Recurrent Neural Networks models : inference, representations, and the role of regularization

Arnaud Fanthomme

PSL UNIVERSITÉ PARIS

THÈSE DE DOCTORAT
DE L'UNIVERSITÉ PSL

Préparée à Ecole Normale Supérieure

Recurrent Neural Networks models: inference, representations, and the role of regularization

Soutenue par
Arnaud Fanthomme
Le 08 12 2021

École doctorale n°564
EDPIF

Spécialité
Physique

Composition du jury :

Sara Solla Northwestern University	*Présidente du Jury*
David Saad Aston University	*Rapporteur*
Omri Barak Israel Institute of Technology	*Examinateur*
Surya Ganguli Stanford University	*Examinateur*
Andrew Saxe Oxford University	*Examinateur*
Rémi Monasson LPENS	*Directeur de thèse*

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

Contents

List of Figures

Résumé en français de la thèse et remerciements

Les travaux présentés dans le cadre de cette thèse se situent à l'interface entre trois grands domaines des sciences : la physique, les neurosciences et l'informatique. Si les objets étudiés sont, à l'exception des systèmes de spins en interaction du Chapitre 2, assez éloignés de ceux habituellement considérés en Physique, même théorique, la démarche fondamentale reste identique : observer les phénomènes, les décrire à l'aide de concepts mathématiques et utiliser les résultats obtenus sur ces modèles pour mieux comprendre la physique des systèmes étudiés.

Au Chapitre 1, nous présentons quelques rappels d'inférence statistique. Après des rappels succincts sur les différentes distributions "usuelles" de statistiques, nous introduisons le concept d'inférence Bayésienne et de "Maximum a Posteriori". Nous introduisons ensuite les modèles graphiques probabilistes, outil versatile pour la représentation des dépendances conditionnelles entre variables aléatoires et indispensable à l'élaboration d'algorithmes d'inférence efficaces par l'introduction d'hypothèses simplificatrices. Nous introduisons ensuite plusieurs familles de distributions, leurs descriptions en tant que modèles graphiques, les procédures d'inférence associées et leurs applications pratiques en modélisation statistique : Vecteurs Gaussiens, modèles de Potts, et machines de Boltzmann restreintes, particulièrement utiles pour la compréhension du Chapitre 2 ; les Chaines de Markov et de Markov Cachées (ainsi que l'algorithme Monte-Carlo), à la base de la théorie de l'Apprentissage par Renforcement présentée au Chapitre 6.

Dans le Chapitre 2, qui présente des résultats de recherche originaux, nous considérons un cas particulier de régularisation dans une procédure d'inférence. La question du rôle de la régularisation est centrale en Apprentissage Machine : empiriquement, l'utilisation judicieuse de régularisation permet d'atteindre de bien meilleurs résultats que ceux obtenus sans contrainte ; théoriquement, l'addition de termes de régularisation dans la "vraisemblance" utilisée pour l'inférence correspond à un biais statistique, dont on s'attend à ce qu'il détériore le modèle obtenu finalement et non à ce qu'il l'améliore. Ce résultat contre-intuitif se comprend mieux lorsque l'on considère séparément la qualité de l'inférence sur les données d'entraînement, qui diminue effectivement dès lors qu'une régularisation est appliquée, et celle sur les données de test, qui présente une évolution non monotone : une régularisation optimale existe. Ce phénomène est souvent qualifié d'équilibre biais-variance : la régularisation empêche le modèle de s'adapter parfaitement aux données d'entraînement, limitant ainsi le phénomène de sur-apprentissage et permettant au modèle de mieux généraliser à de nouveaux exemples. Une question importante en biologie est de comprendre et prédire, la valeur de cette régularisation optimale. Dans le cadre d'un modèle simpliste de spins en interaction, et d'une régularisation par la norme L_2 de la matrice d'interaction inférée, nous avons montré que la régularisation optimale correspond à un croisement entre la vraisemblance des données générées par le modèle inféré et celle des données de test ; forts de ce constat, une estimation de la régularisation optimale dans le régime où le nombre de données excède fortement la dimension des données en question, valide indépendamment de la structure des interactions utilisées pour la génération des données. Nous montrons également empiriquement qu'une régularisation optimale existe dans le cadre d'une pénalité L_1 sur la norme de la matrice d'interaction, la prédiction de sa valeur restant une question ouverte à ce jour.

