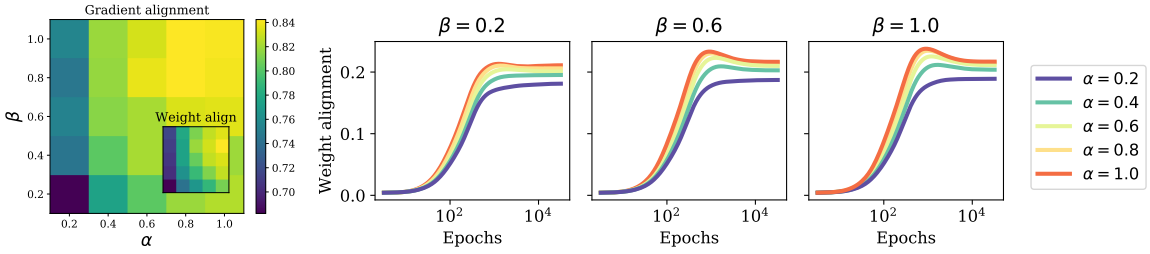


(b) 50% label corruption

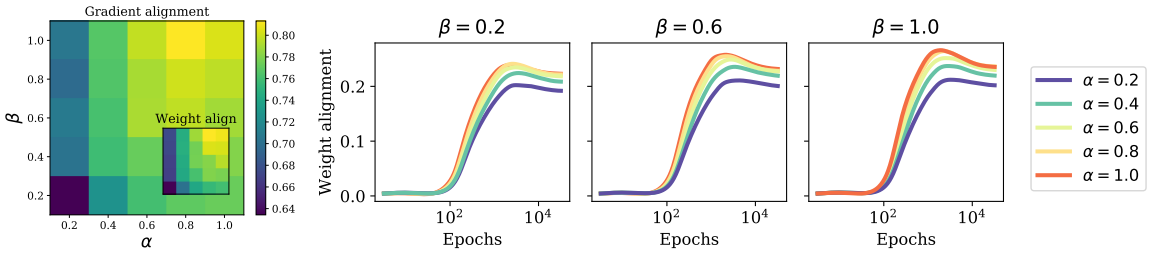


(c) 90% label corruption

Figure H.2: Effect of label corruption on training observables. **A**: Training error. **B** and **C**: Weight and gradient alignment, as defined in the main text. **D**: Cosine similarity of the weight during training.



(a) ReLU



(b) Tanh

Figure H.3: WA is hampered when the output dimensions are correlated ($\beta < 1$) or of different variances ($\alpha < 1$).

Appendix I

Optimal Learning Rate Schedules in Non-Convex Optimization

I.1 Dynamics of the convex model

Here we give additional details and steps in the computations on the convex model of Sec. 9.2. The loss function is given by $\mathcal{L}(x) = \frac{\kappa}{2}x^2$. Integrating the Langevin equation (Eq. 9.1) from t_0 to t for x yields:

$$x(t) = \underbrace{x(t_0)e^{-\kappa\int_{t_0}^{t}\mathrm{d}\tau\eta(\tau)}}_{\bar{x}(t)} + \underbrace{\int_{t_0}^{t}\mathrm{d}t' e^{-\kappa\int_{t_0}^{t'}\mathrm{d}t\tau\eta(\tau)}\eta(t')\xi(t')}_{\delta x(t)}\,. \tag{I.1}$$

In order to obtain a typical realisation of the loss which does not depend on the optimisation noise ξ, we take the expectation over ξ. This gives for the loss $\mathcal{L}$:

$$\langle\mathcal{L}(t)\rangle = \frac{\kappa}{2}\left(\langle\bar{x}(t)^2\rangle + \langle\delta x(t)^2\rangle + 2\underbrace{\langle\bar{x}(t)\delta x(t)\rangle}_{0}\right) \tag{I.2}$$

$$= \frac{\kappa}{2}\left(\underbrace{x(t_0)^2 e^{-2\kappa\int_{t_0}^{t}\mathrm{d}\tau\eta(\tau)}}_{\bar{\mathcal{L}}(t)} + \underbrace{2T\int_{t_0}^{t}\mathrm{d}t'\eta(t')^2 e^{-2\kappa\int_{t'}^{t}\mathrm{d}\tau\eta(\tau)}}_{\delta\mathcal{L}(t)}\right) \tag{I.3}$$

The first term is an optimisation term while the second is the contribution of the noise inherent to the optimisation algorithm. Thus, to converge to the solution as quickly as possible, one has to find the trade-off between decreasing the impact of the noise term while not slowing down optimisation excessively. The ideal schedule is determined by requiring these two effects are comparable. Defining $\eta(t) = \eta_0/t$, we obtain

$$\bar{\mathcal{L}}(t) \propto e^{-2\eta_0\kappa\log(t)} \propto t^{-2\eta_0\kappa} \tag{I.4}$$

$$\delta\mathcal{L}(t)\int_{t_0}^{t}\mathrm{d}t'\frac{1}{t'^2}\left(\frac{t'}{t}\right)^{2\eta_0\kappa} \propto 1/t. \tag{I.5}$$

If $\eta_0 > 1/2\kappa$, the loss is dominated by the noise term $\delta\mathcal{L}$ and decays as $1/t$. If $\eta_0 < 1/2\kappa$, the loss is dominated by the optimization term $\delta\mathcal{L}$ and decays as $t^{-2\eta_0\kappa}$.

I.2 Dynamics of the Sherrington-Kirkpatrick model

In this section, we provide derivations for the results obtained in the SK model.

I.2.1 Unplanted model

The loss function is given by:

$$\mathcal{L}(x) = -\frac{1}{\sqrt{N}} \sum_{i<j}^{N} J_{ij} x_i x_j. \tag{I.6}$$

Solving the dynamics Following [304], we express the spin configurations in the eigenbasis of J and define $x_\mu = {}^{x\cdot J_\mu}/\sqrt{N}$ as the projection of x onto the eigenvector J_μ. x_μ evolves as:

$$\frac{\partial x_\mu(t)}{\partial t} = \eta(t) \left[(\mu - z(t)) x_\mu(t) + \xi_\mu(t) \right]. \tag{I.7}$$

Integrating this equation yields again two terms, one related to the optimisation and the second related to the noise:

$$\begin{aligned} x_\mu(t) = x_\mu(0)\, e^{-\int_0^t d\tau \eta(\tau)(\mu - z(\tau))} \\ + \int_0^t dt'' e^{-\int_{t''}^t d\tau' \eta(\tau')(z(\tau') - \mu)} \eta(t'') \xi_\mu(t''). \end{aligned} \tag{I.8}$$

In the $t \to \infty$ limit, a non-exploding $\bar{x}_\mu$ requires $\mu - z(t)$ to be negative for all μ in the support of ρ, implying $z(t) < 2$. We must also impose $z(t) \to_{t\to\infty} 2$, otherwise $x_\mu(t) \to 0\ \forall \mu$, in contradiction with the spherical constraint. To comply with these two requirements we define $z(t) = 2 - f(t)$, with $f(t) \to_{t\to\infty} 0$.
In the constant learning rate setup $\eta(t) = 1$ we know from [304] that $f(t) = 3/(4t)$. With $\eta(t) = \eta_0/t^\beta$, a natural ansatz is $f(t) = c/t^{1-\beta}$. To determine c, we impose the spherical constraint:

$$\begin{aligned} 1 &= \langle \int \frac{\mathrm{d}\mu}{N} \rho(\mu) x_\mu(t)^2 \rangle \\ &= t^{2c\eta_0} \int_{-2}^{2} \mathrm{d}\mu \sqrt{4 - \mu^2} e^{2\eta_0(\mu-2)t^{1-\beta}} \\ &= t^{2c\eta_0 - 3(1-\beta)/2} \int_0^\infty \mathrm{d}\epsilon \sqrt{2\epsilon} e^{-2\eta_0 \epsilon} \propto t^{2c\eta_0 - 3(1-\beta)/2} \Rightarrow c = \frac{3(1-\beta)}{4\eta_0}. \end{aligned}$$

For $\beta = 1$, we instead use the ansatz $z(t) = 2 - c/\log(t)$:

$$\begin{aligned} 1 &= \int_{-2}^{2} \mathrm{d}\mu \sqrt{4 - \mu^2} e^{2\eta_0(\mu-2)\log t} e^{2c\eta_0 \log\log t} \\ &= (\log t)^{2c\eta_0 - 3/2} \int_0^\infty \mathrm{d}\epsilon \sqrt{2\epsilon} e^{-2\eta_0\epsilon} \propto (\log t)^{2c\eta_0 - 3/2} \Rightarrow c = \frac{3}{4\eta_0}. \end{aligned}$$

Hence, the scaled loss $\ell = \mathcal{L}/N$ converges to the ground state (global minimum) $\ell_{GS} = -1$ as a sum of power-laws:

$$\ell(t) - \ell_{GS} = \frac{\eta_0 T}{2t^\beta} + \begin{cases} \frac{3(1-\beta)}{8\eta_0 t^{1-\beta}}, & \beta < 1 \\ \frac{3}{8\eta_0 \log t}, & \beta = 1 \end{cases}. \tag{I.9}$$

Dependency on the spectrum of J One may naturally ask whether our conclusions are affected by changing the spectrum of the coupling matrix J. Notice that the key to solving the self-consistent equation is the behavior of the spectrum near its right edge. For the semi-circle law considered here, the right edge of the spectrum behaves as a square root. This law applies to a rather wide range of random matrix ensembles. Besides, many other common spectral densities, such as the Marcenko-Pastur law, also exhibit a same square root behavior on their right edge. Hence we expect our results to hold for a wide range of random matrix ensembles.

I.2.2 Planted model

The loss function is given by:

$$\mathcal{L}(x) = -\frac{N}{2}m^2 - \frac{\Delta}{\sqrt{N}}\sum_{i<j}^{N} J_{ij}x_i x_j. \tag{I.10}$$

Solving the dynamics Again we choose $\eta(t) = \eta_0/t^\beta$ and consider the high signal-to-noise setting, $\Delta < \frac{1}{2}$. Writing $z(t) = 1 - f(t)$, we obtain:

$$\begin{aligned}
1 =& \langle \int \frac{\mathrm{d}\mu}{N}\rho(\mu)x_\mu(t)^2\rangle && \text{(I.11)}\\
=& \frac{N-1}{N}\int_{-2}^{2}\mathrm{d}\mu\rho_{sc}(\mu)e^{2\int_{t_0}^{t}\mathrm{d}\tau\eta(\tau)(\mu-1)}e^{2\int_{t_0}^{t}\mathrm{d}\tau\eta(\tau)f(t)} + \frac{1}{N}e^{2\int_{t_0}^{t}\mathrm{d}\tau\eta(\tau)f(t)} && \text{(I.12)}\\
=& e^{2\int_{t_0}^{t}\mathrm{d}\tau\eta(\tau)f(\tau)}\left(\underbrace{e^{-2\eta(t)(1-2\Delta)}\int_{-2}^{2}\mathrm{d}\mu\rho_{sc}(\mu/\Delta)e^{2\eta(t)(\mu-2\Delta)t}}_{A(t)} + \frac{1}{N}\right) && \text{(I.13)}
\end{aligned}$$

The expression above involves two terms. The first is of order one but decays exponentially over time; using results above, we obtain that

$$A(t) \sim t^{-3(1-\beta)/2}e^{-2\eta_0\kappa t^{1-\beta}}. \tag{I.14}$$

Hence, there is a crossover time at which the first term becomes smaller than the second term, given by:

$$A(t) \sim 1 \Rightarrow t_{\text{cross}} = \left(\frac{\log N}{2\eta_0\kappa}\right)^{\frac{1}{1-\beta}} \tag{I.15}$$

Before t_{cross}, the signal is not detected and we have as before $z(t) = 2\Delta - c/t^{1-\beta}$.
After t_{cross}, we have $A(t) \ll 1/N$. Multiply Eq. I.13 by N and taking the log, we obtain:

$$\begin{aligned}
\log N &= 2\int_{t_0}^{t}\mathrm{d}\tau f(\tau)t^{-\beta} + \log\left(1 + NA(t)\right) && \text{(I.16)}\\
&\sim 2\int_{t_0}^{t}\mathrm{d}\tau f(\tau)t^{-\beta} + NA(t) && \text{(I.17)}
\end{aligned}$$

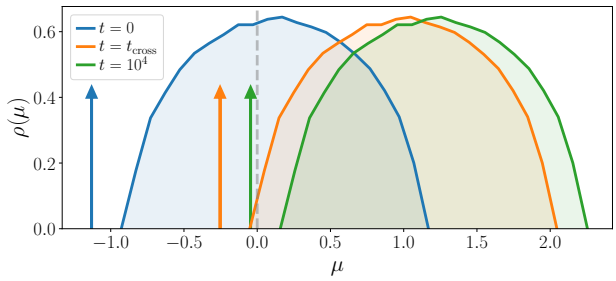


Figure I.1: **The landscape becomes convex at the crossover time.** *Parameters:* $N = 3000$, $\eta_0 = 0.1$.

Taking the derivative with respect to t, we find the following asymptotics for late times:

$$f(t) \sim \frac{-NA'(t)t^\beta}{2} \sim t^{-5(1-\beta)/2} e^{-2\eta_0 \kappa t^{1-\beta}} \tag{I.18}$$

Hence,

$$\ell(t) - \ell_{GS} = \frac{\eta_0 T}{2t^\beta} + \frac{1}{2} f(t) \tag{I.19}$$

with $\ell_{GS} = 1$. As previously, it is straightfoward to extend this to the setup $\beta = 1$, for which we obtain $f(t) \sim t^{-2\eta_0 \kappa}$.

Curvature analysis As before, the spectrum of interest to study the landscape is that of H shifted to the right by the spherical constraint $z(t)$, depicted in Fig. I.1. The crossover time t_{cross} corresponds to the time at which the left edge of the semi-circle reaches 0. Thanks to the presence of the signal, the dynamics do not stop at this point; they continue until the eigenvalue corresponding to the signal reaches zero (which as achieved at $t \to \infty$). After the t_{cross}, the landscape becomes locally convex: the only negative eigenvalue is in the direction of the signal. Due to the spherical constraint, the effective Hessian (of dimension $N - 1$) does not feel this negative eigenvalue when x is close to $x^\star$.

I.3 Dynamics of the p-spin model

I.3.1 Rescaling the temperature

Introducing a learning rate schedule is equivalnt to changing the "clock" directly in the Langevin (Eq. 9.1) as $\mathrm{d}\tilde{t} = \eta(t)\,\mathrm{d}t$. Then, for $\beta < 1$ we have:

$$\delta(\tilde{t})\,\mathrm{d}\tilde{t} = \delta(t)\,\mathrm{d}t \Rightarrow \delta(t) = \delta(\tilde{t}) \left(\frac{\eta_0}{1-\beta}\right)^{\frac{\beta}{1-\beta}} \tilde{t}^{\frac{-\beta}{1-\beta}} \tag{I.20}$$

The Langevin equation becomes:

$$\frac{\mathrm{d}x_i(t)}{\mathrm{d}t} = -\left(\frac{\partial \mathcal{L}(x, x^\star)}{\partial x_i} + \xi_i(t) + z(t) x_i(t)\right), \qquad \langle \xi(t)\xi(t')\rangle = 2T\tilde{\eta}_0 \tilde{t}^{-\beta/1-\beta} \delta(t - t'), \tag{I.21}$$

where we defined $\tilde{\eta}_0 = \left(\frac{\eta_0}{1-\beta}\right)^{\frac{\beta}{1-\beta}}$. This equation reveals that the process optimised with a varying learning rate is equivalent to a process at an effective temperature:

$$\tilde{T} = \tilde{\eta}_0 \tilde{t}^{\frac{-\beta}{1-\beta}} T \tag{I.22}$$

This equation corresponds to the physical protocol in which the temperature $\tilde{T}$ is annealed as a power law. As we show for the p-spin model in the next section, the solution is governed by a speed-noise trade-off.

I.3.2 Application to the p-spin model

For the p-spin model the loss can therefore be written:

$$\mathcal{L}(t;\tilde{T}) = \frac{-N}{p}\left(\tilde{z}(t) - \tilde{T}\right) = \mathcal{L}(t;T=0) + \mathcal{L}_{th}(\tilde{T}) \tag{I.23}$$

where we assumed that, at all times, the temperature dependent contribution to the loss has time to equilibrate in the threshold states.
To find $\mathcal{L}_{th}(\tilde{T})$, we assume that, since we are looking at long times, we have $\tilde{T} \ll 1$. We can consider the loss by performing an expansion around the $\tilde{T} = 0$ minimum i.e. considering that the motion is oscillatory around the minimum. At $T = 0$, the threshold overlap is given by $q_{th} = 1$. At $T \ll 1$, we thus write $q = 1 - \chi T$. Performing a similar matching argument as the one of [303], described in more details in Sec. I.4.4, we find that the close to the threshold, the loss is given by Eq. I.68:

$$\mathcal{L}_{th}(T) = -\frac{1}{p}\left[\sqrt{(p-1)q^{p-2}} + \frac{1}{T}\sqrt{\frac{2}{p}}\left(1 - q^{p-1}\right)\right] \tag{I.24}$$

In addition, we can expand the threshold overlap solution around $T = 0$ [307]:

$$\begin{aligned} q_{th}^{p-2}(1-q_{th})^2 &= \frac{T^2}{p-1} \\ \Rightarrow \chi &= \sqrt{\frac{1}{(p-1)}} \end{aligned} \tag{I.25}$$

Replacing this solution in Eq. I.24, we find:

$$\mathcal{L}_{th}(T) = \underbrace{-\frac{\sqrt{4(p-1)}}{p}}_{\mathcal{L}_{th}(T=0)} + \underbrace{\frac{p-2}{p}T}_{\propto T} \tag{I.26}$$

We see that $\mathcal{L}_{th}(T)$ is composed of a constant term, which is the same as the threshold loss defined in Eq. 9.16 and a term scaling linearly with T. Thus:

$$\mathcal{L}_{th}(\tilde{T}) - \mathcal{L}_{th} \propto \tilde{T} \propto t^{-\beta/1-\beta}. \tag{I.27}$$

We find again the two competing term in the speed of optimisation. On the one hand, the noiseless term, which is the same as the zero temperature loss, decays as $\mathcal{L}(t;T=0) \propto t^{-\gamma}$. On the other hand, the temperature dependent term, which blocks the dynamics at loss $\mathcal{L}_{th}(\tilde{T})$, which decays as $t^{-\beta/1-\beta}$. The loss decays as $t^{-\min\left(\gamma, \frac{\beta}{1-\beta}\right)}$. Equaling the two exponents gives the optimal value of $\beta_{\text{opt}} = \frac{2}{5}$.

I.4 Dynamics of the Spiked Matrix-Tensor model

I.4.1 Derivation of the PDE equations

For simplicity, we detail the derivation of the *p-spin model* without the spike as studied in Sec. 9.3.2 in the case $p = 3$. The derivation for the full spiked tensor model is similar and can be found in [307]. The Langevin equation for each spin x is given by:

$$\dot{x}_i(t) = -z(t)\eta(t) - \eta(t)\partial_{x_i}\mathcal{L} + \eta(t)\xi_i(t) \tag{I.28}$$

where $\boldsymbol{\xi} \in \mathbb{R}^N$ is the Langevin noise with distribution $\langle \xi_i(t) \rangle = 0$ and $\langle \xi_i(t)\xi_j(t') \rangle = 2T\delta_{ij}\delta(t-t')$. The solution x to the Langevin equation depends on the realisation of the noise. We can obtain a probability distribution over x given the distribution of ξ by considering the expectation of an observable $A(t)$:

$$\langle A(x) \rangle = \int D\xi P(\xi) A\left(x_\xi\right) = \int dx \left[\int D\xi P(\xi) \delta\left(\dot{x} + \eta(t)\partial_x \mathcal{L} - \eta(t)\xi\right) \right] A(x) = \int dx P(x) A(x) \tag{I.29}$$

We are now interested in considering $P(x)$ when averaged over the quenched disorder J. We therefore resort to:

$$\begin{aligned}
1 \equiv Z &= \int DxP(x) \\
&= \int DxD\hat{x}D\xi \exp\left[-\frac{1}{2}\int dtdt' \xi(t) D_0^{-1}\left(t,t'\right)\xi\left(t'\right) + i\int dt\hat{x}(t)\left(\partial_t x + \eta(t)\partial_x\mathcal{L}\right) - i\int dt\eta(t)\hat{x}(t)\xi(t) \right] \\
&= \int DxD\hat{x} \exp\left[-\frac{1}{2}\int dtdt' \hat{x}(t)\eta(t) D_0\left(t,t'\right)\hat{x}\left(t'\right)\eta(t') + i\int dt\hat{x}(t)\left(\partial_t x + \eta(t)\partial_x\mathcal{L}\right) \right] \\
&= \int DxD\hat{x} \exp\left[S(x,\hat{x})\right]
\end{aligned} \tag{I.30}$$

where we defined $D_0(t,t') = 2T\delta(t-t')$. Crucially, $S(x,\hat{x})$ acts as a generating functional and allows to obtain correlation functions by a term $\int dt\hat{x}(t)h(t) + x(t)\hat{h}(t)$. We can thus define

$$\langle x(t)\hat{x}(t') \rangle = \partial_{h(t')}\langle x(t) \rangle \equiv R(t,t') \qquad \langle x(t)x(t') \rangle = \partial_{\hat{h}(t')}\langle x(t) \rangle \equiv C(t,t') \tag{I.31}$$

We now want to average the partition function over the quenched disorder Z. We note that the only time depend term in the exponent is $i\hat{x}(t)\eta(t)\partial_x\mathcal{L}$. We thus have to compute:

$$\overline{e^{i\hat{x}(t)\eta(t)\partial_x\mathcal{L}}} \equiv e^{\Delta(x,\hat{x})} \tag{I.32}$$

The average over the disorder will induce corrections both to the propagator D_0 and to the interaction term $\hat{x}(t)x(t')$ i.e. $\Delta(x,\hat{x}) = -\frac{1}{2}\hat{x}D_1(x,\hat{x})\hat{x} + i\hat{x}\mathcal{L}_1(x,\hat{x})$. By performing the average we obtain:

$$\begin{aligned}
\overline{i\hat{x}_i(t)\eta\partial_{x_i}\mathcal{L}} &= \int \prod_{i>k>l} dJ_{ikl} \exp\left\{ -\frac{1}{2}J_{ikl}^2 - \sqrt{\frac{(p-1)!}{N^{p-1}}}J_{ikl}\int dt\eta(t)\left[i\hat{x}_i x_k x_l + x_i i\hat{x}_k x_l + x_i x_k i\hat{x}_l \right] \right\} \\
&= \exp\left\{ \int \frac{dtdt'}{2N^{p-1}}\eta(t)\eta(t')\left[(i\hat{x}\cdot i\hat{x})(x\cdot x)^{p-1} + (p-1)(i\hat{x}\cdot x)(x\cdot i\hat{x})(x\cdot x)^{p-2} \right] \right\}
\end{aligned} \tag{I.33}$$

where we introduced the notation $x \cdot x \equiv \sum_{i=1}^{N} x_i(t)x_i(t')$. We now introduce dynamical overlaps Q_1, Q_2, Q_3 and Q_4 as:

$$\begin{aligned}\overline{e^{i\hat{x}_i(t)\eta\partial_{x_i}\mathcal{L}}} = &\int DQ\delta\left(NQ_1 - \sum_k i\hat{x}_k(t)i\hat{x}_k(t')\right)\delta\left(NQ_2 - \sum_k x_k(t)x_k(t')\right)\\ &\cdot\delta\left(NQ_3 - \sum_k i\hat{x}_k(t)x_k(t')\right)\delta\left(NQ_4 - \sum_k x_k(t)i\hat{x}_k(t')\right)\\ &\cdot\exp\left\{\frac{N}{2}\int dtdt'\eta(t)\eta(t')\left[Q_1(t,t')\,Q_2(t,t')^{p-1} + (p-1)Q_3(t,t')\,Q_4(t,t')\,Q_2(t,t')^{p-2}\right]\right\}\end{aligned} \tag{I.34}$$

We can easily see that we have the correspondence $Q_1(t,t') = 0, Q_2(t,t') = C(t,t'), Q_3(t,t') = R(t',t)$ and $Q_4(t,t') = R(t,t')$. By using the exponential form of the delta function and solving the fix point equations for the conjugate fields $\hat{Q}_1$, $\hat{Q}_2$, $\hat{Q}_3$ and $\hat{Q}_4$ we find:

$$\begin{cases} i\hat{Q}_1 = \frac{1}{2}\eta(t)\eta(t')Q_2^{p-1} \\ i\hat{Q}_2 = \frac{p-1}{2}\eta(t)\eta(t')Q_1Q_2^{p-2} + \frac{(p-1)(p-2)}{2}\eta(t)\eta(t')Q_3Q_4Q_2^{p-3} \equiv 0 \\ i\hat{Q}_3 = \frac{p-1}{2}\eta(t)\eta(t')Q_4Q_2^{p-2} \\ i\hat{Q}_4 = \frac{p-1}{2}\eta(t)\eta(t')Q_3Q_2^{p-2} \end{cases} \tag{I.35}$$

From the definition of the $\hat{Q}$'s we find the new term in the generating functional as:

$$\Delta = \sum_k \int dtdt'\eta(t)\eta(t')\left\{-\frac{1}{2}C(t,t')^{p-1}\,\hat{x}_k(t)\hat{x}_k(t') - (p-1)R(t,t')\,C(t,t')^{p-2}\,i\hat{x}_k(t)x_k(t')\right\} \tag{I.36}$$

This allows us to write an effective Langevin equation for a scalar degree of freedom x:

$$\dot{x}(t) = -z(t)\eta(t)x(t) + \eta(t)(p-1)\int dt''\eta(t')R(t,t'')\,C(t,t'')^{p-2}\,\sigma(t'') + \eta(t)\tilde{\xi}(t), \tag{I.37}$$

with:

$$\langle\tilde{\xi}(t)\tilde{\xi}(t')\rangle = 2T\delta(t-t') + C^{p-1}(t,t'). \tag{I.38}$$

In order to write down a set of PDE's for R and C, note the useful relations:

$$\begin{aligned}\langle\frac{\partial x(t)}{\partial\xi(t')}\rangle &= -i\langle x(t)\hat{x}(t')\rangle \\ \langle x(t)\xi(t')\rangle &= 2T\eta(t')R(t,t^{'}) \\ \langle\tilde{\xi}(t_1)x(t_2)\rangle &= 2T\eta(t_1)R(t_1,t_2) + \int dt''\eta(t'')R(t'',t_2)C^{p-1}(t'',t_1)\end{aligned} \tag{I.39}$$

We therefore find:

$$\begin{aligned}\frac{\partial R(t_1,t_2)}{\partial t_1} &= \left\langle\frac{\delta\dot{x}(t_1)}{\delta\tilde{\xi}(t_2)}\right\rangle \\ &= -z(t_1)\,\eta(t_1)R(t_1,t_2) + \eta(t_1)\delta(t_1,t_2) \\ &\quad + (p-1)\eta(t_1)\int_{t_2}^{t_1} dt''\eta(t'')R(t_1,t'')\,C^{p-2}(t_1,t'')\,R(t'',t_2)\end{aligned} \tag{I.40}$$

$$\begin{aligned}\frac{\partial C\left(t_1,t_2\right)}{\partial t_1} &= \langle \dot{x}\left(t_1\right) x\left(t_2\right)\rangle \\ &= -\eta(t_1) z\left(t_1\right) C\left(t_1,t_2\right) + 2T\eta(t_1)^2 R(t_1,t_2) \\ &\quad + (p-1)\eta(t_1)\int_{-\infty}^{t_1} dt''\eta(t'')R\left(t_1,t''\right) C^{p-2}\left(t_1,t''\right) C\left(t'',t_2\right) \\ &\quad + \eta(t_1)\int dt''\eta(t'')R(t'',t_2)C^{p-1}(t'',t_1)\end{aligned} \tag{I.41}$$

The equation for $z(t)$ is given by differentiation $C(1,1)=1$, i.e. $\left[\partial_t C\left(t,t'\right)+\partial_{t'}C\left(t,t'\right)\right]_{t,t'=s}=0$:

$$z(t_1) = T\eta(t_1) + p\int dt_2\eta(t_2)R(t_2,t_1)C^{p-1}(t_2,t_1). \tag{I.42}$$

The loss at all times is found by using the Ito identity:

$$\frac{1}{N}\frac{d}{dt}\sum_i x_i^2(t) = \frac{2}{N}\sum_i x_i(t)\dot{x}_i(t) + 2 \tag{I.43}$$

which yields:

$$\mathcal{L}(t) = \frac{N}{p}\left(T\eta(t) - z(t)\right) \tag{I.44}$$

Spiked matrix-tensor model The derivation of the PDEs describing the dynamics of C, R and z in the spiked matrix-tensor model are similar as the ones for the p-spin. In addition, one also needs to keep track of the evolution of the overlap of the estimate with the signal i.e. the magnetisation $m = {}^{x\cdot x^\star}/_N$. Using the same method as before we find:

$$\begin{aligned}&\frac{\partial}{\partial t}C\left(t,t'\right) = -z(t)\eta(t)C\left(t,t'\right) + \eta(t)Q'(m(t))m\left(t'\right) + \eta(t)\int_0^t \eta(t')R\left(t,t''\right)Q''\left(C\left(t,t''\right)\right)C\left(t',t''\right)dt'' \\ &\quad + \eta(t)\int_0^{t'}\eta(t')R\left(t',t''\right)Q'\left(C\left(t,t''\right)\right)dt'' + 2T\eta(t)^2R(t,t') \\ &\frac{\partial}{\partial t}R\left(t,t'\right) = -z(t)\eta(t)R\left(t,t'\right) + \eta(t)\int_{t'}^t \eta(t')R\left(t,t''\right)Q''\left(C\left(t,t''\right)\right)R\left(t'',t'\right)dt'' + \delta(t-t')\eta(t) \\ &\frac{d}{dt}m(t) = -\eta(t)z(t)m(t) + \eta(t)Q'(m(t)) + \eta(t)\int_0^t \eta(t')R\left(t,t''\right)m\left(t''\right)Q''\left(C\left(t,t''\right)\right)dt'' \\ &z(t) = \eta(t)T + Q'(m(t))m(t) + \int_0^t \eta(t')R\left(t,t''\right)\left[Q'\left(C\left(t,t''\right)\right) + Q''\left(C\left(t,t''\right)\right)C\left(t,t''\right)\right]dt''\end{aligned} \tag{I.45}$$

where we defined $Q(x) = Q_p(x) + Q_2(x) = \frac{x^p}{p\Delta_p} + \frac{x^2}{2\Delta_2}$. The loss is related to $z(t)$ via:

$$z(t) = T\eta(t) - p\frac{\mathcal{L}_p}{N} - 2\frac{\mathcal{L}_2}{N}, \tag{I.46}$$

with $\mathcal{L}_2$, respectively $\mathcal{L}_2$ are the loss associated with the matrix, respectively tensor, channel.

The Langevin easy phase As explained in [303], one finds different phases in the two dimensional space spamed by the noise intensities Δ_2 and Δ_p. In the *Langevin easy* phase, a system initialised with a magnetisation $m \sim O(1/\sqrt{N})$ recovers the signal and converges to an overlap of order 1. It is delimited by $\Delta_2 < \Delta_2^*$, where Δ_2^* is the solution to the implicit equation:

$$\Delta_2 < \Delta_2^\star = \sqrt{\frac{\Delta_p}{(p-1)(1-\Delta_2^*)^{p-3}}}. \tag{I.47}$$

In contrast, in the *Langevin hard* and *Langevin impossible* phase, i.e. $\Delta_2 > \Delta_2^*$, the dynamics fail to recover the signal and remain at low magnetisation. More details in [307].

I.4.2 Derivation of the Ground-state Loss

In order to derive the ground state properties of the system, we resort to the replica method, developed in physics as a tool to deal with random systems. Using these tools, involves performing a mapping between the optimisation problem, an inference problem and a physical system. We can consider the estimator x as a guess on the planted signal $x^\star$ and y be the observations.The, using Bayes formula we can express the posterior probability of the estimator x given the observation y:

$$P[x|y] = \frac{1}{P[y]} P[x]P[y|x] \approx_{\beta=1} \frac{1}{P[y]} P[x]P[y|x]^{-\beta} = \frac{1}{Z(y)} \mathrm{e}^{-\beta\mathcal{L}}. \tag{I.48}$$

We can identify the last terms with a Gibbs distribution at temperature $\beta = 1/T$ and Z is a normalisation constant named the *partition function*. At $\beta = 1$, the posterior I.48 is the exact posterior of the problem. At $\beta \to \infty$, the distribution is dominated by the spin configuration minimising the loss, i.e. the maximum likely hood approximator of the problem. The partition function, and its logarithm the *free energy*:

$$\Phi = \frac{-1}{N} \log Z, \tag{I.49}$$

act as a generating functional. I.e. they encapsulate all the relevant information needed to describe of the system. Notably, all observables can be obtained by taking derivatives of it. In particular, the loss and the overlap with the signal are given by:

$$\begin{aligned} \mathcal{L} &= \frac{1}{N}\frac{1}{Z}\int_{\S^{N-1}} \mathcal{L} e^{-\beta\mathcal{L}} = \frac{-1}{N}\frac{\partial \log Z}{\partial\beta} = \frac{\partial\Phi}{\partial\beta} \\ m &= \frac{1}{N}\sum_{i=1}^{N}\frac{1}{Z}\int_{\S^{N-1}} x_i x_i^\star e^{-\beta\mathcal{L}+\boldsymbol{h}\cdot x}|_{\boldsymbol{h}=0} = -x^\star \cdot \partial_{\boldsymbol{h}}\Phi. \end{aligned} \tag{I.50}$$

The spiked tensor model is rendered more complex due to the randomness associated with the couplings. We need to evaluate the averaged logarithm of the partition function $\overline{\log Z}$ which is in general prohibitive. To deal with this problem, physics have developed the heuristic *replica method* based on the equality:

$$\overline{\log Z} = \lim_{n\to 0} \frac{\overline{Z^n} - 1}{n}. \tag{I.51}$$

In practice, one computes $\overline{Z^n}$ for $n \in \mathbb{N}$ and then extends the result to real n. The problem can be viewed as introducing n identical, replicated, copies of the system. As we will see, averaging over the

random couplings introduces correlation between the copies. $\overline{Z^n}$ can easily be evaluated as:

$$\begin{aligned}\overline{\mathcal{Z}^n} &= \mathop{\mathbb{E}}_{\substack{J_{i_1\dots i_p}\\ J_{ij}}} \int \prod_{a=1}^{n} \mathrm{e}^{\beta\sqrt{\frac{1}{p\Delta_p N}}\sum_{i_1,\dots,i_p} J_{i_1,\dots,i_p} x_{i_1}^{(a)}\dots x_{i_p}^{(a)} + \beta\sqrt{\frac{1}{2\Delta_2 N}}\sum_{i,j} J_{i,j} x_i^{(a)} x_j^{(a)} + N\beta\sum_i Q\left(\frac{x_i^{(a)} x_i^*}{N}\right)} \prod_{a=1}^{n} dx^{(a)} \\ &= \int_{\mathbb{S}^{n(N-1)}(\sqrt{N})} \mathrm{e}^{N\beta\sum_i Q\left(\frac{x_i^{(a)} x_i^*}{N}\right) + \frac{N\beta^2}{2} Q\left(\sum_{a,b=1}^{n}\sum_i \frac{x_i^{(a)} x_i^{(b)}}{N}\right)} \prod_{a=1}^{n} dx^{(a)}.\end{aligned} \tag{I.52}$$

where we introduced $Q(x) = {}^{x^2}/_{2\Delta_2} + {}^{x^p}/_{p\Delta_p}$. The second term in the exponent carries the interaction between the different copies obtained after averaging out the random couplings. It depends on the overlap $\boldsymbol{\mathcal{Q}}$ having entries $\mathcal{Q}_{ab} = \sum_i \frac{x_i^{(a)} x_i^{(b)}}{N}$. We associate the index $a = 0$ with the ground truth signal $x^\star$. Using the exponential representation of the Dirac delta function, we introduce the overlap matrix into the partition function. After some manipulation we obtain:

$$\overline{Z^n} = \int \mathrm{e}^{N\beta S(\boldsymbol{\mathcal{Q}})} \tag{I.53}$$

$$\beta S(\boldsymbol{\mathcal{Q}}) = \frac{1}{2}\log\det\boldsymbol{\mathcal{Q}} + \beta^2 \sum_{a,b=1}^{n} Q(\mathcal{Q}_{ab}) + \beta\sum_{a=1}^{n} Q(\mathcal{Q}_{a0}). \tag{I.54}$$

The factor N in the exponential in the integrand, implies that in the $N\to\infty$ limit, the integral is dominated by the matrix $\mathcal{Q}$ maximising the action S. In order to progress, we make a replica symmetric ansatz[1]: i.e. we assume the different systems have overlaps q between each other and m with the ground truth. This imposes a matrix $\mathcal{Q}$ has the form:

$$\mathcal{Q} = \begin{pmatrix} 1 & m & m & m \\ m & 1 & q & q \\ m & q & 1 & q \\ m & q & q & 1 \end{pmatrix}. \tag{I.55}$$

Replacing this overlap matrix in I.54 and taking $n\to 0$, we obtain:

$$\beta S_{\mathrm{RS}}(q,m) = n\left\{\frac{1}{2}\frac{q-m^2}{1-q} + \frac{1}{2}\log(1-q) + \frac{\beta^2}{2}Q(1) - \frac{\beta^2}{2}Q(q) + \beta Q(m)\right\} \tag{I.56}$$

We now maximise S with respect to m and q and obtain the saddle point equations:

$$\frac{S_{\mathrm{RS}}(q,m)}{\partial m} = \frac{-m}{1-q} + \beta Q'(m) \tag{I.57}$$

$$\frac{S_{\mathrm{RS}}(q,m)}{\partial q} = \frac{q-m^2}{(q-1)^2} + \beta^2 Q'(q) \tag{I.58}$$

The expression of the loss as a function of the overlaps m and q is given by using Eq. I.50:

$$\mathcal{L}(m,q) = -\beta(Q(1) - Q(q)) + Q(m) \tag{I.59}$$

By evaluating the above at the solutions Eqs. I.58, we obtain the ground state loss at a given temperature.

[1] Since we only consider the Langevin easy phase, where there is no ergodicity breaking, we do not need to consider a 1RSB ansatz.

$T = 1$ **solution** At $T = 1$ (i.e. $\beta = 1$) the posterior Eq. I.48 is exact and we can use the Nishimori identity stating that the distribution of the estimator is the same as the one of the signal implying $m = q$. Replacing the identity in Eq. I.58 and in Eq. I.59 we have:

$$m = (1 - m)Q'(m), \mathcal{L}_{\text{gs}}^{T=1} = -Q(1). \tag{I.60}$$

$T = 0$ **solution** We can think of the 0 temperature system (i.e. $\beta \to \infty$) as physical system coupled to a thermal bath. As the temperature goes to 0, all particles collapse to a point at the minimum of the loss. Thus, the overlap tends to 1. However, we check that Eqs. I.58 are singular at $q = 1$. To properly take the limit, we perform a linear expansion in the temperature by replacing $q = 1 - \chi T$ in the equations and linearising in T. We then obtain the equation for m:

$$\begin{aligned} \chi &= \sqrt{\frac{1 - m^2}{Q'(1)}} \\ m &= \chi Q'(m) \end{aligned} \tag{I.61}$$

and the ground state loss:

$$\mathcal{L}_{\text{gs}}^{T \to 0} = (-Q(m) - \chi Q'(1)). \tag{I.62}$$

I.4.3 Additional results on the optimal learning rate schedule in the SMT model

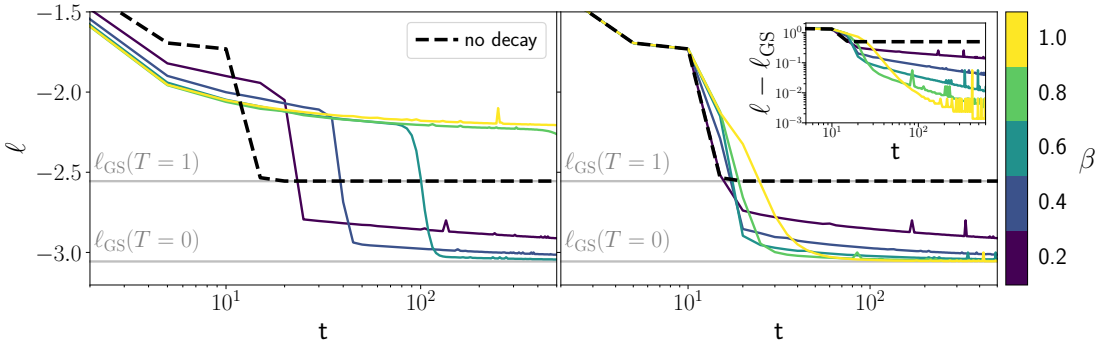


Figure I.2: **Emergence of a crossover time in the SMT model.** By fixing β from start, a randomly initialised system will remain stuck at *threshold* states of 0 overlap until t_{cross} which is minimal for $\beta = 0$. Higher β, allow to reach lower loss solutions but require much longer to converge. *(Right)* The optimal schedule is to keep η constant until t_{cross} and then setting $\beta = 1$. By doing so, we get the best of both worlds: the first phase minimises t_{cross} while the second allows to reach more informative solutions. *(Inset)* $m - m_{\text{gs}}$ shows that choosing higher β after t_{cross} allows to reach more informative minima. *Parameter:* $\beta = 0.8$, $\Delta_2 = 0.2$, $\Delta_p = 6$, $\eta_0 = 1$, $T = 1$, $dt = 10^{-2}$, $m_0 = 10^{-10}$.

In this section, we give additional results confirming the optimal decay of the learning rate in the spiked-matrix tensor model. We have seen in the main text, that there is a crossover time t_{cross} before which the learning rate should be kept fixed as the system is in the search phase. After t_{cross}, the dynamics enter a convex basin and one should decay the learning rate as $\eta(t) \sim t^{-\beta}$. To verify that $\beta = 1$ leads to the lowest loss, in the right panel of Fig. I.2, we keep a constant learning rate until t_{s} after which we vary the exponent with which the learning rate is decayed. We check that $\beta = -1$ allows to

reach the best solutions. However, the left panel shows that if the learning rate is decayed from start, the dynamics take much longer to converge towards the signal and remain stuck at high loss for very long.

I.4.4 Separation of time scales and matching solution

The long time dynamics, i.e. $t \to \infty$ of the p-spin model can be separated into two regimes:

- For all times $t, t \to \infty$ with $\frac{t-t'}{t} \to 0$ the system is stationary. Here, the dynamics are time-translation invariant (TTI) and the fluctuation-dissipation theorem (FDT) holds. The two time functions $C(t,t')$ and $R(t,t')$ are thus only a function of the time difference $\tau = t - t'$. In this regime, we define $C_{\mathrm{TTI}}(\tau) \equiv C(t-t',0)$ and $R_{\mathrm{TTI}}(\tau) \equiv R(t-t',0)$. The FDT gives $R_{\mathrm{TTI}}(\tau) = -\frac{1}{T}\frac{\mathrm{d}C_{\mathrm{TTI}}(\tau)}{\mathrm{d}\tau}$. As a consequence, the equations for R and C collapse into a single equation.

- For all times $t, t \to \infty$ with $\frac{t-t'}{t} = O(1)$ the system *ages* i.e. the dynamics remain trapped in metastable states and does not lose memory of its history. The relevant variable to consider in this regime is $\lambda = t'/t$. The correlation and response functions can be rescaled as $\mathcal{R}(\lambda) = tR(t,t')$ and $q\mathcal{C}(\lambda) = C(t,t')$ with $q = \lim_{\tau\to\infty} C_{\mathrm{TTI}}(\tau)$. In this aging regime, a generalised form of the FDT holds and $\mathcal{R}(\lambda) = \frac{x}{T} q \frac{\mathrm{d}\mathcal{C}(\lambda)}{\mathrm{d}\lambda}$. The *violation parameter* x is found by *matching* i.e. considering the equations for the response and the correlation separetly. q is found by imposing $q = lim_{\tau\to\infty} C_{\mathrm{TTI}}(\tau)$ in the equation of the TTI regime.

In order to derive analytical results, we use the hypothesis of these two times regimes to split the time integrals in Eqs. 9.12. For compactness we also define $Q(x) = x^p/2$. As noted in the main text, we can re-scale time according to $\mathrm{d}\tilde{t} = \eta(t)\,\mathrm{d}t$ and obtain a system at an effective temperature $\tilde{T} = (1-\beta)^{\frac{1}{1-\beta}} \frac{T}{t^{\beta/1-\beta}}$. We are ultimately interested in determining the threshold loss, a static quantity, and can hence perform its derivation using a constant learning rate. This analysis is a special case of the more general on performed in [307]. Here, we show it for the special case of the p-spin model with no signal. In particular, we skip all the computations and refer the reader to [307] (Appendix B) for additional details.

Lagrange multiplier in the long time-limit Let us start to illustrate how to proceed by computing the long time limit of the loss $z_\infty = \lim_{t\to\infty} z(t)$ using Eqs. 9.12:

$$\begin{aligned} z_\infty(T) - T &= p \underbrace{\int_0^t dt'' R(t'',t) Q'(C(t'',t))}_{\int_{\mathrm{TTI}} + \int_{\mathrm{aging}}} \\ &= -\int_0^\infty \frac{1}{T}\frac{d}{d\tilde{t}} Q\left(C_{\mathrm{TTI}}(\tilde{t})\right) d\tilde{t} + \int_0^1 \mathcal{R}(\lambda) Q'(q\mathcal{C}(\lambda)) d\lambda \\ \Leftrightarrow z_\infty &= \frac{1-q^p}{2T} + \int_0^1 \mathcal{R}(\lambda) Q'(q\mathcal{C}(\lambda)) d\lambda, \end{aligned} \tag{I.63}$$

where we used the fact that by definition $C_{\mathrm{TTI}}(\infty) = q$ and $C_{\mathrm{TTI}}(0) = 1$. Also note that we neglected all the finite time contribution to the integrals. We are going to determine z_∞ using this equation.