Afin de mieux situer les travaux présentés par la suite dans leur contexte, ce manuscrit contient au Chapitre 3 une introduction aux neurosciences. Nous commençons par un bref historique de l'étude des neurones individuels, expliquant comment l'évolution du potentiel membranaire constitue le support principal de la représentation d'information dans le cerveau. Nous nous focalisons ensuite sur l'échelle mésoscopique des neurosciences, celle où l'interaction entre de nombreux neurones individuels permet l'émergence de phénomènes collectifs, analogues à ceux que l'on rencontre en Physique Statistique, dont l'une des caractéristiques notables de neurones étant effectivement observé au cours du comportement, un concept central pour la théorie des réseaux de neurones récurrents que nous développons par la suite. Enfin, nous présentons quelques exemples de circuits neuronaux importants non seulement en biologie comportementale, mais aussi en sciences cognitives et par la suite en Apprentissage Machine : le système visuel, le système auditif, la navigation spatiale et le raisonnement abstrait.

Les Chapitres 4 à 6 présentent quant à eux les domaine de l'Apprentissage par Réseaux de Neurones Profonds, en partant d'une perspective proche de celle des Neurosciences. Au Chapitre 4, nous introduisons les neurones de McCulloch-Pitts, exemple historique des premières tentatives de modélisation du traitement d'information par le cerveau, ainsi que la règle d'apprentissage Perceptron permettant de régler les connections entre neurones pour réaliser une tâche de séparation linéaire. Nous généralisons ensuite ce concept afin d'introduire le Perceptron Multi-Couches, prototype des systèmes d'Apprentissage Profond utilisés aujourd'hui, et détaillons le formalisme mathématique associé en introduisant les notions de jeu de données, de modèle différentiable et d'apprentissage par descente de gradient. Au Chapitre 5, nous étendons ce concept à d'autres architectures, notamment les réseaux convolutionnels et récurrents, et établissons le lien entre ces derniers et les modèles d'Ising étudiés en Physique Statistique. Au Chapitre 6, nous présentons le cadre historique et les avancées récentes de l'Apprentissage par Renforcement. Ce paradigme complète les algorithmes d'apprentissage supervisés, nécessitant l'accès à un jeu de données, en autorisant l'apprentissage à partir de "tentatives et échecs". De telles méthodes sont particulièrement utiles dans le cas où un réseau cherche à établir une "politique" reliant, par exemple, l'image affichée par l'écran d'une console de jeu Atari à la commande que le joueur doit fournir à sa manette pour gagner : il est souvent impossible d'écrire de telles politiques "à la main", et il est alors intéressant d'utiliser le Renforcement pour trouver de telles politiques. Un autre avantage majeur de ces méthodes est qu'elles permettent de comparer les solutions "intuitives" pour un humain ou observées biologiquement, à celles trouvées par l'algorithme.

Le Chapitre 7 présente les résultats de nos travaux, publiés dans le journal **Neural Computation** sous le titre "Low-dimensional Manifolds support Multiplexed Integrations in Recurrent Neural Networks", dans lesquels nous étudions le problème de calculer la somme des termes de D (d'ordre 1) séries temporelles en utilisant ces séries comme entrées d'un réseau de neurones récurrent. Nous considérons dans un premier temps le cas d'un réseau récurrent linéaire, pour lequel des résultats analytiques peuvent être obtenus ; nous exhibons des conditions nécessaires et suffisantes sur la matrice de connectivité du réseau qui garantissent que le système correspondant est capable d'intégrer des séries temporelles de durée arbitraire, montrons que de telles solutions sont obtenues de façon garantie dès lors que la durée des séquences d'entraînement est supérieure à la taille du réseau, et qu'en pratique elles sont aussi obtenues dès que les séquences d'entraînement sont de longueur supérieure à 3 ; enfin, nous analysons en détail la dynamique de convergence de la Descente de Gradient vers ces minima de la fonction de coût. Nous considérons dans un second temps le cas de réseaux récurrents non-linéaires, pour lesquels il n'a pas été possible