Stationary regime In order to find the dynamical equations in the stationary regime, we proceed as before and separate the contributions of the TTI regime from those of the aging regime in the integrals. Since both equations for the response and the correlation collapse into a single equation, we consider only the evolution of the correlation C_{TTI}. Using Eqs. 9.12 we have:

$$(z_\infty + \partial_\tau) C_{\mathrm{TTI}}(\tau) = \int_0^{t_1} dt'' R\left(t_1, t''\right) Q''\left(C\left(t_1, t''\right)\right) C\left(t'', t_2\right) + \int_0^{t_2} dt'' R(t'', t_2) Q'\left(C(t'', t_1)\right) \tag{I.64}$$

Using Eqs. 62 of [307], we have:

$$\partial_\tau C_{\mathrm{TTI}}(\tau) + \left(\frac{1}{T} Q'(1) - \mu_\infty\right)\left[1 - C_{\mathrm{TTI}}(\tau)\right] + T = -\frac{1}{T}\int_0^\tau Q'\left(C_{\mathrm{TTI}}\left(\tau - \tau''\right)\right) \frac{d}{d\tau''} C_{\mathrm{TTI}}\left(\tau''\right) d\tau'' \tag{I.65}$$

When $\tau \to \infty$, the time variations of $C_{\mathrm{TTI}}(\tau)$ are negligible. Taking this limit in the above equation gives:

$$z_\infty = \sqrt{Q''(q)} + \frac{Q'(1) - Q'(q)}{T} \tag{I.66}$$

This equation allows to determine the threshold loss, i.e. the loss at the plateau reached by the system before the recovery of the signal. We notice that the equality above holds for all Δ_2, Δ_p and hence also if one of the two is sent to infinity. Therefore, we have:

$$\ell_{th} = \ell_p + \ell_2, \tag{I.67}$$

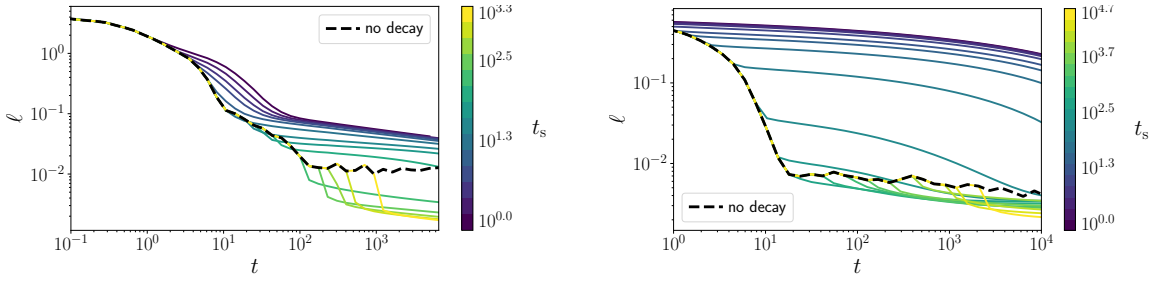
with $\ell_2 = \frac{1}{2}(\eta(t) - z_{\infty;\Delta_p \to \infty})$ and similarly for ℓ_p. Thus, by defining $Q_k(x) = {}^{x^k}/_{k\Delta_k}$, we obtain:

$$\ell_k = \frac{1}{k}\left(\eta(t) - \sqrt{Q_k''(q)} - \frac{Q_k'(1) - Q_k'(q)}{T}\right) \tag{I.68}$$

Using this equation, and performing an expansion around 0 for T and 1 for q, we can determine that at low temperatures, the threshold energy scales linearly with T.

I.5 Additional results for the Teacher-Student Regression Task

In this section we give additional results on the teacher-student regression task discussed in Sec. 9.5. The setting is the same as in the main text: a K hidden nodes 2 layer neural network student is trained to reproduce the output of her 2 layer neural network teacher of M nodes on gaussian inputs. We train the model with on a finite dataset of P examples using a mini-batch size $B = 1$. Fig. I.3 verifies that the conclusions drawn in the main text hold for different values of K and M. The optimal schedule is to keep the learning rate constant until t_{cross} and to then decay it as $1/t$. If the learning rate is decayed too soon, i.e. at $t_s < t_{\text{cross}}$, learning remains stuck at high loss values. Decaying after t_{cross} instead allows to reduce the noise in optimisation and reach lower loss solutions. We verify that in both these cases, t_{cross} matches the end of the "specialisation" transition, where the loss achieved student trained at constant learning rate plateaus.



(a) $K = M = 5$ (b) $K = M = 20$

Figure I.3: **The crossover time is also reflected in a regression task with SGD.** A student with K hidden nodes is trained to reproduce the output of her M hidden nodes. *(Left)* $K = M = 5$. *(Right)* $K = M = 20$. As in the main text.before, we find that decaying the learning rate before the loss plateaus performance, but decaying as $\eta(t) \sim t^{-1}$ once the plateau is reached allows to reach zero loss. *Parameters:* $N = 500$, $P = 10^4$, $\eta_0 = 10^{-1}$, $\beta = 0.8$.

Appendix J

Deep Symbolic Regression for Recurrent Sequences

J.1 Robustness to noise

One particularity of our model is that it is entirely trained and evaluated on synthetic data which is completely noise-free. Can our model also predict recurrence relations when the inputs are corrupted ? In this section, we show that the answer is yes, provided the model is trained with noisy inputs. For simplicity, we restrict ourselves here to the setup of float sequences, but the setup of integer sequences can be dealt with in a similar manner, trading continuous random variables for discrete ones.

Setup Considering the wide range of values that are observed in recurrent sequences, corruption via additive noise with constant variance, i.e. $u_n = f(n, \{u_i\}_{i<n}) + \xi_n, \xi_n \sim \mathcal{N}(0, \sigma)$ is a poor model of stochasticity. Indeed, the noise will become totally negligible when $u_n \gg 1$, and conversely, totally dominate when $u_n \ll 1$. To circumvent this, we scale the variance of the noise with the magnitude of the sequence, i.e. $\xi_n \sim \mathcal{N}(0, \sigma u_n)$, allowing to define a signal-to-noise ratio $\mathrm{SNR} = 1/\sigma$. This can also be viewed as a multiplicative noise $u_n = f(n, \{u_i\}_{i<n})\xi, \xi \sim \mathcal{N}(1, \sigma)$.

Results To make our models robust to corruption in the sequences, we use stochastic training. This involves picking a maximal noise level σ_{train}, then for each input sequence encountered during training, sample $\sigma \sim \mathcal{U}(0, \sigma_{train})$, and corrupt the terms with a multiplicative noise of variance σ. At test time, we corrupt the input sequences with a noise of fixed variance σ_{test}, but remove the stochasticity for next term prediction, to check whether our model correctly inferred the deterministic part of the formula. Results are presented in Table J.1. We see that without the stochastic training, the accuracy of our model plummets from 43% to 1% as soon as noise is injected at test time. However, with stochastic training, we are able to keep decent performance even at very strong noise levels: at $\sigma_{test} = 0.5$, we are able to achieve an accuracy of 17%, which is remarkable given that the signal-to-noise ratio is only of two. However, this robustness comes at a cost: performance on the clean dataset is degraded, falling down to 30%.

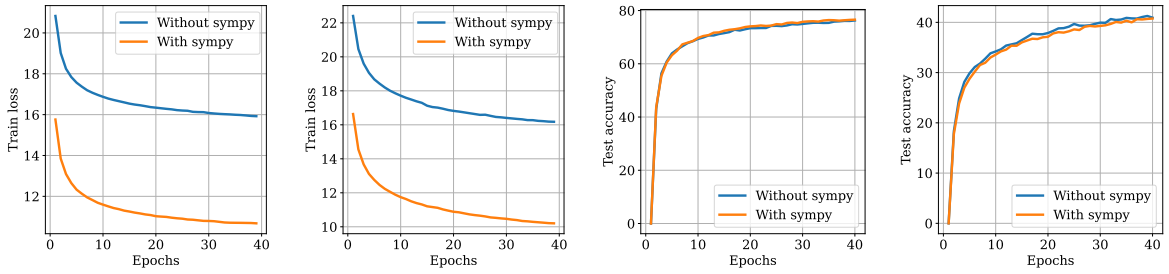
$\sigma_{train/test}$	$\sigma_{test} = 0$	$\sigma_{test} = 0.1$	$\sigma_{test} = 0.5$
$\sigma_{train} = 0$	**43.3**	0.9	0.0
$\sigma_{train} = 0.1$	38.4	**31.9**	0.2
$\sigma_{train} = 0.5$	35.6	31.8	**11.1**

Table J.1: **Our symbolic model can be made robust to noise in the inputs, with a moderate drop in performance on clean inputs.** We report the accuracy on expressions with up to 10 operators, for $n_{pred} = 10$, $\tau = 10^{-10}$, varying the noise level during training σ_{train} and evaluation σ_{test}.

J.2 The effect of expression simplification

One issue with symbolic regression is the fact that a mathematical expression such as `mul, 2, cos, n` can be written in many different ways. Hence, cross-entropy supervision to the tokens of the expression can potentially penalize the model for generating the same formula written in a different way (e.g. `mul, cos, n, 2` or `mul, cos, n, add, 1, 1`). To circumvent this issue, [403] first predict the formula, then evaluate it and supervise the evaluations to those of the target function. Yet, since the

evaluation step is non-differentiable, they are forced to use a Reinforcement Learning loop to provide reward signals. In our framework, we noticed that such an approach is actually unnecessary.
Instead, one could simply preprocess the mathematical formula to simplify it with SymPy [465] before feeding it to the model. This not only simplifies redundant parts such as `add, 1, 1`$\rightarrow$2, but also gets rid of the permutation invariance `mul, x, 2` $=$ `mul, 2, x` by following deterministic rules for the order of the expressions. However, and rather surprisingly, we noticed that this simplification does not bring any benefit to the predictive power of our model: although it lowers the training loss (by getting rid of permutation invariance, it lowers cross-entropy), it does not improve the test accuracy, as shown in Fig. J.1. This suggests that expression syntax is not an issue for our model: the hard part of the problem indeed lies in the mathematics.
Aside from predictive power, SymPy comes with several advantages and drawbacks. On the plus side, it enables the generated expressions to be written in a cleaner way, and improves the diversity of the beam. On the negative side, it slows down training, both because it is slow to parse complex expressions, and because it often lengthens expression since it does not handle division (SymPy rewrites `div,a,b` as `mul,a,pow,b,-1`). Additionally, simplification actually turns out to be detrimental to the out-of-domain generalization of the float model. Indeed, to generate approximations of out-of-vocabulary prefactors, the latter benefits from non-simplified numerical expressions. Hence, we chose to not use SymPy in our experiments.



(a) Training loss, integer (b) Training loss, float (c) Test accuracy, integer (d) Test accuracy, float

Figure J.1: **Simplification reduces the training loss, but does not bring any improvement in test accuracy**. We displayed the first 40 epochs of training of our symbolic models.

J.3 Does memorization occur?

It is natural to ask the following question: due to the large amount of data seen during training, is our model simply memorizing the training set ? Answering this question involves computing the number of possible inputs sequences N_{seq} which can be generated. To estimate this number, calculating the number of possible mathematical expressions N_{expr} is insufficient, since a given expression can give very different sequences depending on the random sampling of the initial terms. Hence, one can expect that N_{expr} is only a very loose lower bound for N_{seq}.
Nonetheless, we provide the lower bound N_{expr} as a function of the number of nodes in Fig. J.2, using the equations provided in [390]. For small expressions (up to four operators), the number of possible expressions is lower or similar to than the number of expressions encountered during training, hence one cannot exclude the possibility that some expressions were seen several times during training, but

with different realizations due to the initial conditions. However, for larger expressions, the number of possibilities is much larger, and one can safely assume that the expressions encountered at test time have not been seen during training.

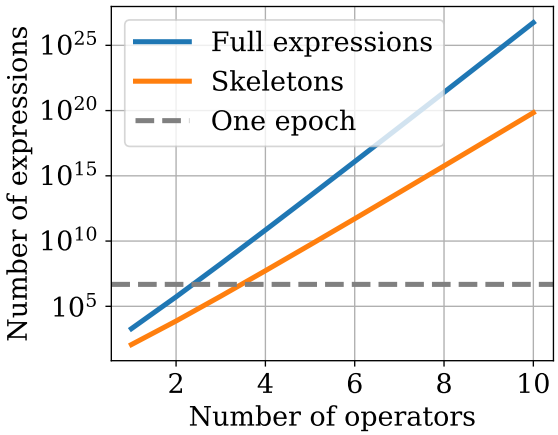


Figure J.2: **Our models only see a small fraction of the possible expressions during training.** We report the number of possible expressions for each number of operators (skeleton refers to an expression with the choice of leaves factored out). Even after a hundred epochs, our models have only seen a fraction of the possible expressions with more than 4 operators.

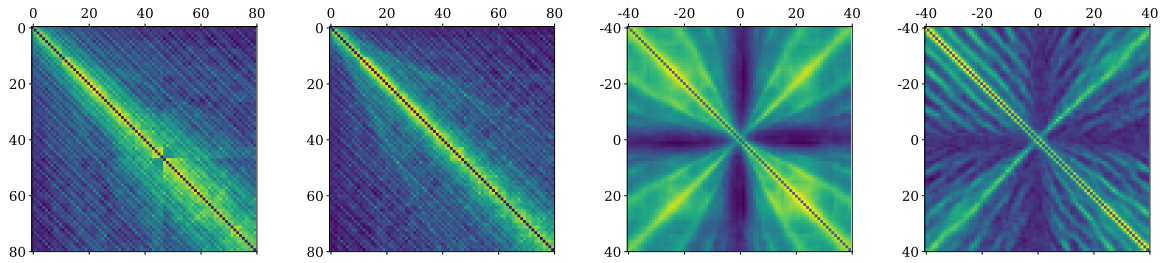
J.4 The issue of finite precision

As explained in the main text, our model tends to ignore subdominant terms in expressions with terms of vastly different magnitudes, partly due to finite precision. For example, when using a float precision of $p = 4$ digits, we obtain a discretization error of 10^5 for numbers of magnitude 10^9. However, there exists two methods to circumvent this issue.

Increasing the precision Naively increase the precision p would cause the vocabulary size of the encoder to rapidly explode, as it scales as 10^p. However, one can instead encode the mantissa on multiple tokens, as performed for integers. For example, using two tokens instead of one, e.g. encoding π as `+ 3141, 5926, E-11`, doubles the precision while increasing the sequence length only by 33%.
This approach works well for recurrence prediction, but slightly hampers the ability of the model to approximate prefactors as shown in Tab. 11.2, hence we did not use it in the runs presented in this chapter.

Iterative refinement Another method to improve the precision of the model is to use an iterative refinement of the predicted expression, akin to perturbation theory in physics.
Consider, for example, the polynomial $f(x) = \sum_{k=0}^{d} a_k x^k$, for which our model generally predicts $\hat{f}(x) = \sum_{k=0}^{d} \hat{a}_k x^k$, with the first coefficient correct ($\hat{a}_d = a_d$) but potentially the next coefficients incorrect ($\hat{a}_k \neq a_k$ for $k < d$). One can correct these subdominant coefficients iteratively, order by order. To obtain the term a_{d-1}, fit the values of $g(x) = f(x) - \hat{f}(x) = \sum_{k=1}^{d-1}(a_k - \hat{a}_k)$. Then fit the values of

$h(x) = g(x) - \hat{g}(x)$, etc. By iterating this procedure k times, one can obtain the k highest coefficients a_k.
We checked that this method allows us to approximate any polynomial function. One could in fact use iterative refinement to predict the Taylor approximation of any function, or use a similar approach to catch multiplicative corrections, by fitting $g(x) = f(x)/\hat{f}(x)$; we leave these investigations for future work.



(a) Integer symbolic (b) Integer numeric (c) Float symbolic (d) Float numeric

Figure J.3: **The similarity matrices reveal more details on the structure of the embeddings**. The element (i, j) is the cosine similarity between embeddings i and j.

J.5 Structure of the embeddings

To gain better understanding on the number embeddings of our models, we depict similarity matrices whose element (i, j) is the cosine similarity between embeddings i and j in Fig. J.3.

Integer sequences The numeric and symbolic similarity matrices look rather similar, with a bright region appearing around the line $y = x$, reflecting the sequential nature of the embeddings. In both cases, we see diagonal lines appear: these correspond to lines of common divisors between integers. Strikingly, these lines appear most clearly along multiples of 6 and 12, especially in the symbolic model, suggesting that 6 is a natural base for reasoning. These results are reminiscent of the much earlier explorations of [466].

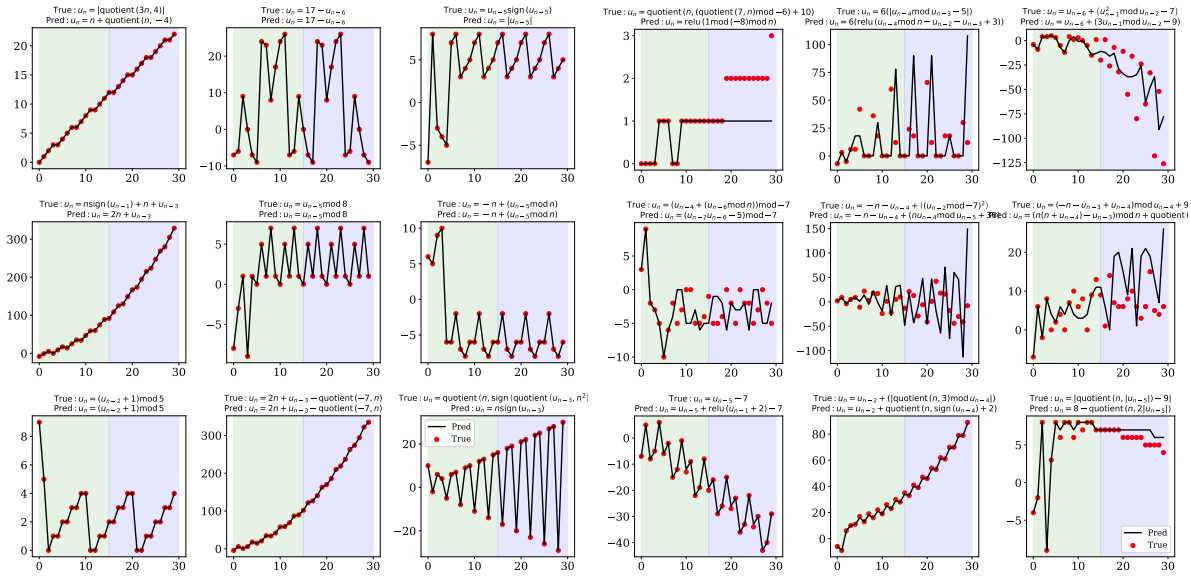
Float sequences Both for the numeric and symbolic setups, the brightest regions appear along the diagonal lines $y = x$ and $y = -x$, reflecting respectively the sequential nature of the embeddings and their symmetry around 0. The darkest regions appears around the vertical line $x = 0$ and the horizontal line $y = 0$, corresponding to exponents close to zero: these exponents strongly overlap with each other, but weakly overlap with the rest of the exponents. Interestingly, dimmer lines appear in both setups, but follow a very different structure. In the symbolic setup, the lines appear along lines $y = \pm x/k$, reminiscent of the effect of polynomials of degree k. In the numeric setup, the lines are more numerous, all diagonal but offset vertically.

J.6 Visualizations

Success and failure modes In Fig. J.4, we show a few examples of success and failure modes of our symbolic models. The failure modes are particularly interesting, as they reflect the strong behavioral difference between our symbolic models and models usually used for regression tasks.
The latter generally try to interpolate the values of the function they are given, whereas our symbolic model tries to predict the expression of the function. Hence, our model cannot simply "overfit" the inputs. A striking consequence of this is that in case of failure, the predicted expression is wrong both on the input points (green area) and the extrapolation points (blue area).
In some cases, the incorrectly predicted formula provides a decent approximation of the true function (e.g. when the model gets a prefactor wrong). In others, the predicted formula is catastrophically wrong (e.g. when the model makes a mistake on an operator or a leaf).

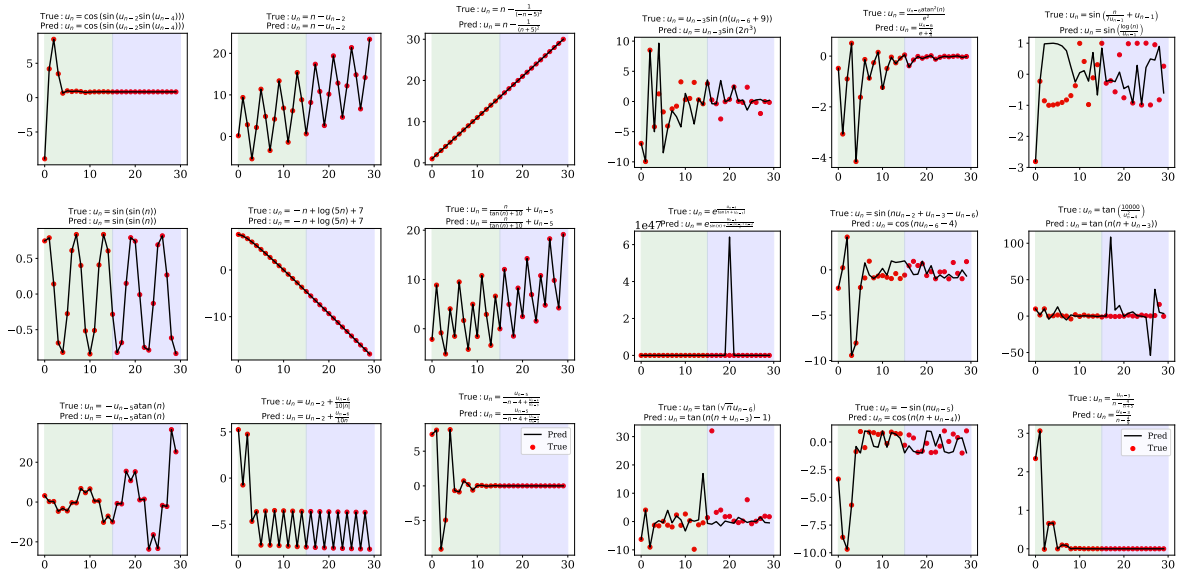
Training curves In Fig. J.5, we show the training curves of our models, presenting an ablation over the tolerance τ, the number of predictions n_{pred}, the number of operators o, the recurrence degree d and the number of input points l, as explained in the main text.

Attention maps In Fig. J.6, we provide attention maps for the 8 attention heads and 4 layers of our Transformer encoders. Clearly, different heads play very different roles, some focusing on local interactions and others on long-range interactions. However, the role of different layers is hard to interpret.



(a) Integer, success

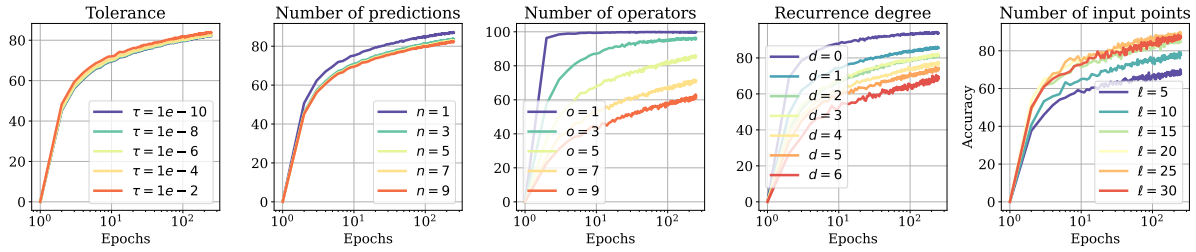
(b) Integer, failure



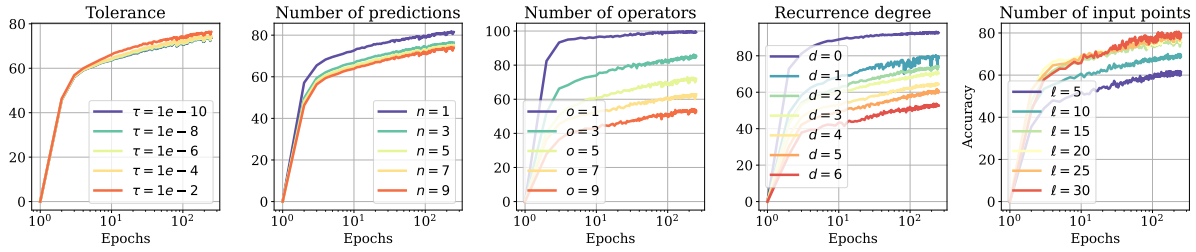
(c) Float, success

(d) Float, failure

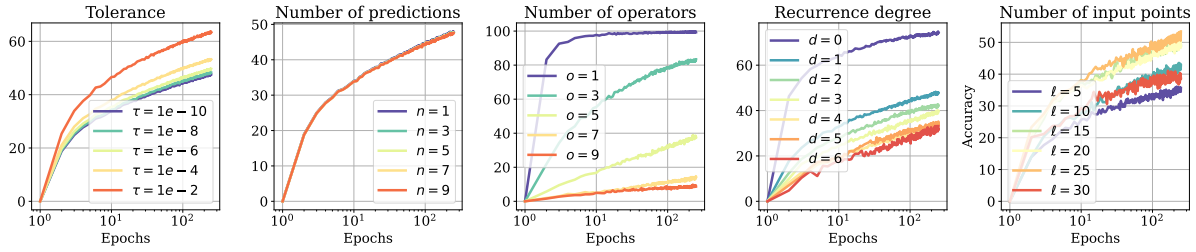
Figure J.4: **Success and failure modes of our models.** The models are fed the first 15 terms of the sequence (green area) and predict the next 15 terms (blue area). We randomly selected expressions with 4 operators from our generator, and picked the first successes and failures.



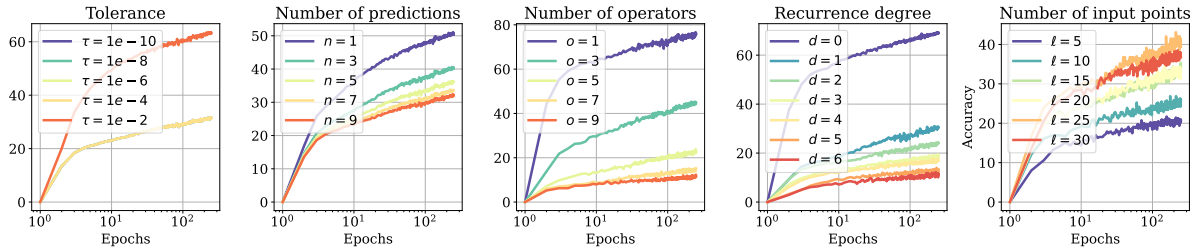
(a) Integer symbolic



(b) Integer numeric

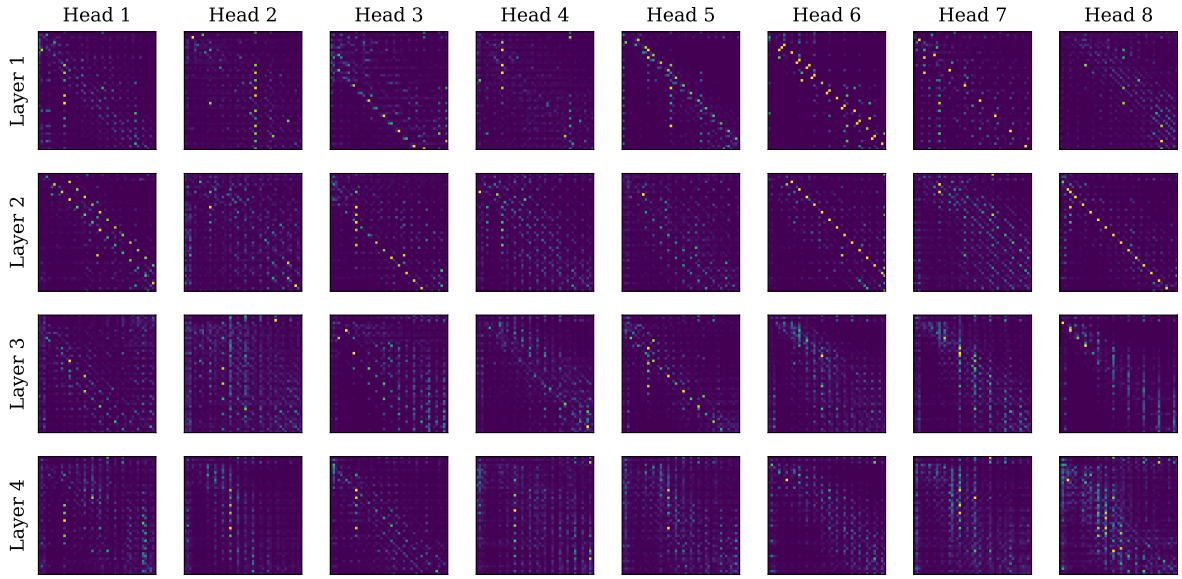


(c) Float symbolic

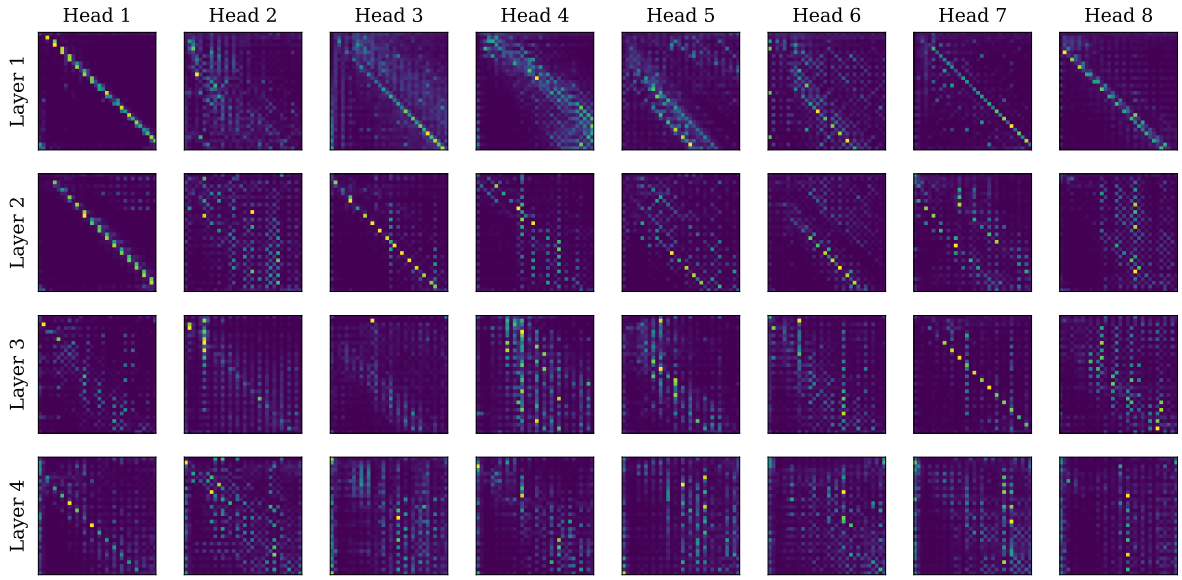


(d) Float numeric

Figure J.5: **Training curves of our models.** We plot the accuracy of our models at every epoch, evaluated of 10,000 sequences generated from the same distribution as during trianing. From left to right, we vary the tolerance τ, the number of predictions n_{pred}, the number of operators o, the recurrence degree d and the number of input terms l. In each plot, we use the following defaults for quantities which are not varied: $\tau = 10^{-10}$, $n_{pred} = 10$, $o \in [\![1, 10]\!]$, $d \in [\![1, 6]\!]$, $l \in [\![5, 30]\!]$.



(a) Integer



(b) Float

Figure J.6: **Attention maps of our integer model and float models**. We evaluated the integer model on the first 25 terms of the sequence $u_n = -(6 + u_{n-2}) \bmod n$ and the float model on the first 25 terms of the sequence $u_n = \exp(\cos(u_{n-2}))$.

Appendix K

End-to-end Symbolic Regression with Transformers

K.1 Details on the training data

In Tab. K.1 we provide the detailed set of parameters used in our data generator. The probabilities of the unary operators were selected to match the natural frequencies appearing in the Feynman dataset.
In Fig. K.1, we show the statistics of the data generation.The number of expressions diminishes with the input dimension and number of unary operators because of the higher likelihood of generating out-of-domain inputs. One could easily make the distribution uniform by enforcing to retry as long as a valid example is not found, however we find empirically that having more easy examples than hard ones eases learning and provides better out-of-domain generalization, which is our ultimate goal.
In Fig. K.2, we show some examples of the input distributions generated by our multimodal approach. Notice the diversity of shapes obtained by this procedure.

Parameter	Description	Value
D_{max}	Max input dim	10
$\mathcal{D}_{\text{aff}}$	Distrib of (a,b)	sign $\sim \mathcal{U}\{-1, 1\}$, mantissa $\sim \mathcal{U}(0, 1)$, exponent $\sim \mathcal{U}(-2, 2)$
b_{max}	Max binary ops	$5 + D$
O_b	Binary operators	add:1, sub:1, mul:1
u_{max}	Max unary ops	5
O_u	Unary operators	inv:5, abs:1, sqr:3, sqrt:3, sin:1, cos:1, tan:0.2, atan:0.2, log:0.2, exp:1
N_{min}	Min number of points	$10D$
N_{max}	Max number of points	200
k_{max}	Max num clusters	10

Table K.1: **Parameters of our generator.**

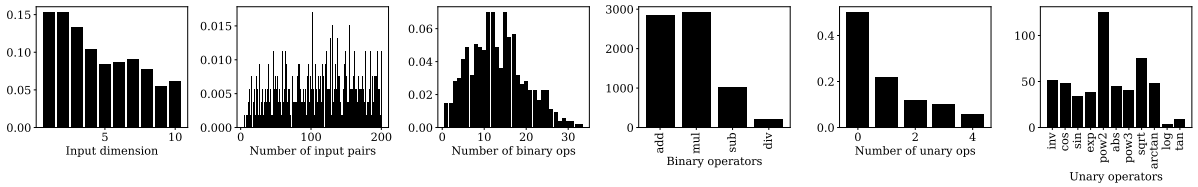


Figure K.1: **Statistics of the synthetic data.** We calculated the latter on $10,000$ generated examples.

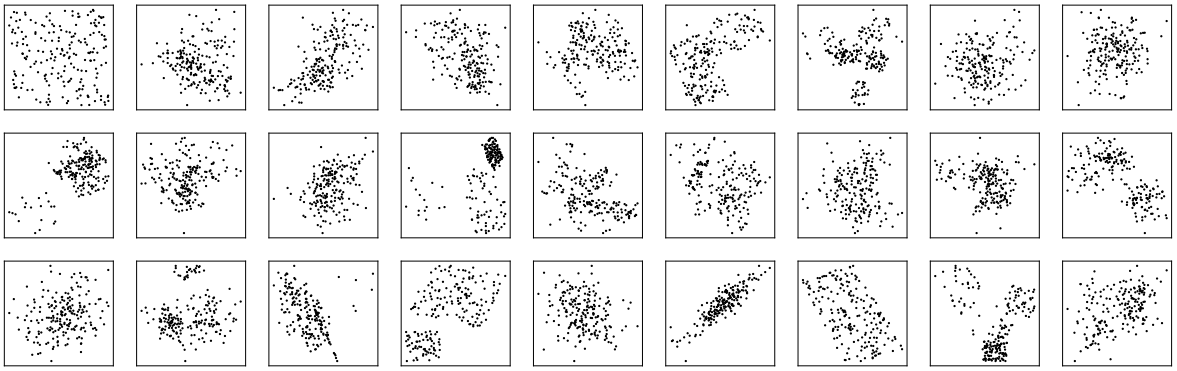


Figure K.2: **Diversity of the input distributions generated by the multimodal approach.** Here we show distributions obtained for $D = 2$.

K.2 Attention maps

A natural question is whether self-attention based architectures are optimally suited for symbolic regression tasks. In Fig. K.3, we show the attention maps produced by the encoder of our Transformer model, which contains 4 layers avec 16 attention heads (we only keep the first 8 for the sake of space). In order to make the maps readable, we consider one-dimensional inputs and sort them in ascending order. The attention plots demonstrate the complementarity of the attention heads. Some focus on specific regions of the input, whereas others are more spread out. Some are concentrated along the diagonal (focusing on neighboring points), whereas others are concentrated along the anti-diagonal (focusing on far-away points.
Most strikingly, the particular features of the functions studied clearly stand out in the attention plots. Focus, for example, on the 7th head of layer 2. For the exponential function, it focuses on the extreme points (near -1 and 1); for the inverse function, it focuses on the singularity around the origin; for the sine function, it reflects the periodicity, with evenly spaces vertical lines. The same phenomenology can be acrossed is several other heads.

K.3 Does memorization occur?

It is natural to ask the following question: due to the large amount of data seen during training, is our model simply memorizing the training set ? Answering this question involves computing the number of possible functions which can be generated. To estimate this number, calculating the number of possible

skeleton N_s is insufficient, since a given skeleton can give rise to very different functions according to the sampling of the constants, and even for a given choice of the constants, the input points $\{x\}$ can be sampled in many different ways.
Nonetheless, we provide the lower bound N_s as a function of the number of nodes in Fig. K.4, using the equations provided in [390]. For small expressions (up to four operators), the number of possible expressions is lower or similar to than the number of expressions encountered during training, hence one cannot exclude the possibility that some expressions were seen several times during training, but with different realizations due to the initial conditions. However, for larger expressions, the number of possibilities is much larger, and one can safely assume that the expressions encountered at test time have not been seen during training.

K.4 Additional in-domain results

Fig. K.5, we present a similar ablation as Fig. 12.4 of the main text but using the R^2 score as metric rather than accuracy.

K.5 Additional out-of-domain results

Complexity-accuracy In Fig. K.6, we display a Pareto plot comparing accuracy and formula complexity on SRBench datasets.

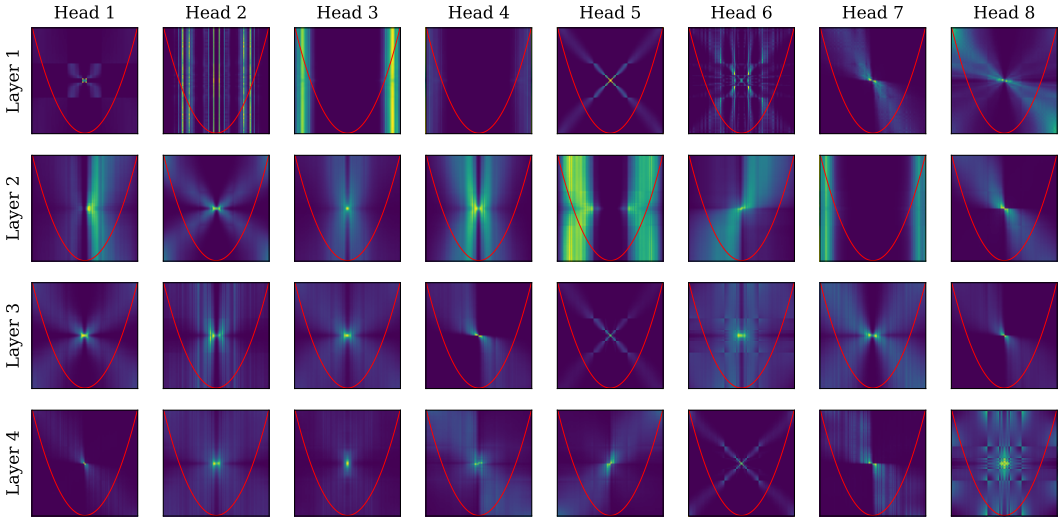
Jin benchmark In Fig. K.7, we show the predictions of our model on the functions provided in [467]. Our model gets all of them correct except for one.

Black-box datasets In Fig. K.8, we display the results of our model on the black-box problems of SRBench.

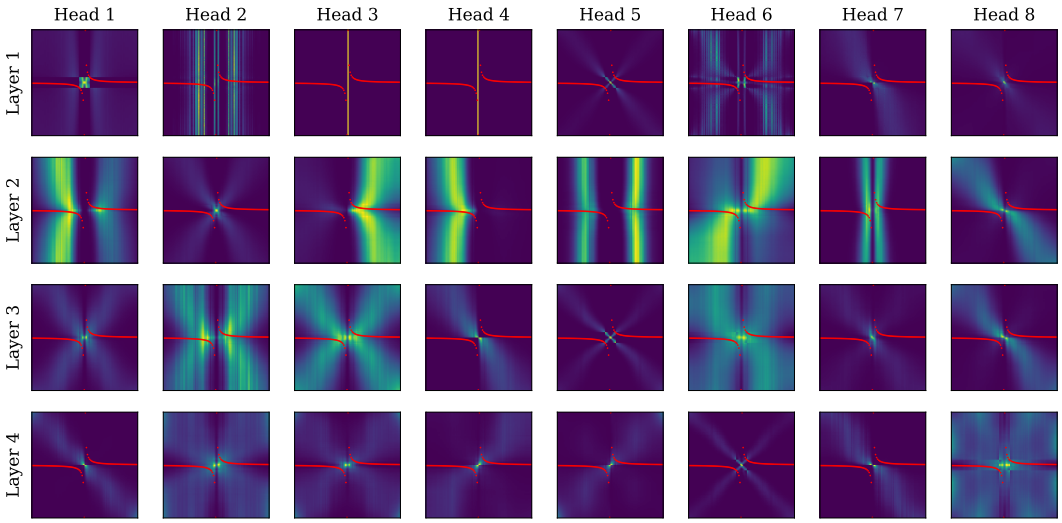
Strogatz datasets Each of the 14 datasets from the ODE-Strogatz benchmark is the trajectory of a 2-state system following a first-order ordinary differential equation (ODE). Therefore, the input data has a very particular, time-ordered distribution, which differs significantly from that seen at train time. Unsurprisingly, Fig. K.9 shows that our model performs somewhat less well to this kind of data in comparison with GP-based methods.

Ablation on input dimension In Fig. K.10, we show how the performance of our model depends on the dimensionality of the inputs on Feynamn and black-box datasets.

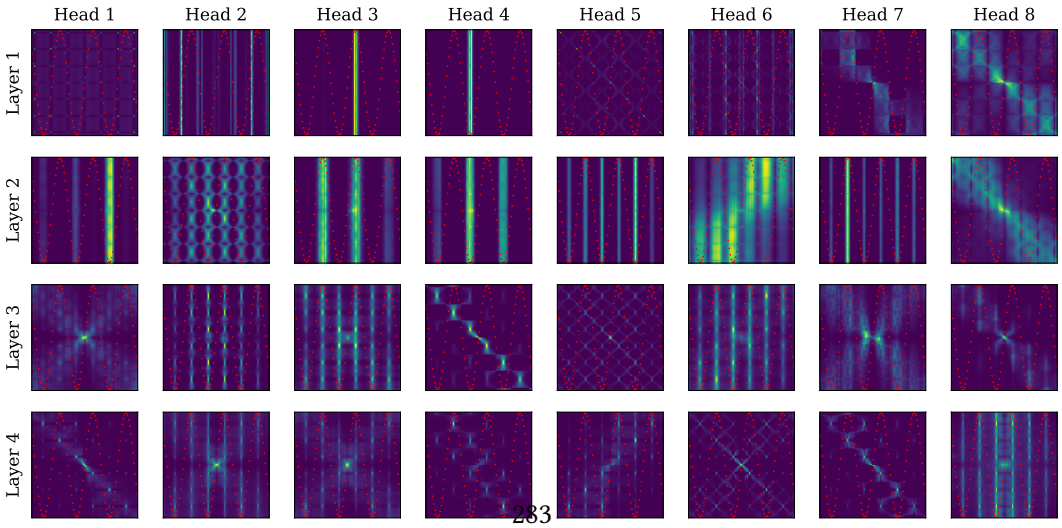
Ablation on decoding strategy In Fig. K.11, we display the difference in performance using two decoding strategies.



(a) $f(x) = x^2$



(b) $f(x) = 1/x$



(c) $f(x) = \sin(10x)$

Figure K.3: **Attention maps reveal distinctive features of the functions considered.** We presented the model 1-dimensional functions with 100 input points sorted in ascending order, in order to better visualize the attention. We plotted the self-attention maps of the first 8 (out of 16) heads of the Transformer encoder, across all four layers. We see very distinctive patterns appears: exploding areas for

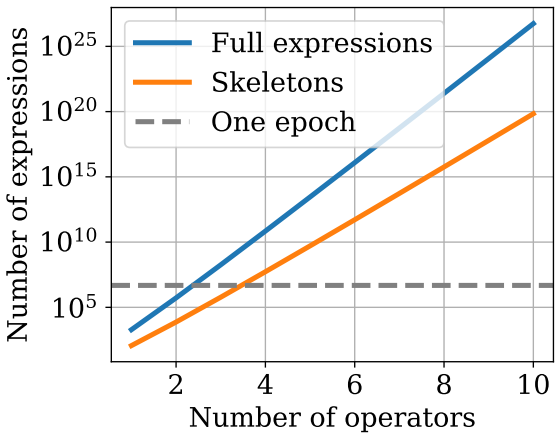


Figure K.4: **Our models only see a small fraction of the possible expressions during training.** We report the number of possible skeletons for each number of operators. Even after a hundred epochs, our models have only seen a fraction of the possible expressions with more than 4 operators.

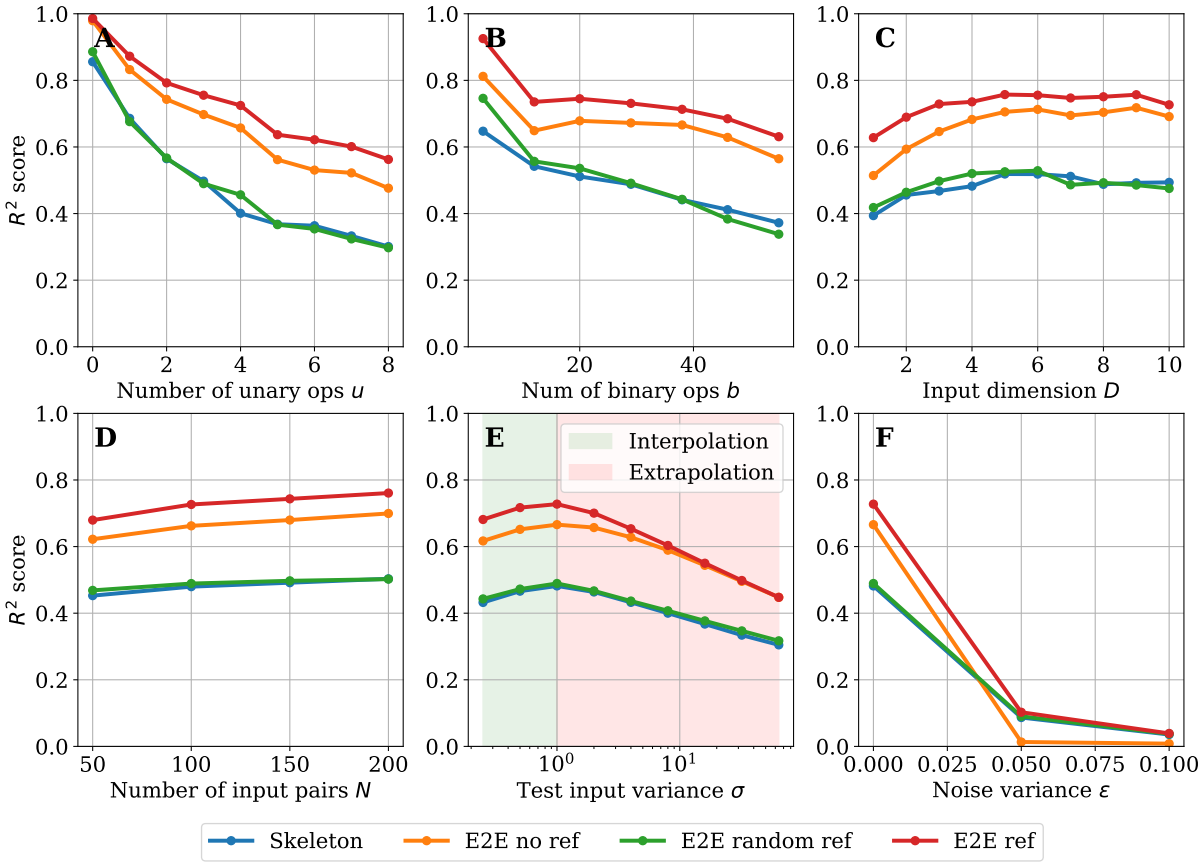


Figure K.5: **Ablation over the function difficulty (top row) and input difficulty (bottom row).** We plot the R^2 score (Eq. 12.1). **A:** number of unary operators. **B:** number of binary operators. **C:** input dimension. **D:** Low-resource performance, evaluated by varying the number of input points. **E:** Extrapolation performance, evaluated by varying the variance of the inputs. **F:** Robustness to noise, evaluated by varying the multiplicative noise added to the labels.