d'obtenir de résultats analytiques. Nous adoptons donc une démarche phénoménologique d'observation des solutions obtenues par Descente de Gradient, et observons que les matrices de connectivité de ces solutions ont un spectre de valeurs singulières correspondant à une "masse" proche de 0 et dont la densité décroit exponentiellement, accompagnée de D valeurs singulières grandes devant les autres. Nous notons également que l'état interne du réseau vit dans un voisinage proche d'une variété de dimension D, et que la position dans cette variété est en bijection avec la valeur des intégrales. De plus, cette variété s'interprète également comme l'image par la non-linéarité du réseau d'un hyperplan de dimension D et la position dans cet hyperplan est une fonction linéaire des intégrales. Nous qualifions ce codage de "courant-linéaire", et proposons une fonction de coût, définie indépendamment de données, qui lorsque optimisée génère des intégrateurs parfaits avec une non-linéarité arbitraire et satisfaisant le codage "courant-linéaire".

Le Chapitre 8 présente lui aussi des résultats originaux, soumis pour publication à l'International Conference on Learning and Representations. Nous y présentons une nouvelle approche au problème de "fusion de capteurs", dans notre cas un capteur simulant une image et l'autre simulant un signal proprioceptif de vitesse, en utilisant ces deux signaux comme entrées d'un réseau de neurone récurrent entraîné à déterminer son déplacement total au cours d'une série de mouvements. Ce problème, appelé en anglais "Path Integration", a été étudié en détail depuis de nombreuses années par les neurobiologistes, car cette tâche est supposée être à l'origine du développement de "cartes mentales", des représentations internes qu'un agent (par exemple un rat) se fait de son environnement, et plus particulièrement de sa structure spatiale. Nous montrons qu'une approche particulièrement efficace pour obtenir des réseaux récurrents capables de créer des cartes mentales est d'entraîner ce réseau à fournir, simultanément, un modèle de la dynamique directe de son environnement (le prochain état visuel, état donné l'état visuel actuel et une action) et de sa dynamique inverse (quelle action sépare deux états visuels de l'environnement) ; ces deux modèles sont intégrés dans une structure plus large, capable d'effectuer une "remise à zéro" de son état interne (obtenu en itérant le modèle direct le long de la trajectoire) en utilisant le signal visuel de l'environnement. Cette structure permet de générer des représentations non ambiguës d'espaces partiellement observables, qui peuvent ensuite servir de base pour des algorithmes de Renforcement (permettant par exemple de faire de la navigation spatiale vers un objectif donné) dans un environnement où les capteurs sont peu fiables.

Enfin, le Chapitre 9 présente une conclusion générale reliant nos différents travaux originaux. Nous insistons notamment sur le fait que les réseaux de neurones que nous avons étudié fournissent des représentations de basse dimension des données qui leur sont fournies en entrée. La dimension de ces représentations est, comme attendu, égale au nombre de variables indépendantes nécessaire pour quantifier exactement l'état du système représenté (par exemple, une image d'un environnement planaire est représentée comme un vecteur de dimension 2, représentant les coordonnées x et y dans ce plan). Nous notons également que de telles représentations n'ont d'intérêt que conjointement avec un certain nombre d'opérateurs agissant dessus (tels que ceux représentant l'évolution des intégrales à l'arrivée d'une nouvelle entrée, ou de la position lors d'un mouvement), et que représenter correctement l'action des opérateurs constitue une étape importante dans l'élaboration de fonctions de coût permettant d'apprendre de nouvelles tâches.