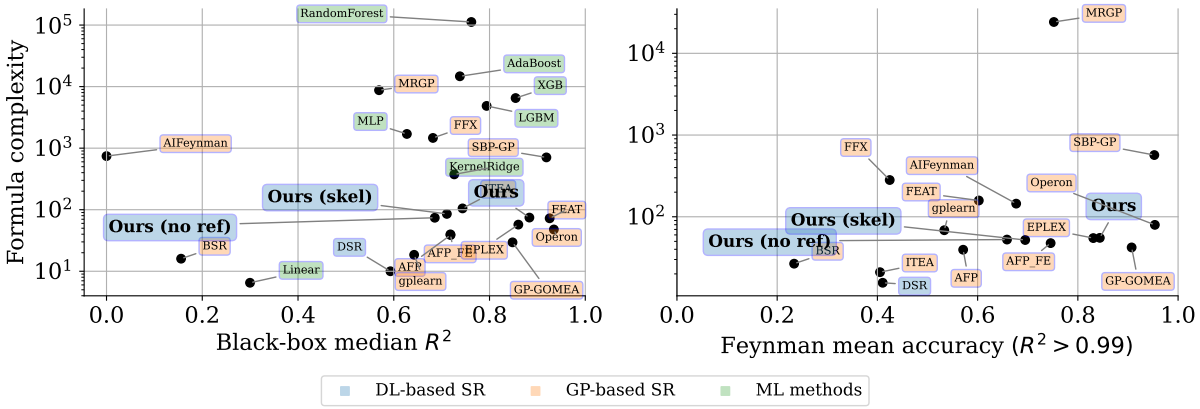
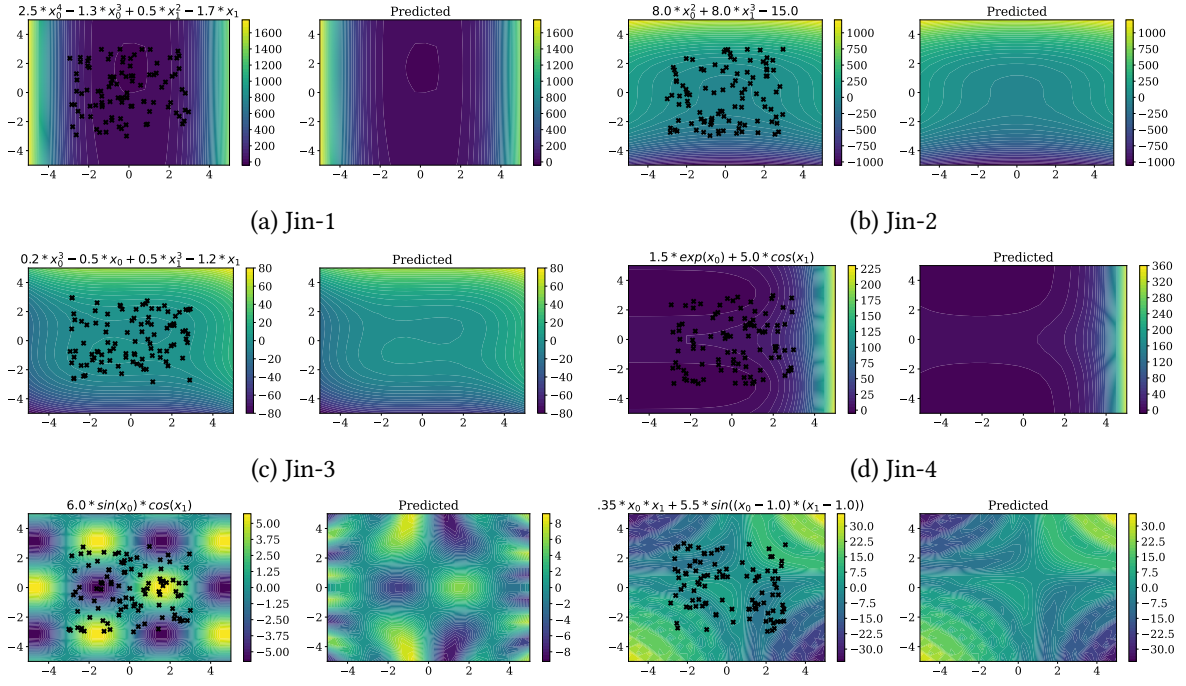


Figure K.6: **Complexity-accuracy pareto plot.** Pareto plot comparing the average test performance and formula complexity of our models with baselines provided by the SRbench benchmark [417], both on Feynman SR problems [413] and black-box regression problems. We use colors to distinguish three families of models: deep-learning based SR, genetic programming-based SR and classic machine learning methods (which do not provide an interpretable solution).



(a) Jin-1 (b) Jin-2

(c) Jin-3 (d) Jin-4

(e) Jin-5 (f) Jin-6

Figure K.7: **Illustration of our model on a few benchmark datasets from the litterature.** We show the prediction of our model on six 2-dimensional datasets presented in [467] and used as a comparison point in a few recent works [468]. The input points are marked as black crosses. Our model retrieves the correct expression in all but one of the cases: in Jin5, the prediction matches the input points correctly, but extrapolates badly.

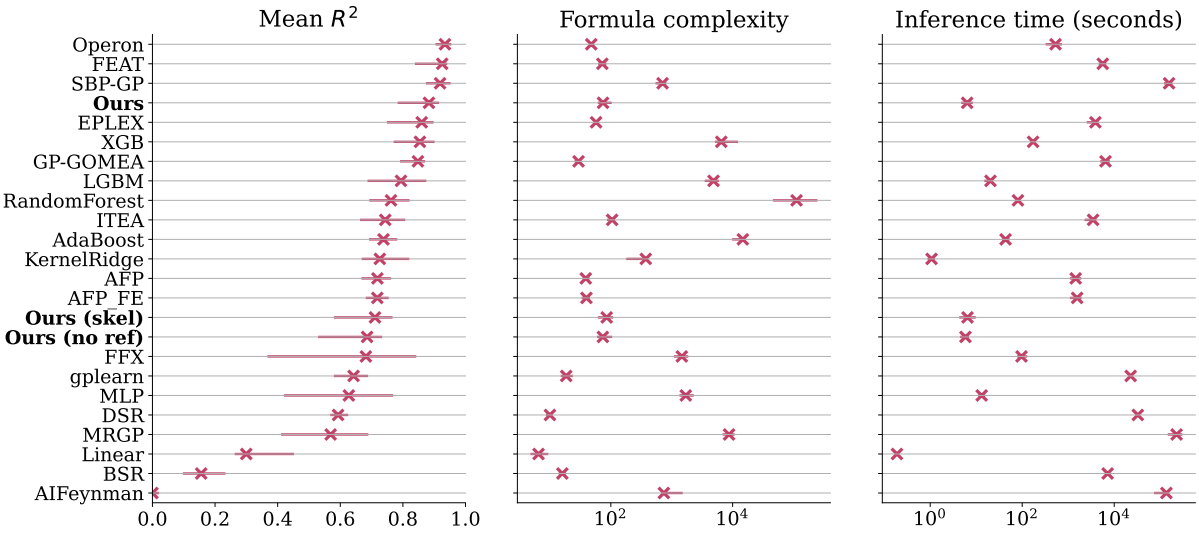


Figure K.8: **Performance metrics on black-box datasets.**

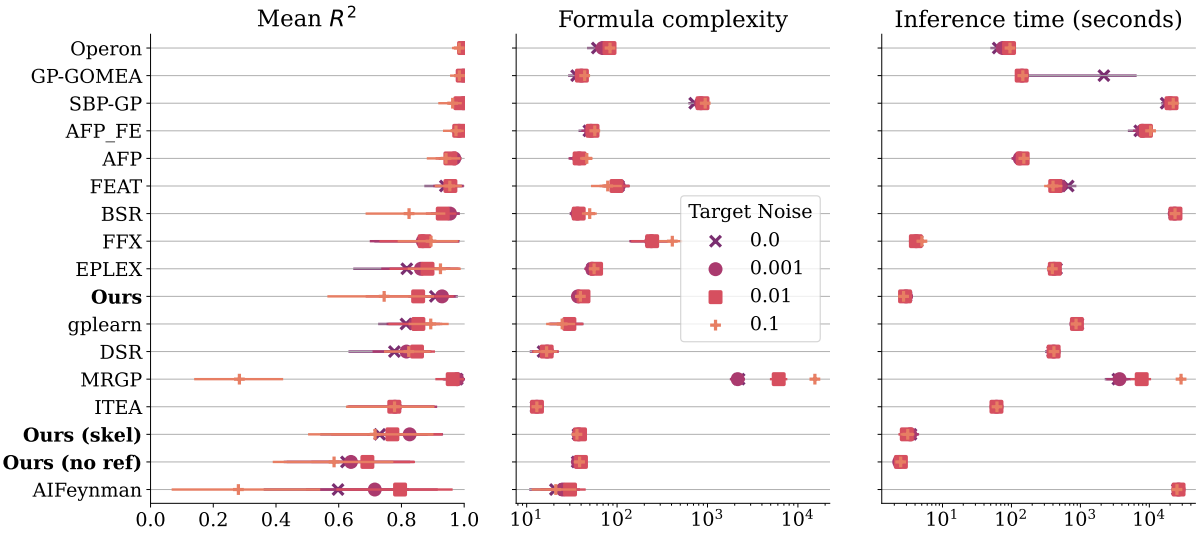
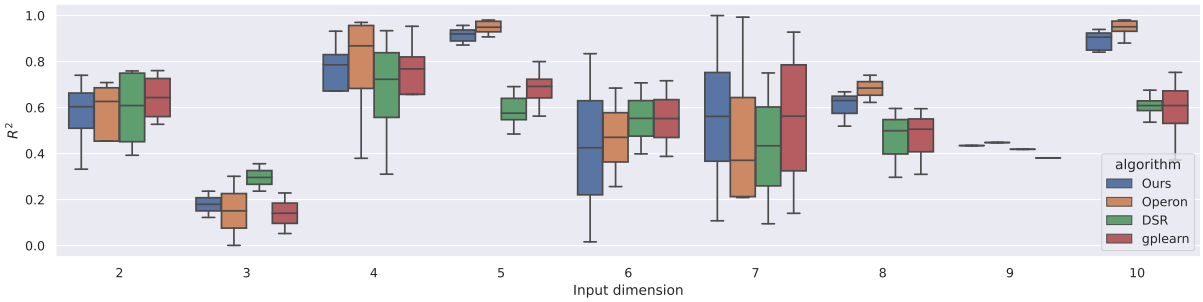
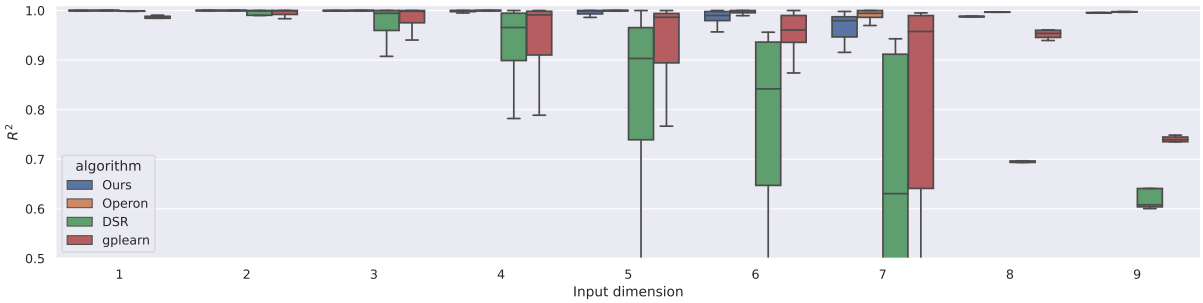


Figure K.9: **Performance metrics on Strogatz datasets.**



(a) Black-box



(b) Feynman

Figure K.10: **Performance metrics on SRBench, separated by input dimension.**

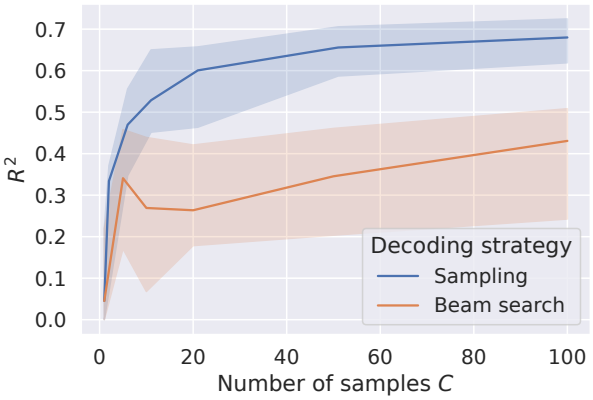


Figure K.11: **Median R^2 of our method without refinement on black-box datasets when $B = 1$, varying the number of decoded function samples.** The beam search [435] used in [404] leads to low-diversity candidates in our setup due to expressions differing only by small modifications of the coefficients.

Bibliography

1. Krizhevsky, A., Sutskever, I. & Hinton, G. *Imagenet classification with deep convolutional neural networks* in *Advances in neural information processing systems* (2012), 1097–1105.
2. Hinton, G. *et al.* Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal processing magazine* **29,** 82–97 (2012).
3. Sutskever, I., Vinyals, O. & Le, Q. V. *Sequence to sequence learning with neural networks* in *Advances in neural information processing systems* (2014), 3104–3112.
4. Silver, D. *et al.* Mastering the game of go without human knowledge. *nature* **550,** 354–359 (2017).
5. Zhang, C., Bengio, S., Hardt, M., Recht, B. & Vinyals, O. Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530* (2016).
6. Geiger, M. *et al.* Jamming transition as a paradigm to understand the loss landscape of deep neural networks. ***Physical Review E*** (2019).
7. Spigler, S. *et al.* A jamming transition from under-to over-parametrization affects generalization in deep learning. ***Journal of Physics A*** (2019).
8. Geiger, M. *et al.* Scaling description of generalization with number of parameters in deep learning. ***Journal of Statistical Mechanics*** (2020).
9. d'Ascoli, S., Refinetti, M., Biroli, G. & Krzakala, F. Double Trouble in Double Descent: Bias and Variance (s) in the Lazy Regime. ***International Conference on Machine Learning*** (2020).
10. d'Ascoli, S., Sagun, L. & Biroli, G. Triple descent and the two kinds of overfitting: where and why do they appear? ***Advances in Neural Information Processing Systems*** (2020).
11. d'Ascoli, S., Sagun, L. & Biroli, G. Triple descent and the two kinds of overfitting: where and why do they appear? ***Journal of Statistical Mechanics (special issue)*** (2022).
12. d'Ascoli, S., Gabrié, M., Sagun, L. & Biroli, G. On the interplay between loss function and data structure in classification problems. ***Advances in Neural Information Processing Systems*** (2021).
13. Krizhevsky, A., Sutskever, I. & Hinton, G. E. *Imagenet classification with deep convolutional neural networks* in *Advances in neural information processing systems* (2012), 1097–1105.
14. Dosovitskiy, A. *et al.* An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. *arXiv preprint arXiv:2010.11929* (2020).
15. Tolstikhin, I. O. *et al.* Mlp-mixer: An all-mlp architecture for vision. *Advances in Neural Information Processing Systems* **34** (2021).

16. d'Ascoli, S., Sagun, L., Biroli, G. & Bruna, J. Finding the Needle in the Haystack with Convolutions: on the benefits of architectural bias. ***Advances in Neural Information Processing Systems*** (2019).

17. d'Ascoli, S. *et al.* ConViT: Improving Vision Transformers with Soft Convolutional Inductive Biases. ***International Conference on Machine Learning*** (2021).

18. d'Ascoli, S., Sagun, L., Biroli, G. & Morcos, A. Transformed CNNs: recasting pre-trained convolutional layers with self-attention. *arXiv preprint arXiv:2106.05795* (2021).

19. d'Ascoli, S., Refinetti, M. & Biroli, G. On the optimal learning rate schedule in non-convex optimization landscapes. *arXiv preprint arXiv:2202.04509* (2022).

20. Refinetti, M., d'Ascoli, S., Ohana, R. & Goldt, S. The dynamics of learning with feedback alignment. ***International Conference on Machine Learning*** (2020).

21. Refinetti, M., d'Ascoli, S., Ohana, R. & Goldt, S. The dynamics of learning with feedback alignment. ***Journal of Physics A (special issue)*** (2022).

22. d'Ascoli, S., Kamienny, P.-A., Lample, G. & Charton, F. Deep symbolic regression for recurrent sequences. ***International Conference on Machine Learning*** (2022).

23. Kamienny, P.-A., d'Ascoli, S., Lample, G. & Charton, F. End-to-end symbolic regression with transformers. *arXiv preprint arXiv:2204.51222* (2022).

24. d'Ascoli, S., Coucke, A., Caltagirone, F., Caulier, A. & Lelarge, M. Conditioned Text Generation with Transfer for Closed-Domain Dialogue Systems. ***Statistical Language and Speech Processing*** (2020).

25. d'Ascoli, S. *Comprendre la révolution de l'intelligence artificielle* (**First**, 2020).

26. d'Ascoli, S. *L'Intelligence Artificielle en 5 minutes par jour* (**First**, 2020).

27. d'Ascoli, S. & Touati, A. *Voyage au coeur de l'espace-temps* (**First**, 2021).

28. d'Ascoli, S. & Bouscal, A. *Voyage au coeur de l'atome* (**First**, 2022).

29. Zhang, J. Basic neural units of the brain: neurons, synapses and action potential. *arXiv preprint arXiv:1906.01703* (2019).

30. Bountos, N. I., Michail, D. & Papoutsis, I. Learning class prototypes from Synthetic InSAR with Vision Transformers. *arXiv preprint arXiv:2201.03016* (2022).

31. Pedro, R. & Oliveira, A. L. Assessing the Impact of Attention and Self-Attention Mechanisms on the Classification of Skin Lesions. *arXiv preprint arXiv:2112.12748* (2021).

32. Belkin, M., Hsu, D., Ma, S. & Mandal, S. Reconciling modern machine learning and the bias-variance trade-off. *arXiv preprint arXiv:1812.11118* (2018).

33. Nakkiran, P., Kaplun, G., Bansal Y. amd Yang, T., Barak, B. & Sutskever, I. *Deep Double Descent: Where Bigger Models and More Data Hurt* 2019. arXiv: 1912.02292.

34. LeCun, Y., Bengio, Y., *et al.* Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks* **3361,** 1995 (1995).

35. He, K., Zhang, X., Ren, S. & Sun, J. *Deep residual learning for image recognition* in *Proceedings of the IEEE conference on computer vision and pattern recognition* (2016), 770–778.

36. Ioffe, S. & Szegedy, C. *Batch normalization: Accelerating deep network training by reducing internal covariate shift* in *International conference on machine learning* (2015), 448–456.

37. Advani, M. S. & Saxe, A. M. High-dimensional dynamics of generalization error in neural networks. *arXiv:1710.03667* (2017).

38. Neal, B. *et al.* A modern take on the bias-variance tradeoff in neural networks. *arXiv preprint arXiv:1810.08591* (2018).

39. Abu-Mostafa, Y. S. Hints and the VC dimension. *Neural Computation* **5,** 278–288 (1993).

40. Neyshabur, B., Tomioka, R. & Srebro, N. In search of the real inductive bias: On the role of implicit regularization in deep learning. *arXiv preprint arXiv:1412.6614* (2014).

41. Franz, S., Parisi, G., Urbani, P. & Zamponi, F. Universal spectrum of normal modes in low-temperature glasses. *Proceedings of the National Academy of Sciences* **112,** 14539–14544 (2015).

42. Wyart, M. On the Rigidity of Amorphous Solids. *Annales de Phys* **30,** 1–113 (2005).

43. J Liu, A., R Nagel, S., Saarloos, W. & Wyart, M. *The jamming scenario - an introduction and outlook* (OUP Oxford, June 2010).

44. Dauphin, Y. *et al. Identifying and attacking the saddle point problemin high-dimensional non-convex optimization* in *Advances in Neural Information Processing Systems* (2014), 2933–2941. http://papers.nips.cc/paper/5486-identifying-and-attacking-the-saddle-point-problem-in-high-dimensional-non-convex-optimization.pdf.

45. Choromanska, A., Henaff, M., Mathieu, M., Ben Arous, G. & LeCun, Y. *The loss surfaces of multilayer networks* in *Artificial Intelligence and Statistics* (2015), 192–204.

46. Freeman, C. D. & Bruna, J. Topology and Geometry of Deep Rectified Network Optimization Landscapes. *International Conference on Learning Representations* (2017).

47. Venturi, L., Bandeira, A. & Bruna, J. Neural Networks with Finite Intrinsic Dimension have no Spurious Valleys. *arXiv preprint arXiv:1802.06384* (2018).

48. Hoffer, E., Hubara, I. & Soudry, D. *Train longer, generalize better: closing the generalization gap in large batch training of neural networks* in *Advances in Neural Information Processing Systems* (2017), 1729–1739.

49. Soudry, D. & Carmon, Y. No bad local minima: Data independent training error guarantees for multilayer neural networks. *arXiv preprint arXiv:1605.08361* (2016).

50. Cooper, Y. The loss landscape of overparameterized neural networks. *arXiv preprint arXiv:1804.10200* (2018).

51. Sagun, L., Bottou, L. & LeCun, Y. Singularity of the Hessian in Deep Learning. *International Conference on Learning Representations* (2017).

52. Sagun, L., Evci, U., Güney, V. U., Dauphin, Y. & Bottou, L. Empirical Analysis of the Hessian of Over-Parametrized Neural Networks. *ICLR 2018 Workshop Contribution, arXiv:1706.04454* (2017).

53. Ballard, A. J. *et al.* Energy landscapes for machine learning. *Physical Chemistry Chemical Physics* (2017).

54. Lipton, Z. C. Stuck in a what? adventures in weight space. *International Conference on Learning Representations* (2016).

55. Baity-Jesi, M. *et al. Comparing Dynamics: Deep Neural Networks versus Glassy Systems* in *Proceedings of the 35th International Conference on Machine Learning* (2018), 314–323.

56. Montufar, G. F., Pascanu, R., Cho, K. & Bengio, Y. *On the number of linear regions of deep neural networks* in *Advances in neural information processing systems* (2014), 2924–2932.

57. Bianchini, M. & Scarselli, F. On the complexity of neural network classifiers: A comparison between shallow and deep architectures. *IEEE transactions on neural networks and learning systems* **25,** 1553–1565 (2014).

58. Raghu, M., Poole, B., Kleinberg, J., Ganguli, S. & Sohl-Dickstein, J. *On the Expressive Power of Deep Neural Networks* in *Proceedings of the 34th International Conference on Machine Learning* (eds Precup, D. & Teh, Y. W.) **70** (PMLR, International Convention Centre, Sydney, Australia, June 2017), 2847–2854. http://proceedings.mlr.press/v70/raghu17a.html.

59. Eldan, R. & Shamir, O. *The power of depth for feedforward neural networks* in *Conference on Learning Theory* (2016), 907–940.

60. Lee, H., Ge, R., Ma, T., Risteski, A. & Arora, S. *On the Ability of Neural Nets to Express Distributions* in *Proceedings of the 2017 Conference on Learning Theory* (eds Kale, S. & Shamir, O.) **65** (PMLR, Amsterdam, Netherlands, July 2017), 1271–1296. http://proceedings.mlr.press/v65/lee17a.html.

61. Gardner, E. The space of interactions in neural network models. *Journal of physics A: Mathematical and general* **21,** 257 (1988).

62. Monasson, R. & Zecchina, R. Weight space structure and internal representations: a direct approach to learning and generalization in multilayer neural networks. *Physical review letters* **75,** 2432 (1995).

63. Zhang, C., Bengio, S., Hardt, M., Recht, B. & Vinyals, O. Understanding deep learning requires rethinking generalization. *International Conference on Learning Representations* (2017).

64. Baum, E. B. On the capabilities of multilayer perceptrons. *Journal of complexity* **4,** 193–215 (1988).

65. Caruana, R., Lawrence, S. & Giles, C. L. *Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping* in *Advances in neural information processing systems* (2001), 402–408.

66. Prechelt, L. in *Neural Networks: Tricks of the trade* 55–69 (Springer, 1998).

67. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. & Salakhutdinov, R. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research* **15,** 1929–1958 (2014).

68. Krogh, A. & Hertz, J. A. *A simple weight decay can improve generalization* in *Advances in neural information processing systems* (1992), 950–957.

69. Neyshabur, B., Tomioka, R., Salakhutdinov, R. & Srebro, N. Geometry of optimization and implicit regularization in deep learning. *arXiv preprint arXiv:1705.03071* (2017).

70. Neyshabur, B., Li, Z., Bhojanapalli, S., LeCun, Y. & Srebro, N. Towards understanding the role of over-parametrization in generalization of neural networks. *arXiv preprint arXiv:1805.12076* (2018).

71. Bansal, Y., Advani, M., Cox, D. D. & Saxe, A. M. Minnorm training: an algorithm for training over-parameterized deep neural networks. *CoRR* (2018).

72. Le Cun, Y., Kanter, I. & Solla, S. A. Eigenvalues of covariance matrices: Application to neural-network learning. *Physical Review Letters* **66,** 2396 (1991).

73. Krogh, A. & Hertz, J. A. Generalization in a linear perceptron in the presence of noise. *Journal of Physics A: Mathematical and General* **25,** 1135 (1992).

74. Duin, R. P. *Small sample size generalization* in *Proceedings of the Scandinavian Conference on Image Analysis* **2** (1995), 957–964.

75. Opper, M. & Kinzel, W. in *Models of neural networks III* 151–209 (Springer, 1996).

76. Engel, A. & Van den Broeck, C. *Statistical mechanics of learning* (Cambridge University Press, 2001).

77. He, K., Zhang, X., Ren, S. & Sun, J. *Deep residual learning for image recognition* in *Proceedings of the IEEE conference on computer vision and pattern recognition* (2016), 770–778.

78. Vaswani, A. *et al. Attention is all you need* in *Advances in neural information processing systems* (2017), 5998–6008.

79. Hastie, T., Montanari, A., Rosset, S. & Tibshirani, R. J. Surprises in high-dimensional ridgeless least squares interpolation. *arXiv preprint arXiv:1903.08560* (2019).

80. Belkin, M., Hsu, D. & Xu, J. Two models of double descent for weak features. *arXiv preprint arXiv:1903.07571* (2019).

81. Liao, Z. & Couillet, R. The Dynamics of Learning: A Random Matrix Approach. *arXiv preprint arXiv:1805.11917* (2018).

82. Mei, S. & Montanari, A. *The generalization error of random features regression: Precise asymptotics and double descent curve* arXiv:1908.05355. 2019. arXiv: 1908.05355.

83. Neyshabur, B. Implicit regularization in deep learning. *arXiv preprint arXiv:1709.01953* (2017).

84. Marčenko, V. A. & Pastur, L. A. Distribution of eigenvalues for some sets of random matrices. *Mathematics of the USSR-Sbornik* **1,** 457 (1967).

85. Opper, M. Statistical mechanics of learning: Generalization. *The Handbook of Brain Theory and Neural Networks,* 922–925 (1995).

86. Geiger, M. *et al.* The jamming transition as a paradigm to understand the loss landscape of deep neural networks. *arXiv preprint arXiv:1809.09349* (2018).

87. Tkachenko, A. V. & Witten, T. A. Stress propagation through frictionless granular material. *Phys. Rev. E* **60,** 687–696 (July 1999).

88. Düring, G., Lerner, E. & Wyart, M. Phonon gap and localization lengths in floppy materials. *Soft Matter* **9,** 146–154 (2013).

89. Wyart, M., Silbert, L. E., Nagel, S. R. & Witten, T. A. Effects of compression on the vibrational modes of marginally jammed solids. *Physical Review E* **72,** 051306 (2005).

90. Rosset, S., Zhu, J. & Hastie, T. J. *Margin maximizing loss functions* in *Advances in neural information processing systems* (2004), 1237–1244.

91. Chizat, L., Oyallon, E. & Bach, F. in *Advances in Neural Information Processing Systems 32* (eds Wallach, H. *et al.*) 2933–2943 (Curran Associates, Inc., 2019).

92. Arora, S. *et al.* On exact computation with an infinitely wide neural net. *arXiv preprint arXiv:1904.11955* (2019).

93. Mei, S., Misiakiewicz, T. & Montanari, A. *Mean-field theory of two-layers neural networks: dimension-free bounds and kernel limit* arXiv:1902.06015. 2019. arXiv: 1902.06015.

94. Geiger, M. *et al.* Scaling description of generalization with number of parameters in deep learning. *arXiv preprint arXiv:1901.01608* (2019).

95. Rahimi, A. & Recht, B. *Random features for large-scale kernel machines* in *Advances in neural information processing systems* (2008), 1177–1184.

96. Woodworth, B., Gunasekar, S., Lee, J., Soudry, D. & Srebro, N. Kernel and Deep Regimes in Overparametrized Models. *arXiv preprint arXiv:1906.05827* (2019).

97. Geiger, M., Spigler, S., Jacot, A. & Wyart, M. Disentangling feature and lazy learning in deep neural networks: an empirical study. *arXiv preprint arXiv:1906.08034* (2019).

98. Arora, S. *et al.* Harnessing the power of infinitely wide deep nets on small-data tasks. *arXiv preprint arXiv:1910.01663* (2019).

99. Daniely, A., Frostig, R. & Singer, Y. *Toward deeper understanding of neural networks: The power of initialization and a dual view on expressivity* in *Advances In Neural Information Processing Systems* (2016), 2253–2261.

100. Lee, J. *et al.* Deep neural networks as gaussian processes. *arXiv preprint arXiv:1711.00165* (2017).

101. Jacot, A., Gabriel, F. & Hongler, C. *Neural tangent kernel: Convergence and generalization in neural networks* in *Advances in Neural Information Processing Systems 32* (2018), 8571–8580.

102. Li, Y. & Liang, Y. Learning overparameterized neural networks via stochastic gradient descent on structured data. *Advances in Neural Information Processing Systems* **31** (2018).

103. Soltanolkotabi, M., Javanmard, A. & Lee, J. D. Theoretical insights into the optimization landscape of over-parameterized shallow neural networks. *IEEE Transactions on Information Theory* **65,** 742–769 (2018).

104. Kawaguchi, K. & Huang, J. *Gradient descent finds global minima for generalizable deep neural networks of practical sizes* in *2019 57th Annual Allerton Conference on Communication, Control, and Computing (Allerton)* (2019), 92–99.

105. Daniely, A. Neural networks learning and memorization with (almost) no over-parameterization. *Advances in Neural Information Processing Systems* **33,** 9007–9016 (2020).

106. Bresler, G. & Nagaraj, D. *A corrective view of neural networks: Representation, memorization and learning* in *Conference on Learning Theory* (2020), 848–901.

107. Montanari, A. & Zhong, Y. The interpolation phase transition in neural networks: Memorization and generalization under lazy training. *arXiv preprint arXiv:2007.12826* (2020).

108. Bubeck, S., Eldan, R., Lee, Y. T. & Mikulincer, D. Network size and weights size for memorization with two-layers neural networks. *arXiv preprint arXiv:2006.02855* (2020).

109. Mézard, M., Parisi, G. & Virasoro, M. *Spin glass theory and beyond: An Introduction to the Replica Method and Its Applications* (World Scientific Publishing Company, 1987).

110. Seung, H. S., Sompolinsky, H. & Tishby, N. Statistical mechanics of learning from examples. *Physical review A* **45,** 6056 (1992).

111. Advani, M., Lahiri, S. & Ganguli, S. Statistical mechanics of complex neural systems and high dimensional data. *Journal of Statistical Mechanics: Theory and Experiment* **2013,** P03014 (2013).

112. Zdeborová, L. & Krzakala, F. Statistical physics of inference: Thresholds and algorithms. *Advances in Physics* **65,** 453–552 (2016).

113. Livan, G., Novaes, M. & Vivo, P. *Introduction to random matrices: theory and practice* (Springer, 2018).

114. Tarquini, E., Biroli, G. & Tarzia, M. Level statistics and localization transitions of levy matrices. *Physical review letters* **116,** 010601 (2016).

115. Aggarwal, A., Lopatto, P. & Yau, H.-T. GOE statistics for Levy matrices. *arXiv:1806.07363* (2018).

116. Gabrié, M. Mean-field inference methods for neural networks. *Journal of Physics A: Mathematical and Theoretical* **53** (2020).

117. Bahri, Y. *et al.* Statistical Mechanics of Deep Learning. *Annual Review of Condensed Matter Physics* **11,** 501–528 (2020).

118. Nakkiran, P. More Data Can Hurt for Linear Regression: Sample-wise Double Descent. *arXiv preprint arXiv:1912.07242* (2019).

119. d'Ascoli, S., Sagun, L. & Biroli, G. Triple descent and the two kinds of overfitting: Where & why do they appear? *arXiv preprint arXiv:2006.03509* (2020).

120. Deng, Z., Kammoun, A. & Thrampoulidis, C. A Model of Double Descent for High-dimensional Binary Linear Classification. *arXiv preprint arXiv:1911.05822* (2019).

121. Kini, G. & Thrampoulidis, C. Analytic Study of Double Descent in Binary Classification: The Impact of Loss. *arXiv preprint arXiv:2001.11572* (2020).

122. Gerace, F., Loureiro, B., Krzakala, F., Mézard, M. & Zdeborová, L. *Generalisation error in learning with random features and the hidden manifold model* arXiv:2002.09339. 2020. arXiv: 2002.09339.

123. Rocks, J. W. & Mehta, P. Bias-variance decomposition of overparameterized regression with random linear features. *arXiv preprint arXiv:2203.05443* (2022).

124. Rocks, J. W. & Mehta, P. Memorizing without overfitting: Bias, variance, and interpolation in over-parameterized models. *arXiv preprint arXiv:2010.13933* (2020).

125. Loureiro, B., Gerbelot, C., Refinetti, M., Sicuro, G. & Krzakala, F. Fluctuations, Bias, Variance & Ensemble of Learners: Exact Asymptotics for Convex Losses in High-Dimension. *arXiv preprint arXiv:2201.13383* (2022).

126. Adlam, B. & Pennington, J. Understanding double descent requires a fine-grained bias-variance decomposition. *Advances in neural information processing systems* **33,** 11022–11032 (2020).

127. Lin, L. & Dobriban, E. What causes the test error? going beyond bias-variance via anova. *Journal of Machine Learning Research* **22,** 1–82 (2021).

128. Goldt, S., Mézard, M., Krzakala, F. & Zdeborová, L. *Modelling the influence of data structure on learning in neural networks* arXiv:1909.11500. 2019. arXiv: 1909.11500.

129. Bun, J., Bouchaud, J.-P. & Potters, M. Cleaning correlation matrices. *Risk magazine* **2015** (2016).

130. Abu-Mostafa, Y. S., Magdon-Ismail, M. & Lin, H.-T. *Learning from data* (AMLBook New York, NY, USA: 2012).

131. Drucker, H., Cortes, C., Jackel, L. D., LeCun, Y. & Vapnik, V. Boosting and other ensemble methods. *Neural Computation* **6,** 1289–1301 (1994).

132. Jacot, A., Şimşek, B., Spadaro, F., Hongler, C. & Gabriel, F. Implicit Regularization of Random Feature Models. *arXiv preprint arXiv:2002.08404* (2020).

133. Zhang, Y., Duchi, J. & Wainwright, M. Divide and conquer kernel ridge regression: A distributed algorithm with minimax optimal rates. *The Journal of Machine Learning Research* **16,** 3299–3340 (2015).

134. Chawla, N. V. *et al.* Distributed learning with bagging-like performance. *Pattern recognition letters* **24,** 455–471 (2003).

135. Geiger, M. *et al.* Scaling description of generalization with number of parameters in deep learning. *Journal of Statistical Mechanics: Theory and Experiment* **2020,** 023401 (2020).

136. Spigler, S. *et al.* A jamming transition from under-to over-parametrization affects generalization in deep learning. *Journal of Physics A: Mathematical and Theoretical* **52,** 474001 (2019).

137. Emami, M., Sahraee-Ardakan, M., Pandit, P., Rangan, S. & Fletcher, A. K. Generalization Error of Generalized Linear Models in High Dimensions. *arXiv preprint arXiv:2005.00180* (2020).

138. Aubin, B., Krzakala, F., Lu, Y. M. & Zdeborová, L. Generalization error in high-dimensional perceptrons: Approaching Bayes error with convex optimization. *arXiv preprint arXiv:2006.06560* (2020).

139. Li, Z., Xie, C. & Wang, Q. Provable More Data Hurt in High Dimensional Least Squares Estimator. *arXiv preprint arXiv:2008.06296* (2020).

140. Saxe, A. M., McClelland, J. L. & Ganguli, S. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv preprint arXiv:1312.6120* (2013).

141. Lampinen, A. K. & Ganguli, S. An analytic theory of generalization dynamics and transfer learning in deep linear networks. *arXiv preprint arXiv:1809.10374* (2018).

142. Loog, M. & Duin, R. P. *The dipping phenomenon* in *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)* (2012), 310–317.

143. Loog, M., Viering, T. & Mey, A. *Minimizers of the empirical risk and risk monotonicity* in *Advances in Neural Information Processing Systems* (2019), 7478–7487.

144. Min, Y., Chen, L. & Karbasi, A. The curious case of adversarially robust models: More data can help, double descend, or hurt generalization. *arXiv preprint arXiv:2002.11080* (2020).

145. Liang, T., Rakhlin, A. & Zhai, X. On the risk of minimum-norm interpolants and restricted lower isometry of kernels. *arXiv preprint arXiv:1908.10292* (2019).

146. Nakkiran, P., Venkat, P., Kakade, S. & Ma, T. Optimal regularization can mitigate double descent. *arXiv preprint arXiv:2003.01897* (2020).

147. Chen, L., Min, Y., Belkin, M. & Karbasi, A. Multiple Descent: Design Your Own Generalization Curve. *arXiv preprint arXiv:2008.01036* (2020).

148. Adlam, B. & Pennington, J. The Neural Tangent Kernel in High Dimensions: Triple Descent and a Multi-Scale Theory of Generalization. *arXiv preprint arXiv:2008.06786* (2020).

149. Pennington, J. & Worah, P. *Nonlinear random matrix theory for deep learning* in *Advances in Neural Information Processing Systems* (2017), 2637–2646.

150. Péché, S. *et al.* A note on the Pennington-Worah distribution. *Electronic Communications in Probability* **24** (2019).

151. Goldt, S., Reeves, G., Mézard, M., Krzakala, F. & Zdeborová, L. The Gaussian equivalence of generative models for learning with two-layer neural networks (2020).

152. Benigni, L. & Péché, S. Eigenvalue distribution of nonlinear models of random matrices. *arXiv preprint arXiv:1904.03090* (2019).

153. Adlam, B., Levinson, J. & Pennington, J. A Random Matrix Perspective on Mixtures of Nonlinearities for Deep Learning. *arXiv:1912.00827* (2019).

154. Liao, Z. & Couillet, R. On the spectrum of random features maps of high dimensional data. *arXiv preprint arXiv:1805.11916* (2018).

155. Dupic, T. & Castillo, I. P. Spectral density of products of wishart dilute random matrices. part i: the dense case. *arXiv preprint arXiv:1401.7802* (2014).

156. Akemann, G., Ipsen, J. R. & Kieburg, M. Products of rectangular random matrices: singular values and progressive scattering. *Physical Review E* **88,** 052118 (2013).

157. Spigler, S., Geiger, M. & Wyart, M. Asymptotic learning curves of kernel methods: empirical data vs Teacher-Student paradigm. *arXiv preprint arXiv:1905.10843* (2019).

158. Ansuini, A., Laio, A., Macke, J. H. & Zoccolan, D. *Intrinsic dimension of data representations in deep neural networks* in *Advances in Neural Information Processing Systems* (2019), 6109–6119.

159. Pope, P., Zhu, C., Abdelkader, A., Goldblum, M. & Goldstein, T. The Intrinsic Dimension of Images and Its Impact on Learning. *arXiv preprint arXiv:2104.08894* (2021).

160. Baratin, A. *et al.* Implicit Regularization in Deep Learning: A View from Function Space. *arXiv preprint arXiv:2008.00938* (2020).

161. Barbier, J., Krzakala, F., Macris, N., Miolane, L. & Zdeborová, L. Optimal errors and phase transitions in high-dimensional generalized linear models. *Proceedings of the National Academy of Sciences* **116,** 5451–5460 (2019).

162. Aubin, B. *et al. The committee machine: Computational to statistical gaps in learning a two-layers neural network* in *Neural Information Processing Systems 2018* (2018), 1–44.

163. Gabrié, M. *et al. Entropy and mutual information in models of deep neural networks* in *Advances in Neural Information Processing Systems 31* (2018), 1826–1836.

164. Hu, H. & Lu, Y. M. Universality laws for high-dimensional learning with random features. *arXiv preprint arXiv:2009.07669* (2020).

165. Loureiro, B. *et al.* Capturing the learning curves of generic features maps for realistic data sets with a teacher-student model. **2102.08127** (2021).

166. Hui, L. & Belkin, M. Evaluation of neural architectures trained with square loss vs cross-entropy in classification tasks. *arXiv preprint arXiv:2006.07322* (2020).

167. Demirkaya, A., Chen, J. & Oymak, S. *Exploring the role of loss functions in multiclass classification* in *2020 54th Annual Conference on Information Sciences and Systems (CISS)* (2020), 1–5.

168. Liao, Z., Couillet, R. & Mahoney, M. W. A random matrix analysis of random Fourier features: beyond the Gaussian kernel, a precise phase transition, and the corresponding double descent. *arXiv preprint arXiv:2006.05013* (2020).

169. Canatar, A., Bordelon, B. & Pehlevan, C. Statistical Mechanics of Generalization in Kernel Regression. *arXiv preprint arXiv:2006.13198* (2020).

170. Dobriban, E. & Wager, S. High-dimensional asymptotics of prediction: Ridge regression and classification. *arXiv preprint arXiv:1507.03003* (2015).

171. Wu, D. & Xu, J. On the Optimal Weighted l2 Regularization in Overparameterized Linear Regression. *arXiv preprint arXiv:2006.05800* (2020).

172. Richards, D., Mourtada, J. & Rosasco, L. Asymptotics of Ridge (less) Regression under General Source Condition. *arXiv preprint arXiv:2006.06386* (2020).

173. Kobak, D., Lomond, J. & Sanchez, B. Optimal ridge penalty for real-world high-dimensional data can be zero or negative due to the implicit ridge regularization. *liver* **1,** 1–2 (2019).

174. Ghorbani, B., Mei, S., Misiakiewicz, T. & Montanari, A. When do neural networks outperform kernel methods? *Advances in Neural Information Processing Systems* **33** (2020).

175. Soudry, D., Hoffer, E., Nacson, M. S., Gunasekar, S. & Srebro, N. The implicit bias of gradient descent on separable data. *Journal of Machine Learning Research (JMLR)* **19,** 1–57 (2018).

176. Montanari, A., Ruan, F., Sohn, Y. & Yan, J. *The generalization error of max-margin linear classifiers: High-dimensional asymptotics in the overparametrized regime* arXiv:1911.01544. 2019. arXiv: 1911.01544.

177. Chatterji, N. S. & Long, P. M. Finite-sample analysis of interpolating linear classifiers in the overparameterized regime. *arXiv preprint arXiv:2004.12019* (2020).

178. Liang, T. & Sur, P. A precise high-dimensional asymptotic theory for boosting and min-l1-norm interpolated classifiers. *arXiv preprint arXiv:2002.01586* (2020).

179. Mignacco, F., Krzakala, F., Lu, Y. M. & Zdeborová, L. The role of regularization in classification of high-dimensional noisy Gaussian mixture. *arXiv preprint arXiv:2002.11544* (2020).

180. Muthukumar, V. *et al.* Classification vs regression in overparameterized regimes: Does the loss function matter? *arXiv preprint arXiv:2005.08054* (2020).

181. El Karoui, N. *et al.* The spectrum of kernel random matrices. *Annals of statistics* **38,** 1–50 (2010).

182. Scherer, D., Müller, A. & Behnke, S. *Evaluation of Pooling Operations in Convolutional Architectures for Object Recognition* en. in *Artificial Neural Networks – ICANN 2010* (eds Diamantaras, K., Duch, W. & Iliadis, L. S.) (Springer, Berlin, Heidelberg, 2010), 92–101. ISBN: 978-3-642-15825-4.

183. Schmidhuber, J. Deep learning in neural networks: An overview. en. *Neural Networks* **61,** 85–117. ISSN: 0893-6080. http://www.sciencedirect.com/science/article/pii/S0893608014002135 (2021) (Jan. 2015).

184. Goodfellow, I., Bengio, Y. & Courville, A. *Deep Learning* (MIT Press, 2016).

185. Simoncelli, E. P. & Olshausen, B. A. Natural image statistics and neural representation. *Annual review of neuroscience* **24,** 1193–1216 (2001).

186. Ruderman, D. L. & Bialek, W. Statistics of natural images: Scaling in the woods. *Physical review letters* **73,** 814 (1994).

187. Geiger, M. *et al.* The jamming transition as a paradigm to understand the loss landscape of deep neural networks. *arXiv preprint arXiv:1809.09349* (2018).

188. Nowlan, S. J. & Hinton, G. E. Simplifying neural networks by soft weight-sharing. *Neural computation* **4,** 473–493 (1992).

189. Buciluǎ, C., Caruana, R. & Niculescu-Mizil, A. *Model compression* in *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining* (2006), 535–541.

190. Hinton, G., Vinyals, O. & Dean, J. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531* (2015).

191. Ba, J. & Caruana, R. *Do deep nets really need to be deep?* in *Advances in neural information processing systems* (2014), 2654–2662.

192. Han, S., Mao, H. & Dally, W. J. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149* (2015).

193. Frankle, J. & Carbin, M. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635* (2018).

194. Chen, T., Goodfellow, I. & Shlens, J. Net2net: Accelerating learning via knowledge transfer. *arXiv preprint arXiv:1511.05641* (2015).

195. Saxena, S. & Verbeek, J. *Convolutional neural fabrics* in *Advances in Neural Information Processing Systems* (2016), 4053–4061.

196. Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M. & Tang, P. T. P. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836* (2016).

197. Baldassi, C., Lauditi, C., Malatesta, E. M., Perugini, G. & Zecchina, R. Unveiling the structure of wide flat minima in neural networks. *Physical Review Letters* **127,** 278301 (2021).

198. Wu, L., Zhu, Z., *et al.* Towards understanding generalization of deep learning: Perspective of loss landscapes. *arXiv preprint arXiv:1706.10239* (2017).

199. Chaudhari, P. *et al.* Entropy-sgd: Biasing gradient descent into wide valleys. *arXiv preprint arXiv:1611.01838* (2016).

200. Pittorino, F. *et al.* Entropic gradient descent algorithms and wide flat minima. *Journal of Statistical Mechanics: Theory and Experiment* **2021,** 124015 (2021).

201. Freeman, C. D. & Bruna, J. Topology and Geometry of Deep Rectified Network Optimization Landscapes. *arXiv preprint arXiv:1611.01540* (2016).

202. Garipov, T., Izmailov, P., Podoprikhin, D., Vetrov, D. P. & Wilson, A. G. *Loss surfaces, mode connectivity, and fast ensembling of dnns* in *Advances in Neural Information Processing Systems* (2018), 8789–8798.