Remerciements

Tout d'abord, merci à mes parents qui m'ont toujours soutenu, même si je n'ai pas forcément rendu ça facile. Merci à Rémi pour tout ce qu'il m'a appris ces trois dernières années, cette thèse n'aurait jamais pu aboutir à un tel résultat sans ses conseils et suggestions. Merci aux membres du Jury, Sara, David, Andrew, Omri et Surya qui ont pris le temps de s'intéresser à mes travaux, et ont aidé à rendre ma soutenance encore plus mémorable. Merci à Hugo, Thibault et Vincent pour tous les déjeuners, dîners et autres moments qu'on a partagé. Merci à Damien et Hugo pour tous les week-ends à Montgé, à Gigaro, et pour nous avoir réunis avec tant de personnes formidables qu'on ne voit pas assez sans vous. Merci à Victor et Clément, qui ont partagé mon quotidien au labo pendant toute cette thèse. Merci à Victor, Marine, Wyatt et Marlowe, qui me rappellent qu'il y a autre chose dans la vie que la recherche. Merci à mes colocs, Alexandre et Roxane, avec qui nous avons partagé une année de télétravail, et toutes les joies qui vont avec. Merci à tous les membres du labo, Sébastien (merci pour le petit concert entre thésards, des comme ça on en fait pas tous les jours!), Simona, Aldo, Marco, Cyril, Eugenio, Andrea, Max, Jorge, Tobias, Mariia, Lorenzo, Jérôme, le groupe n'aurait pas été le même sans vous. Merci à Nataniel et Giulio pour Les Houches, les meilleures vacances "studieuses" que j'ai passé depuis bien longtemps, et une occasion unique de créer un esprit de labo. Merci à tous mes cobureaux, Tristan, Clément, Cathelijne, Augustin, Andrea, Francesco, Meriem, pour les bons moments qu'on a partagés pour faire retomber la pression de la recherche. Merci à tous les personnels supports de l'ENS, sans qui la recherche ne serait tout simplement pas possible. Merci aux restaurants de la rue mouffetard, pour m'avoir si souvent rempli de joie.

Merci à toutes celles et ceux qui ont cru en moi.

Outline of the contents

Chapter 1 presents some useful reminders about Statistical Inference.

Chapter 2 presents original research we led on regularization in Gaussian model inference, from our upcoming paper:

- Fanthomme, A., Rizzato, F., Cocco, S. and Monasson, R. (2021). Optimal regularizations for data generation with probabilistic graphical models, *Arxiv preprint 2112.01292*.

Chapter 3 to 6 are reminders on the theory of Deep Learning, and its relationship to computational neuroscience and physics.

Chapter 7 is a reproduction of our Neural Computation article on low-dimensional manifolds in Recurrent Neural Networks integrators:

- Fanthomme, A. and Monasson, R. (2021). Low-Dimensional Manifolds Support Multiplexed Integrations in Recurrent Neural Networks, *Neural Computation* 33 (4): 1063–1112.

Chapter 8 is a reproduction of our submission to the 2022 International Conference on Learning Representations:

- Fanthomme, A. and Monasson, R. (2021). Stable cognitive maps for Path Integration emerge from fusing visual and proprioceptive sensors. *Submission to ICLR 2022.*

Chapter 9 provides a unified perspective on the different original results we presented in the rest of the manuscript.

Appendices present additional details on Chapter 7 and Chapter 8.

Chapter 1

Statistical Inference

Abstract

In this first introductory Chapter, we present some reminders on Statistical Inference. After a brief summary of usual random variable distributions, we introduce the concepts of Bayesian and "Maximum A Posteriori" inference. We then present "Probabilistic Graphical Models" as a versatile tool to represent conditional dependencies between random variable, indispensable to the formulation of efficient inference algorithms by allowing the introduction of simplifying hypothesis (most notably of conditional independence). We consider several families of distributions, together with their formulation as Graphical Models, inference procedure, and practical applications in statistical modeling: Gaussian Vectors, Potts models and Restricted Boltzmann Machines, which will be particularly relevant for Chapter 1; Markov Chains and Hidden Markov Models, together with the Monte Carlo algorithm, which form the foundation of the Reinforcement Learning theory presented in Chapter 6.

1.1 Usual distributions

First, let us introduce the concept of a **random variable** X as an object that can take values (referred to as samples or observations) in an ensemble called the **universe** Ω, which can be either discrete (*e.g.* the values on the faces of a die, the words in the dictionary, the set of amino acids in a protein) or continuous (the resting position of the dice on the table, the height of an individual, the electric potential on an electrode).