203. Draxler, F., Veschgini, K., Salmhofer, M. & Hamprecht, F. A. Essentially no barriers in neural network energy landscape. *arXiv preprint arXiv:1803.00885* (2018).

204. Baldassi, C. *et al.* Unreasonable effectiveness of learning neural networks: From accessible states and robust ensembles to basic algorithmic schemes. *Proceedings of the National Academy of Sciences* **113,** E7655–E7662 (2016).

205. Baldassi, C., Pittorino, F. & Zecchina, R. Shaping the learning landscape in neural networks around wide flat minima. *arXiv preprint arXiv:1905.07833* (2019).

206. Novak, R. *et al.* Bayesian deep convolutional networks with many channels are gaussian processes. *arXiv preprint arXiv:1810.05148* (2018).

207. Du, S. S. *et al. How Many Samples are Needed to Estimate a Convolutional Neural Network?* in *Advances in Neural Information Processing Systems* (2018), 373–383.

208. Long, P. M. & Sedghi, H. Size-free generalization bounds for convolutional neural networks. *arXiv preprint arXiv:1905.12600* (2019).

209. Lee, J. & Raginsky, M. Learning finite-dimensional coding schemes with nonlinear reconstruction maps. *arXiv preprint arXiv:1812.09658* (2018).

210. Neyshabur, B. Towards learning convolutions from scratch. *Advances in Neural Information Processing Systems* **33,** 8078–8088 (2020).

211. Pellegrini, F. & Biroli, G. Sifting out the features by pruning: Are convolutional networks the winning lottery ticket of fully connected ones? *arXiv preprint arXiv:2104.13343* (2021).

212. Ingrosso, A. & Goldt, S. Data-driven emergence of convolutional structure in neural networks. *arXiv preprint arXiv:2202.00565* (2022).

213. Golatkar, A., Achille, A. & Soatto, S. Time Matters in Regularizing Deep Networks: Weight Decay and Data Augmentation Affect Early Learning Dynamics, Matter Little Near Convergence. *arXiv preprint arXiv:1905.13277* (2019).

214. Jastrzebski, S. *et al.* Three Factors Influencing Minima in SGD. *arXiv preprint arXiv:1711.04623* (2017).

215. Achille, A., Rovere, M. & Soatto, S. Critical learning periods in deep neural networks. *arXiv preprint arXiv:1711.08856* (2017).

216. Anandkumar, A., Deng, Y., Ge, R. & Mobahi, H. Homotopy analysis for tensor pca. *arXiv preprint arXiv:1610.09322* (2016).

217. Touvron, H. *et al.* Training data-efficient image transformers & distillation through attention. *arXiv preprint arXiv:2012.12877* (2020).

218. Mitchell, T. M. *The need for biases in learning generalizations* (Department of Computer Science, Laboratory for Computer Science Research …, 1980).

219. LeCun, Y., Bottou, L., Bengio, Y. & Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE* **86,** 2278–2324 (1998).

220. LeCun, Y. *et al.* Backpropagation applied to handwritten zip code recognition. *Neural computation* **1,** 541–551 (1989).

221. Krizhevsky, A., Sutskever, I. & Hinton, G. E. Imagenet classification with deep convolutional neural networks. *Communications of the ACM* **60,** 84–90 (2017).

222. Gers, F. A., Schmidhuber, J. & Cummins, F. Learning to forget: Continual prediction with LSTM (1999).

223. Sundermeyer, M., Schlüter, R. & Ney, H. *LSTM neural networks for language modeling* in *Thirteenth annual conference of the international speech communication association* (2012).

224. Greff, K., Srivastava, R. K., Koutník, J., Steunebrink, B. R. & Schmidhuber, J. LSTM: A Search Space Odyssey. *IEEE Transactions on Neural Networks and Learning Systems* **28.** Conference Name: IEEE Transactions on Neural Networks and Learning Systems, 2222–2232. ISSN: 2162-2388 (Oct. 2017).

225. Bahdanau, D., Cho, K. & Bengio, Y. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473* (2014).

226. Devlin, J., Chang, M.-W., Lee, K. & Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

227. Chen, Y., Kalantidis, Y., Li, J., Yan, S. & Feng, J. A2-Nets: Double Attention Networks. *arXiv preprint arXiv:1810.11579* (2018).

228. Bello, I., Zoph, B., Vaswani, A., Shlens, J. & Le, Q. V. *Attention augmented convolutional networks* in *Proceedings of the IEEE International Conference on Computer Vision* (2019), 3286–3295.

229. Ramachandran, P. *et al.* Stand-alone self-attention in vision models. *arXiv preprint arXiv:1906.05909* (2019).

230. Tan, M. & Le, Q. *Efficientnet: Rethinking model scaling for convolutional neural networks* in *International Conference on Machine Learning* (2019), 6105–6114.

231. Wu, B. *et al.* Visual Transformers: Token-based Image Representation and Processing for Computer Vision. *arXiv:2006.03677 [cs, eess]*. arXiv: 2006.03677. http://arxiv.org/abs/2006.03677 (2020) (July 2020).

232. Yuan, L. *et al.* Tokens-to-Token ViT: Training Vision Transformers from Scratch on ImageNet. *arXiv preprint arXiv:2101.11986* (2021).

233. Carion, N. *et al.* End-to-End Object Detection with Transformers. *arXiv preprint arXiv:2005.12872* (2020).

234. Hu, H., Gu, J., Zhang, Z., Dai, J. & Wei, Y. *Relation networks for object detection* in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2018), 3588–3597.

235. Chen, Y.-C. *et al. Uniter: Universal image-text representation learning* in *European Conference on Computer Vision* (2020), 104–120.

236. Locatello, F. *et al.* Object-centric learning with slot attention. *arXiv preprint arXiv:2006.15055* (2020).

237. Sun, C., Myers, A., Vondrick, C., Murphy, K. & Schmid, C. *Videobert: A joint model for video and language representation learning* in *Proceedings of the IEEE International Conference on Computer Vision* (2019), 7464–7473.

238. Srinivas, A. *et al.* Bottleneck Transformers for Visual Recognition. *arXiv e-prints*, arXiv:2101.11605. arXiv: 2101.11605 [cs.CV] (Jan. 2021).

239. Elsayed, G., Ramachandran, P., Shlens, J. & Kornblith, S. *Revisiting spatial invariance with low-rank local connectivity* in *International Conference on Machine Learning* (2020), 2868–2879.

240. Hu, J., Shen, L., Albanie, S., Sun, G. & Vedaldi, A. in *Advances in Neural Information Processing Systems 31* (eds Bengio, S. *et al.*) 9401–9411 (Curran Associates, Inc., 2018). (2020).

241. Hu, J., Shen, L. & Sun, G. *Squeeze-and-Excitation Networks* in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition* ISSN: 2575-7075 (June 2018), 7132–7141.

242. Wang, X., Girshick, R., Gupta, A. & He, K. *Non-local Neural Networks* en. in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition* (IEEE, Salt Lake City, UT, USA, June 2018), 7794–7803. ISBN: 978-1-5386-6420-9. https://ieeexplore.ieee.org/document/8578911/ (2020).

243. Cordonnier, J.-B., Loukas, A. & Jaggi, M. On the relationship between self-attention and convolutional layers. *arXiv preprint arXiv:1911.03584* (2019).

244. Xu, Y., Zhang, Q., Zhang, J. & Tao, D. Vitae: Vision transformer advanced by exploring intrinsic inductive bias. *Advances in Neural Information Processing Systems* **34** (2021).

245. Zhang, Q., Xu, Y., Zhang, J. & Tao, D. ViTAEv2: Vision Transformer Advanced by Exploring Inductive Bias for Image Recognition and Beyond. *arXiv preprint arXiv:2202.10108* (2022).

246. Ye, D. *et al.* CSformer: Bridging Convolution and Transformer for Compressive Sensing. *arXiv preprint arXiv:2112.15299* (2021).

247. Yang, M. *et al.* Integrating convolution and self-attention improves language model of human genome for interpreting non-coding regions at base-resolution. *bioRxiv* (2021).

248. Wu, H. *et al. Cvt: Introducing convolutions to vision transformers* in *Proceedings of the IEEE/CVF International Conference on Computer Vision* (2021), 22–31.

249. Dai, Z., Liu, H., Le, Q. & Tan, M. Coatnet: Marrying convolution and attention for all data sizes. *Advances in Neural Information Processing Systems* **34** (2021).

250. Li, Y., Zhang, K., Cao, J., Timofte, R. & Van Gool, L. Localvit: Bringing locality to vision transformers. *arXiv preprint arXiv:2104.05707* (2021).

251. Yuan, K. *et al. Incorporating convolution designs into visual transformers* in *Proceedings of the IEEE/CVF International Conference on Computer Vision* (2021), 579–588.

252. Pan, X. *et al.* On the Integration of Self-Attention and Convolution. *arXiv preprint arXiv:2111.14556* (2021).

253. Wightman, R. *PyTorch Image Models* https://github.com/rwightman/pytorch-image-models. 2019.

254. Sukhbaatar, S., Grave, E., Bojanowski, P. & Joulin, A. Adaptive attention span in transformers. *arXiv preprint arXiv:1905.07799* (2019).

255. Touvron, H., Cord, M., Sablayrolles, A., Synnaeve, G. & Jégou, H. *Going deeper with Image Transformers* 2021. arXiv: 2103.17239 [cs.CV].

256. Zhai, X., Oliver, A., Kolesnikov, A. & Beyer, L. *S4l: Self-supervised semi-supervised learning* in *Proceedings of the IEEE/CVF International Conference on Computer Vision* (2019), 1476–1485.

257. Abnar, S., Dehghani, M. & Zuidema, W. Transferring inductive biases through knowledge distillation. *arXiv preprint arXiv:2006.00555* (2020).

258. Dong, Y., Cordonnier, J.-B. & Loukas, A. *Attention is not all you need: Pure attention loses rank doubly exponentially with depth* in *International Conference on Machine Learning* (2021), 2793–2803.

259. Touvron, H. *et al.* ResMLP: Feedforward networks for image classification with data-efficient training. *arXiv preprint arXiv:2105.03404* (2021).

260. Bello, I. Lambdanetworks: Modeling long-range interactions without attention. *arXiv preprint arXiv:2102.08602* (2021).

261. Wang, S., Li, B., Khabsa, M., Fang, H. & Ma, H. L. Self-Attention with Linear Complexity. *arXiv preprint arXiv:2006.04768* (2020).

262. Choromanski, K. *et al.* Rethinking attention with performers. *arXiv preprint arXiv:2009.14794* (2020).

263. Katharopoulos, A., Vyas, A., Pappas, N. & Fleuret, F. *Transformers are rnns: Fast autoregressive transformers with linear attention* in *International Conference on Machine Learning* (2020), 5156–5165.

264. Zhang, J. *et al.* Why are Adaptive Methods Good for Attention Models? *arXiv preprint arXiv:1912.03194* (2019).

265. Liu, L., Liu, X., Gao, J., Chen, W. & Han, J. Understanding the difficulty of training transformers. *arXiv preprint arXiv:2004.08249* (2020).

266. Jiang, Z. *et al. Token Labeling: Training a 85.4% Top-1 Accuracy Vision Transformer with 56M Parameters on ImageNet* 2021. arXiv: 2104.10858 [cs.CV].

267. Graham, B. *et al.* LeViT: a Vision Transformer in ConvNet's Clothing for Faster Inference. *arXiv preprint arXiv:2104.01136* (2021).

268. Chen, Z. *et al. Visformer: The Vision-friendly Transformer* 2021. arXiv: 2104.12533 [cs.CV].

269. Srinivas, A. *et al.* Bottleneck Transformers for Visual Recognition. *arXiv e-prints,* arXiv:2101.11605. arXiv: 2101.11605 [cs.CV] (Jan. 2021).

270. Wen, W., Yan, F., Chen, Y. & Li, H. *Autogrow: Automatic layer growing in deep convolutional networks* in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (2020), 833–841.

271. d'Ascoli, S. *ConViT: improving Vision Transformers with Soft Convolutional Inductive Biases* 2021. arXiv: 2102.10882 [cs.CV].

272. Bello, I. *et al.* Revisiting ResNets: Improved Training and Scaling Strategies. *arXiv preprint arXiv:2103.07579* (2021).

273. Touvron, H., Vedaldi, A., Douze, M. & Jégou, H. Fixing the train-test resolution discrepancy. *arXiv preprint arXiv:1906.06423* (2019).

274. Bhojanapalli, S. *et al.* Understanding robustness of transformers for image classification. *arXiv preprint arXiv:2103.14586* (2021).

275. Mao, X. *et al.* Rethinking the Design Principles of Robust Vision Transformer. *arXiv preprint arXiv:2105.07926* (2021).

276. Mahmood, K., Mahmood, R. & Van Dijk, M. On the robustness of vision transformers to adversarial examples. *arXiv preprint arXiv:2104.02610* (2021).

277. Shao, R., Shi, Z., Yi, J., Chen, P.-Y. & Hsieh, C.-J. On the adversarial robustness of visual transformers. *arXiv preprint arXiv:2103.15670* (2021).

278. Hendrycks, D. & Dietterich, T. Benchmarking Neural Network Robustness to Common Corruptions and Perturbations. *Proceedings of the International Conference on Learning Representations* (2019).

279. Goodfellow, I. J., Shlens, J. & Szegedy, C. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572* (2014).

280. Madry, A., Makelov, A., Schmidt, L., Tsipras, D. & Vladu, A. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083* (2017).

281. Hendrycks, D., Zhao, K., Basart, S., Steinhardt, J. & Song, D. Natural Adversarial Examples. *CVPR* (2021).

282. Hendrycks, D. *et al.* The Many Faces of Robustness: A Critical Analysis of Out-of-Distribution Generalization. *arXiv preprint arXiv:2006.16241* (2020).

283. Lewkowycz, A., Bahri, Y., Dyer, E., Sohl-Dickstein, J. & Gur-Ari, G. The large learning rate phase of deep learning: the catapult mechanism. *arXiv preprint arXiv:2003.02218* (2020).

284. Caron, M. *et al.* Emerging properties in self-supervised vision transformers. *arXiv preprint arXiv:2104.14294* (2021).

285. d'Ascoli, S., Sagun, L., Biroli, G. & Bruna, J. *Finding the Needle in the Haystack with Convolutions: on the benefits of architectural bias* in *Advances in Neural Information Processing Systems* (2019), 9334–9345.

286. Liu, G.-H. & Theodorou, E. A. Deep learning theory review: An optimal control and dynamical systems perspective. *arXiv preprint arXiv:1908.10920* (2019).

287. Li, Q., Tai, C. & Weinan, E. *Stochastic modified equations and adaptive stochastic gradient algorithms* in *International Conference on Machine Learning* (2017), 2101–2110.

288. Brea, J., Simsek, B., Illing, B. & Gerstner, W. Weight-space symmetry in deep networks gives rise to permutation saddles, connected by equal-loss valleys across the loss landscape. *arXiv preprint arXiv:1907.02911* (2019).

289. Choromanska, A., Henaff, M., Mathieu, M., Ben Arous, G. & LeCun, Y. *The loss surfaces of multilayer networks* in *Artificial Intelligence and Statistics* (2015), 192–204.

290. Ben Arous, G., Dembo, A. & Guionnet, A. Cugliandolo-Kurchan equations for dynamics of spin-glasses. *Probability theory and related fields* **136,** 619–660 (2006).

291. Dembo, A. & Subag, E. Dynamics for spherical spin glasses: disorder dependent initial conditions. *Journal of Statistical Physics* **181,** 465–514 (2020).

292. Arous, G. B., Gheissari, R. & Jagannath, A. Algorithmic thresholds for tensor PCA. *The Annals of Probability* **48,** 2052–2087 (2020).

293. Mannelli, S. S. & Zdeborová, L. Thresholds of descending algorithms in inference problems. *Journal of Statistical Mechanics: Theory and Experiment* **2020,** 034004 (2020).

294. Park, D., Sohl-Dickstein, J., Le, Q. & Smith, S. *The effect of network width on stochastic gradient descent and generalization: an empirical study* in *International Conference on Machine Learning* (2019), 5042–5051.

295. Smith, S. L., Kindermans, P.-J., Ying, C. & Le, Q. V. Don't decay the learning rate, increase the batch size. *arXiv preprint arXiv:1711.00489* (2017).

296. Mignacco, F. & Urbani, P. The effective noise of Stochastic Gradient Descent. *arXiv preprint arXiv:2112.10852* (2021).

297. Cheng, X., Yin, D., Bartlett, P. & Jordan, M. *Stochastic gradient and langevin processes* in *International Conference on Machine Learning* (2020), 1810–1819.

298. Mingard, C., Valle-Pérez, G., Skalse, J. & Louis, A. A. Is SGD a Bayesian sampler? Well, almost. *Journal of Machine Learning Research* **22,** 1–64 (2021).

299. Hu, W., Li, C. J., Li, L. & Liu, J.-G. On the diffusion approximation of nonconvex stochastic gradient descent. *arXiv preprint arXiv:1705.07562* (2017).

300. Moulines, E. & Bach, F. Non-asymptotic analysis of stochastic approximation algorithms for machine learning. *Advances in neural information processing systems* **24,** 451–459 (2011).

301. Xu, W. Towards optimal one pass large scale learning with averaged stochastic gradient descent. *arXiv preprint arXiv:1107.2490* (2011).

302. Sherrington, D. & Kirkpatrick, S. Solvable model of a spin-glass. *Physical review letters* **35,** 1792 (1975).

303. Mannelli, S. S. *et al.* Marvels and pitfalls of the langevin algorithm in noisy high-dimensional inference. *Physical Review X* **10,** 011057 (2020).

304. Cugliandolo, L. F. & Dean, D. S. Full dynamical solution for a spherical spin-glass model. *Journal of Physics A: Mathematical and General* **28,** 4213 (1995).

305. Barbier, D., Pimenta, P. H., Cugliandolo, L. F. & Stariolo, D. A. Finite size effects and loss of self-averageness in the relaxational dynamics of the spherical Sherrington-Kirkpatrick model. *arXiv preprint arXiv:2103.12654* (2021).

306. Cugliandolo, L. F. & Kurchan, J. Analytical solution of the off-equilibrium dynamics of a long-range spin-glass model. *Physical Review Letters* **71,** 173 (1993).

307. Sarao Mannelli, S., Biroli, G., Cammarota, C., Krzakala, F. & Zdeborová, L. Who is afraid of big bad minima? analysis of gradient-flow in spiked matrix-tensor models. *Advances in Neural Information Processing Systems* **32,** 8679–8689 (2019).

308. Ben Arous, G., Mei, S., Montanari, A. & Nica, M. The landscape of the spiked tensor model. *Communications on Pure and Applied Mathematics* **72,** 2282–2330 (2019).

309. Ros, V., Ben Arous, G., Biroli, G. & Cammarota, C. Complex energy landscapes in spiked-tensor and simple glassy models: Ruggedness, arrangements of local minima, and phase transitions. *Physical Review X* **9,** 011003 (2019).

310. Bottou, L. *Stochastic learning* in *Summer School on Machine Learning* (2003), 146–168.

311. You, K., Long, M., Wang, J. & Jordan, M. I. How does learning rate decay help modern neural networks? *arXiv preprint arXiv:1908.01878* (2019).

312. Li, Y., Wei, C. & Ma, T. *Towards explaining the regularization effect of initial large learning rate in training neural networks* in *Advances in Neural Information Processing Systems* (2019), 11674–11685.

313. Ge, R., Kakade, S. M., Kidambi, R. & Netrapalli, P. The step decay schedule: A near optimal, geometrically decaying learning rate procedure for least squares. *arXiv preprint arXiv:1904.12838* (2019).

314. Loshchilov, I. & Hutter, F. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983* (2016).

315. Smith, L. N. *Cyclical learning rates for training neural networks* in *2017 IEEE winter conference on applications of computer vision (WACV)* (2017), 464–472.

316. Lewkowycz, A. How to decay your learning rate. *arXiv preprint arXiv:2103.12682* (2021).

317. Goyal, P. *et al.* Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour. *arXiv preprint arXiv:1706.02677* (2017).

318. Gilmer, J. *et al.* A Loss Curvature Perspective on Training Instability in Deep Learning. *arXiv preprint arXiv:2110.04369* (2021).

319. Gotmare, A., Keskar, N. S., Xiong, C. & Socher, R. A closer look at deep learning heuristics: Learning rate restarts, warmup and distillation. *arXiv preprint arXiv:1810.13243* (2018).

320. Kingma, D. P. & Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

321. Zeiler, M. D. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701* (2012).

322. Duchi, J., Hazan, E. & Singer, Y. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research* **12** (2011).

323. Keskar, N. S. & Socher, R. Improving generalization performance by switching from adam to sgd. *arXiv preprint arXiv:1712.07628* (2017).

324. Chen, J. *et al.* Closing the generalization gap of adaptive gradient methods in training deep neural networks. *arXiv preprint arXiv:1806.06763* (2018).

325. Wilson, A. C., Roelofs, R., Stern, M., Srebro, N. & Recht, B. The marginal value of adaptive gradient methods in machine learning. *arXiv preprint arXiv:1705.08292* (2017).

326. Wigner, E. P. On the distribution of the roots of certain symmetric matrices. *Annals of Mathematics,* 325–327 (1958).

327. Berthier, L. & Biroli, G. Theoretical perspective on the glass transition and amorphous materials. *Reviews of modern physics* **83,** 587 (2011).

328. Auffinger, A., Ben Arous, G. & Černy, J. Random matrices and complexity of spin glasses. *Communications on Pure and Applied Mathematics* **66,** 165–201 (2013).

329. Crisanti, A. & Sommers, H.-J. The spherical p-spin interaction spin glass model: the statics. *Zeitschrift für Physik B Condensed Matter* **87,** 341–354 (1992).

330. Castellani, T. & Cavagna, A. Spin-glass theory for pedestrians. *Journal of Statistical Mechanics: Theory and Experiment* **2005,** P05012 (2005).

331. Thalmann, F. Geometrical approach for the mean-field dynamics of a particle in a short range correlated random potential. *The European Physical Journal B-Condensed Matter and Complex Systems* **19,** 49–63 (2001).

332. Bray, A. J. Theory of phase-ordering kinetics. *Advances in Physics* **51,** 481–587 (2002).

333. Bouchaud, J.-P., Cugliandolo, L. F., Kurchan, J. & Mezard, M. Out of equilibrium dynamics in spin-glasses and other glassy systems. *Spin glasses and random fields* **12,** 161 (1998).

334. Biroli, G. A crash course on ageing. *Journal of Statistical Mechanics: Theory and Experiment* **2005,** P05014 (2005).

335. Baik, J., Ben Arous, G., Péché, S., *et al.* Phase transition of the largest eigenvalue for nonnull complex sample covariance matrices. *The Annals of Probability* **33,** 1643–1697 (2005).

336. Tracy, C. A. & Widom, H. On orthogonal and symplectic matrix ensembles. *Communications in Mathematical Physics* **177,** 727–754 (1996).

337. Ben Arous, G., Gheissari, R. & Jagannath, A. A classification for the performance of online SGD for high-dimensional inference. *arXiv:2003.10409* (2020).

338. Power, A., Burda, Y., Edwards, H., Babuschkin, I. & Misra, V. *Grokking: Generalization beyond overfitting on small algorithmic datasets* in *ICLR MATH-AI Workshop* (2021).

339. Agoritsas, E., Biroli, G., Urbani, P. & Zamponi, F. Out-of-equilibrium dynamical mean-field equations for the perceptron model. *Journal of Physics A: Mathematical and Theoretical* **51,** 085002 (2018).

340. Mignacco, F., Krzakala, F., Urbani, P. & Zdeborová, L. Dynamical mean-field theory for stochastic gradient descent in Gaussian mixture classification. *arXiv preprint arXiv:2006.06098* (2020).

341. Celentano, M., Cheng, C. & Montanari, A. The high-dimensional asymptotics of first order methods with random data. *arXiv preprint arXiv:2112.07572* (2021).

342. Goldt, S., Advani, M. S., Saxe, A. M., Krzakala, F. & Zdeborová, L. Dynamics of stochastic gradient descent for two-layer neural networks in the teacher–student setup. *Journal of Statistical Mechanics: Theory and Experiment* **2020,** 124010 (2020).

343. Refinetti, M., D'Ascoli, S., Ohana, R. & Goldt, S. *Align, then memorise: the dynamics of learning with feedback alignment* in *Proceedings of the 38th International Conference on Machine Learning* (eds Meila, M. & Zhang, T.) **139** (PMLR, 2021), 8925–8935. http://proceedings.mlr.press/v139/refinetti21a.html.

344. Rumelhart, D. E., Hinton, G. E. & Williams, R. J. Learning representations by back-propagating errors. *Nature* **323,** 533–536 (1986).

345. Grossberg, S. Competitive learning: From interactive activation to adaptive resonance. *Cognitive science* **11,** 23–63 (1987).

346. Crick, F. The recent excitement about neural networks. *Nature* **337,** 129–132 (1989).

347. Lillicrap, T., Cownden, D., Tweed, D. & Akerman, C. Random synaptic feedback weights support error backpropagation for deep learning. *Nature Communications* **7,** 1–10 (2016).

348. Nøkland, A. *Direct Feedback Alignment Provides Learning in Deep Neural Networks* in *Advances in Neural Information Processing Systems 29* (2016).

349. Gilmer, J., Raffel, C., Schoenholz, S. S., Raghu, M. & Sohl-Dickstein, J. *Explaining the learning dynamics of direct feedback alignment* in *ICLR workshop track* (2017).

350. Bartunov, S. *et al. Assessing the scalability of biologically-motivated deep learning algorithms and architectures* in *Advances in Neural Information Processing Systems* (2018), 9368–9378.

351. Launay, J., Poli, I., Boniface, F. & Krzakala, F. *Direct Feedback Alignment Scales to Modern Deep Learning Tasks and Architectures* in *Advances in neural information processing systems* (2020).

352. Moskovitz, T. H., Litwin-Kumar, A. & Abbott, L. Feedback alignment in deep convolutional networks. *arXiv preprint arXiv:1812.06488* (2018).

353. Launay, J., Poli, I. & Krzakala, F. Principled Training of Neural Networks with Direct Feedback Alignment. *arXiv:1906.04554* (2019).

354. Han, D. & Yoo, H.-j. *Direct Feedback Alignment Based Convolutional Neural Network Training for Low-Power Online Learning Processor* in *Proceedings of the IEEE International Conference on Computer Vision Workshops* (2019).

355. Saad, D. & Solla, S. Exact Solution for On-Line Learning in Multilayer Neural Networks. *Phys. Rev. Lett.* **74,** 4337–4340 (1995).

356. Saad, D. & Solla, S. On-line learning in soft committee machines. *Phys. Rev. E* **52,** 4225–4243 (1995).

357. Biehl, M. & Schwarze, H. Learning by on-line gradient descent. *J. Phys. A. Math. Gen.* **28,** 643–656 (1995).

358. Frenkel, C., Lefebvre, M. & Bol, D. *Learning without feedback: Direct random target projection as a feedback-alignment algorithm with layerwise feedforward training* 2019. arXiv: 1909.01311.

359. Gardner, E. & Derrida, B. Three unfinished works on the optimal storage capacity of networks. *Journal of Physics A: Mathematical and General* **22,** 1983–1994 (1989).

360. Seung, H. S., Sompolinsky, H. & Tishby, N. Statistical mechanics of learning from examples. *Physical Review A* **45,** 6056–6091 (1992).

361. Watkin, T., Rau, A. & Biehl, M. The statistical mechanics of learning a rule. *Reviews of Modern Physics* **65,** 499–556 (1993).

362. Zdeborová, L. & Krzakala, F. Statistical physics of inference: thresholds and algorithms. *Adv. Phys.* **65,** 453–552 (2016).

363. Zhong, K., Song, Z., Jain, P., Bartlett, P. & Dhillon, I. *Recovery guarantees for one-hidden-layer neural networks* in *Proceedings of the 34th International Conference on Machine Learning-Volume 70* (2017), 4140–4149.

364. Advani, M. S., Saxe, A. M. & Sompolinsky, H. High-dimensional dynamics of generalization error in neural networks. *Neural Networks* **132,** 428–446 (2020).

365. Tian, Y. *An Analytical Formula of Population Gradient for Two-Layered ReLU Network and Its Applications in Convergence and Critical Point Analysis* in *Proceedings of the 34th International Conference on Machine Learning (ICML)* (2017), 3404–3413.

366. Du, S., Lee, J., Tian, Y., Singh, A. & Poczos, B. *Gradient Descent Learns One-hidden-layer CNN: Don't be Afraid of Spurious Local Minima* in *Proceedings of the 35th International Conference on Machine Learning* **80** (2018), 1339–1348.

367. Aubin, B. *et al. The committee machine: Computational to statistical gaps in learning a two-layers neural network* in *Advances in Neural Information Processing Systems 31* (2018), 3227–3238.

368. Saxe, A. *et al. On the information bottleneck theory of deep learning* in *ICLR* (2018).

369. Baity-Jesi, M. *et al. Comparing Dynamics: Deep Neural Networks versus Glassy Systems* in *Proceedings of the 35th International Conference on Machine Learning* (2018).

370. Goldt, S., Advani, M., Saxe, A., Krzakala, F. & Zdeborová, L. *Dynamics of stochastic gradient descent for two-layer neural networks in the teacher-student setup* in *Advances in Neural Information Processing Systems 32* (2019).

371. Ghorbani, B., Mei, S., Misiakiewicz, T. & Montanari, A. *Limitations of Lazy Training of Two-layers Neural Network* in *Advances in Neural Information Processing Systems 32* (2019), 9111–9121.

372. Yoshida, Y. & Okada, M. *Data-Dependence of Plateau Phenomenon in Learning with Neural Network — Statistical Mechanical Analysis* in *Advances in Neural Information Processing Systems 32* (2019), 1720–1728.

373. Bahri, Y. *et al.* Statistical Mechanics of Deep Learning. *Annual Review of Condensed Matter Physics* **11,** 501–528 (2020).

374. Gabrié, M. Mean-field inference methods for neural networks. *Journal of Physics A: Mathematical and Theoretical* **53,** 223002 (2020).

375. Kinzel, W. & Ruján, P. Improving a Network Generalization Ability by Selecting Examples. *EPL (Europhysics Letters)* **13,** 473–477 (1990).

376. Saad, D. *On-line learning in neural networks* (Cambridge University Press, 2009).

377. Brutzkus, A. & Globerson, A. *Globally Optimal Gradient Descent for a ConvNet with Gaussian Inputs* in *Proceedings of the 34th International Conference on Machine Learning - Volume 70* (2017), 605–614.

378. Mei, S., Montanari, A. & Nguyen, P. A mean field view of the landscape of two-layer neural networks. *Proceedings of the National Academy of Sciences* **115,** E7665–E7671 (2018).

379. Rotskoff, G. & Vanden-Eijnden, E. *Parameters as interacting particles: long time convergence and asymptotic error scaling of neural networks* in *Advances in Neural Information Processing Systems 31* (2018), 7146–7155.

380. Chizat, L. & Bach, F. *On the Global Convergence of Gradient Descent for Over-parameterized Models using Optimal Transport* in *Advances in Neural Information Processing Systems 31* (2018), 3040–3050.

381. Sirignano, J. & Spiliopoulos, K. Mean field analysis of neural networks: A central limit theorem. *Stochastic Processes and their Applications* (2019).

382. Liao, Q., Leibo, J. Z. & Poggio, T. *How important is weight symmetry in backpropagation?* in *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence* (2016), 1837–1844.

383. Baldi, P. & Hornik, K. Neural networks and principal component analysis: Learning from examples without local minima. *Neural networks* **2,** 53–58 (1989).

384. Saxe, A., McClelland, J. & Ganguli, S. *Exact solutions to the nonlinear dynamics of learning in deep linear neural networks* in *International Conference on Learning Representations (ICLR)* (2014).

385. Ji, Z. & Telgarsky, M. *Gradient descent aligns the layers of deep linear networks* in *International Conference on Learning Representations (ICLR)* (2019).

386. Crafton, B., Parihar, A., Gebhardt, E. & Raychowdhury, A. Direct feedback alignment with sparse connections for local learning. *Frontiers in neuroscience* **13,** 525 (2019).

387. Saxton, D., Grefenstette, E., Hill, F. & Kohli, P. Analysing mathematical reasoning abilities of neural models. *arXiv preprint arXiv:1904.01557* (2019).

388. Cobbe, K. *et al.* Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168* (2021).

389. Davies, A. *et al.* Advancing mathematics by guiding human intuition with AI. *Nature* (2021).

390. Lample, G. & Charton, F. Deep learning for symbolic mathematics. *arXiv preprint arXiv:1912.01412* (2019).

391. Allamanis, M., Chanthirasegaran, P., Kohli, P. & Sutton, C. *Learning continuous semantic representations of symbolic expressions* in *International Conference on Machine Learning* (2017), 80–88.

392. Arabshahi, F., Singh, S. & Anandkumar, A. *Towards solving differential equations through neural programming* in *ICML Workshop on Neural Abstract Machines and Program Induction (NAMPI)* (2018).

393. Charton, F., Hayat, A. & Lample, G. Learning advanced mathematical computations from examples. *arXiv preprint arXiv:2006.06462* (2020).

394. Kaiser, Ł. & Sutskever, I. Neural gpus learn algorithms. *arXiv preprint arXiv:1511.08228* (2015).

395. Trask, A. *et al.* Neural arithmetic logic units. *arXiv preprint arXiv:1808.00508* (2018).

396. Charton, F. Linear algebra with transformers. *arXiv preprint arXiv:2112.01898* (2021).

397. Augusto, D. A. & Barbosa, H. J. *Symbolic regression via genetic programming* in *Proceedings. Vol. 1. Sixth Brazilian Symposium on Neural Networks* (2000), 173–178.

398. Schmidt, M. & Lipson, H. Distilling free-form natural laws from experimental data. *science* **324,** 81–85 (2009).

399. Murari, A. *et al.* Symbolic regression via genetic programming for data driven derivation of confinement scaling laws without any assumption on their mathematical form. *Plasma Physics and Controlled Fusion* **57,** 014008 (2014).

400. McKay, B., Willis, M. J. & Barton, G. W. *Using a tree structured genetic algorithm to perform symbolic regression* in *First international conference on genetic algorithms in engineering systems: innovations and applications* (1995), 487–492.

401. Sahoo, S., Lampert, C. & Martius, G. *Learning equations for extrapolation and control* in *International Conference on Machine Learning* (2018), 4442–4450.

402. Kim, S. *et al.* Integration of neural network-based symbolic regression in deep learning for scientific discovery. *IEEE Transactions on Neural Networks and Learning Systems* (2020).

403. Petersen, B. K. *et al.* Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients. *arXiv preprint arXiv:1912.04871* (2019).

404. Biggio, L., Bendinelli, T., Neitz, A., Lucchi, A. & Parascandolo, G. Neural Symbolic Regression that Scales. *arXiv preprint arXiv:2106.06427* (2021).

405. Valipour, M., You, B., Panju, M. & Ghodsi, A. SymbolicGPT: A Generative Transformer Model for Symbolic Regression. *arXiv preprint arXiv:2106.14131* (2021).

406. Sloane, N. J. in *Towards mechanized mathematical assistants* 130–130 (Springer, 2007).

407. Wu, C. W. Can machine learning identify interesting mathematics? An exploration using empirically observed laws. *arXiv preprint arXiv:1805.07431* (2018).

408. Ryskina, M. & Knight, K. Learning Mathematical Properties of Integers. *arXiv preprint arXiv:2109.07230* (2021).

409. Ragni, M. & Klein, A. *Predicting Numbers: An AI Approach to Solving Number Series* in *KI 2011: Advances in Artificial Intelligence* (eds Bach, J. & Edelkamp, S.) (Springer Berlin Heidelberg, 2011), 255–259.

410. Nam, H., Kim, S. & Jung, K. *Number Sequence Prediction Problems for Evaluating Computational Powers of Neural Networks* 2018. arXiv: 1805.07494 [cs.NE].

411. Van der Maaten, L. & Hinton, G. Visualizing Data using t-SNE. *Journal of Machine Learning Research* **9,** 2579–2605 (2008).

412. Brückner, D. B., Ronceray, P. & Broedersz, C. P. Inferring the dynamics of underdamped stochastic systems. *Physical review letters* **125,** 058103 (2020).

413. Udrescu, S.-M. & Tegmark, M. *AI Feynman: a Physics-Inspired Method for Symbolic Regression* 2020. arXiv: 1905.11481 [physics.comp-ph].

414. Cranmer, M. *et al.* Discovering Symbolic Models from Deep Learning with Inductive Biases. *ArXiv* **abs/2006.11287** (2020).

415. Garnelo, M., Arulkumaran, K. & Shanahan, M. Towards deep symbolic reinforcement learning. *arXiv preprint arXiv:1609.05518* (2016).

416. Landajuela, M. *et al. Discovering symbolic policies with deep reinforcement learning* in *International Conference on Machine Learning* (2021), 5979–5989.

417. La Cava, W. *et al.* Contemporary symbolic regression methods and their relative performance. *arXiv preprint arXiv:2107.14351* (2021).

418. Aréchiga, N. *et al.* Accelerating Understanding of Scientific Experiments with End to End Symbolic Regression. *ArXiv* **abs/2112.04023** (2021).

419. Udrescu, S.-M. & Tegmark, M. Symbolic Pregression: Discovering Physical Laws from Raw Distorted Video. *Physical review. E* **103 4-1,** 043307 (2021).

420. Butter, A., Plehn, T., Soybelman, N. & Brehmer, J. *Back to the Formula – LHC Edition* in (2021).

421. Schmidt, M. & Lipson, H. in *Genetic programming theory and practice VIII* 129–146 (Springer, 2011).

422. La Cava, W., Singh, T. R., Taggart, J., Suri, S. & Moore, J. H. Learning concise representations for regression by evolving networks of trees. *arXiv preprint arXiv:1807.00981* (2018).

423. McConaghy, T. in *Genetic Programming Theory and Practice IX* 235–260 (Springer, 2011).

424. Virgolin, M., Alderliesten, T., Witteveen, C. & Bosman, P. A. Improving model-based genetic programming for symbolic regression of small expressions. *Evolutionary computation* **29,** 211–237 (2021).

425. De França, F. O. & Aldeia, G. S. I. Interaction–Transformation Evolutionary Algorithm for Symbolic Regression. *Evolutionary computation* **29,** 367–390 (2021).

426. Arnaldo, I., Krawiec, K. & O'Reilly, U.-M. *Multiple regression genetic programming* in *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation* (2014), 879–886.

427. Virgolin, M., Alderliesten, T. & Bosman, P. A. N. *Linear Scaling with and within Semantic Backpropagation-Based Genetic Programming for Symbolic Regression* in *Proceedings of the Genetic and Evolutionary Computation Conference* (Association for Computing Machinery, Prague, Czech Republic, 2019), 1084–1092. ISBN: 9781450361118. https://doi.org/10.1145/3321707.3321758.

428. Kommenda, M., Burlacu, B., Kronberger, G. & Affenzeller, M. Parameter identification for symbolic regression using nonlinear least squares. *Genetic Programming and Evolvable Machines* **21,** 471–501. https://doi.org/10.1007/s10710-019-09371-3 (2020).

429. Martius, G. & Lampert, C. H. Extrapolation and learning equations. *arXiv preprint arXiv:1610.02995* (2016).

430. Hahn, C., Schmitt, F., Kreber, J. U., Rabe, M. N. & Finkbeiner, B. Teaching temporal logics to neural networks. *arXiv preprint arXiv:2003.04218* (2020).

431. Polu, S. & Sutskever, I. Generative language modeling for automated theorem proving. *arXiv preprint arXiv:2009.03393* (2020).

432. Guimerà, R. *et al.* A Bayesian machine scientist to aid in the solution of challenging scientific problems. *Science advances* **6,** eaav6971 (2020).

433. Kidger, P. *SympyTorch* https://github.com/patrick-kidger/sympytorch. 2021.

434. Horace He, R. Z. *functorch: JAX-like composable function transforms for PyTorch* https://github.com/pytorch/functorch. 2021.

435. Wiseman, S. & Rush, A. M. *Sequence-to-Sequence Learning as Beam-Search Optimization* 2016. https://arxiv.org/abs/1606.02960.

436. d'Ascoli, S., Kamienny, P.-A., Lample, G. & Charton, F. Deep Symbolic Regression for Recurrent Sequences. *arXiv preprint arXiv:2201.04600* (2022).

437. Friedman, J. H. Greedy function approximation: a gradient boosting machine. *Annals of statistics,* 1189–1232 (2001).

438. Strogatz, S. H. *Nonlinear Dynamics and Chaos: With Applications to Physics, Biology, Chemistry and Engineering* (Westview Press, 2000).

439. Zdeborová, L. Understanding deep learning is also a job for physicists. *Nature Physics* **16,** 602–604 (2020).

440. Chung, S., Lee, D. D. & Sompolinsky, H. Classification and geometry of general perceptual manifolds. *Physical Review X* **8,** 031003 (2018).

441. Pastore, M., Rotondo, P., Erba, V. & Gherardi, M. Statistical learning theory of structured data. *Physical Review E* **102,** 032119 (2020).

442. Seddik, M. E. A., Louart, C., Tamaazousti, M. & Couillet, R. *Random matrix theory proves that deep learning representations of gan-data behave as gaussian mixtures* in *International Conference on Machine Learning* (2020), 8573–8582.

443. Gur-Ari, G., Roberts, D. A. & Dyer, E. Gradient descent happens in a tiny subspace. *arXiv preprint arXiv:1812.04754* (2018).

444. Gurbuzbalaban, M., Simsekli, U. & Zhu, L. *The heavy-tail phenomenon in SGD* in *International Conference on Machine Learning* (2021), 3964–3975.

445. Gunasekar, S., Woodworth, B., Bhojanapalli, S., Neyshabur, B. & Srebro, N. *Implicit Regularization in Matrix Factorization* in *Advances in Neural Information Processing Systems 30* (2017), 6151–6159.

446. He, F., Liu, T. & Tao, D. Control batch size and learning rate to generalize well: Theoretical and empirical evidence. *Advances in Neural Information Processing Systems* **32** (2019).

447. Mei, S., Montanari, A. & Nguyen, P.-M. A mean field view of the landscape of two-layer neural networks. *Proceedings of the National Academy of Sciences* **115,** E7665–E7671 (2018).

448. Saxe, A. M., McClelland, J. L. & Ganguli, S. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *International Conference on Learning Representations* (2014).

449. Li, Q. & Sompolinsky, H. Statistical mechanics of deep linear neural networks: The backpropagating kernel renormalization. *Physical Review X* **11,** 031059 (2021).

450. Wiatowski, T. & Bölcskei, H. A mathematical theory of deep convolutional neural networks for feature extraction. *IEEE Transactions on Information Theory* **64,** 1845–1866 (2017).

451. Nguyen, H.-T., Li, S. & Cheah, C. C. A Layer-Wise Theoretical Framework for Deep Learning of Convolutional Neural Networks. *IEEE Access* **10,** 14270–14287 (2022).

452. Liu, H., Dai, Z., So, D. & Le, Q. V. Pay attention to mlps. *Advances in Neural Information Processing Systems* **34,** 9204–9215 (2021).

453. Jacot, A., Gabriel, F. & Hongler, C. *Neural Tangent Kernel: Convergence and Generalization in Neural Networks* in *Advances in Neural Information Processing Systems 31* (eds Bengio, S. *et al.*) (Curran Associates, Inc., 2018), 8571–8580.

454. Yang, G. *et al.* Tuning Large Neural Networks via Zero-Shot Hyperparameter Transfer. *Advances in Neural Information Processing Systems* **34** (2021).

455. Jumper, J. *et al.* Highly accurate protein structure prediction with AlphaFold. *Nature* **596,** 583–589 (2021).

456. Degrave, J. *et al.* Magnetic control of tokamak plasmas through deep reinforcement learning. *Nature* **602,** 414–419 (2022).

457. Schoenholz, S. S., Gilmer, J., Ganguli, S. & Sohl-Dickstein, J. Deep information propagation. *arXiv preprint arXiv:1611.01232* (2016).

458. Saxe, A. M., McClelland, J. L. & Ganguli, S. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *International Conference on Learning Representations* (2014).

459. Kingma, D. P. & Ba, J. Adam: A method for stochastic optimization. *International Conference on Learning Representations* (2015).

460. Brito, C., Ikeda, H., Urbani, P., Wyart, M. & Zamponi, F. Universality of jamming of non-spherical particles. *arXiv preprint arXiv:1807.01975* (2018).

461. Geiger, M. *et al.* Jamming transition as a paradigm to understand the loss landscape of deep neural networks. *Physical Review E* **100,** 012115 (2019).

462. Radosavovic, I., Kosaraju, R. P., Girshick, R., He, K. & Dollár, P. *Designing network design spaces* in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2020), 10428–10436.

463. Zhao, S. *et al.* SplitNet: Divide and Co-training. *arXiv preprint arXiv:2011.14660* (2020).

464. He, T. *et al. Bag of tricks for image classification with convolutional neural networks* in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2019), 558–567.

465. Meurer, A. *et al.* SymPy: symbolic computing in Python. *PeerJ Computer Science* **3,** e103 (2017).

466. Paccanaro, A. & Hinton, G. E. Learning distributed representations of concepts using linear relational embedding. *IEEE Transactions on Knowledge and Data Engineering* **13,** 232–244 (2001).

467. Jin, Y., Fu, W., Kang, J., Guo, J. & Guo, J. *Bayesian Symbolic Regression* 2020. arXiv: 1910.08892 [stat.ME].

468. Mundhenk, T. N. *et al. Symbolic Regression via Neural-Guided Genetic Programming Population Seeding* 2021. arXiv: 2111.00053 [cs.NE].

RÉSUMÉ

L'intelligence artificielle, devenue l'un des enjeux technologiques majeurs de notre siècle, est actuellement dominée par les réseaux de neurones artificiels. Si ces derniers ont déjà permis des avancées majeures, leur fonctionnement reste mal compris sur le plan théorique.
L'objectif principal de cette thèse est de réduire l'écart entre théorie en pratique, en s'aidant d'outils développés dans la littérature de physique statistique. Nous nous intéresserons à trois questions centrales: (i) l'effet de la sur-paramétrisation sur la généralisation, (ii) les biais inductifs découlant des choix architecturaux et (iii) la dynamique de l'apprentissage.
En guise d'ouverture, nous présentons une nouvelle application du deep learning dans le domaine de la régression symbolique, qui consiste à prédire l'expression mathematique d'une fonction à partir de ses valeurs.

MOTS CLÉS

Réseaux de neurones, physique statistique, méthode des répliques, sur-paramétrisation, biais inductifs, convolution, attention, dynamique, paysages énergétiques, régression symbolique.

ABSTRACT

Deep learning has become the cornerstone of artificial intelligence, and has fueled breakthroughs in a number of fields. Yet, the key reasons underpinning the success of deep neural networks remain to be clarified.
The main objective of this thesis is to bridge the gap between theory and practice, armed with the toolbox of statistical physics. We focus on three central questions: (i) the benefit of over-parametrization on generalization, (ii) the inductive bias which stems from their architectural choices and (iii) the dynamics of learning.
As an opening, we introduce a novel application of deep learning to the task of symbolic regression, i.e. predicting the expression of a mathematical function from its values.

KEYWORDS

Neural networks, statistical physics, replica method, overparametrization, inductive biases, convolutional networks, attention networks, dynamics, loss landscapes, symbolic regression.