In both cases, we define the **probability distribution** p of the random variable X as a function from Ω to $\mathbb{R}$ that is normalized: in the case of a discrete variable, this is a discrete set of **probabilities**, one for each event, which sums to 1; in the case of a continuous variable, it is a function over the universe called the **probability density**, whose integral is equal to 1. In the following, by abuse of notation, we use the term "probability" for both the probability of a discrete event, or the value of the probability density function at a certain point in the universe.

In practice, it is often useful to consider **parametrized families** of distributions, so that to each value of the parameters θ is associated a probability distribution p_θ. Examples of such families of distributions are:

- Bernoulli distributions, which take values $b \in \Omega = \{0, 1\}$, and are parametrized by the probability $\theta \in [0, 1]$ of drawing a 1:

$$p_\theta(b) = \theta^b \, (1 - \theta)^{1-b}. \tag{1.1}$$

- Binomial distributions, corresponding to the sum of n samples from a Bernoulli distribution, can take values $k \in \Omega = \{0, 1, \ldots, n\}$ and are parametrized by the same parameter $\theta \in [0, 1]$ as the underlying Bernoulli distribution:

$$p_\theta(k) = \binom{n}{k} \theta^k \, (1 - \theta)^{n-k}. \tag{1.2}$$

- Multinomial distributions, which extend the concept of a binomial distribution to the case of n draws from a distribution which can take K discrete values with probabilities $\pi_1, \ldots, \pi_K$. The probability distribution is then:

$$p_{\boldsymbol{\pi}}(x) = \frac{n!}{\prod_{k=1}^{K} x_k} \prod_{k=1}^{K} \pi_k^{x_k}, \tag{1.3}$$

  where, for all $k \in 0, \ldots, k$, x_k is the number of times output k has been obtained.

- Gaussian distributions, parametrized by their means μ and standard deviation σ, which take value in $\mathbb{R}$ and such that the probability distribution can be written:

$$p_{\mu,\sigma}(x) = \frac{1}{\sigma\sqrt{2\pi}} \, e^{-\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2}}. \tag{1.4}$$

  These distributions are extremely relevant in Statistical Physics because of a property known as the central limit theorem: the distribution of the sum of a large number of random variables converges towards a Gaussian.

1.2 Bayesian statistics, Maximum A Posteriori inference

In the following, we will be interested in the concept of Statistical Inference, *i.e.* the problem of determining the "best" parameter θ given a set of observations from the distribution. To do so, we define the **Likelihood** $\mathcal{L}$[1] of parameter θ given observations $\boldsymbol{x}$ as:

$$\begin{aligned} \mathcal{L}: \Theta &\to \mathbb{R}^+ \\ \theta &\mapsto p(\boldsymbol{x}|\theta) \end{aligned} , \tag{1.5}$$

where $p(\boldsymbol{x}|\theta)$ is the probability of observations x **conditioned** on the parameters θ.

Now adopting the Bayesian point of view that considers θ itself as a random variable, one can define a **prior** distribution $p(\theta)$, and make use of Bayes rule to compute the **posterior** distribution $p(\theta|\boldsymbol{x})$ as:

$$p(\theta|\boldsymbol{x}) = \frac{p(\boldsymbol{x}|\theta)\, p(\theta)}{p(x)}. \tag{1.6}$$

The maximum of this function with respect to θ is called the Maximum A Posteriori (MAP) estimation θ^{MAP} of the parameter θ. In the special case where the prior $p(\theta)$ is uniform, *i.e.* does not depend on θ, the MAP estimator coïncides with the Maximum Likelihood Estimator θ^{MLE} which maximizes $p(\boldsymbol{x}|\theta)$.

1.3 Conditional probabilities and Graphical Models

In this dissertation, we will often consider the problem of performing inference on the joint distribution of numerous variables, for example the activity of all neurons in a certain area of the brain, the sequence of amino acids in a given protein, or all phonemes in a spoken sentence. Since those objects live in very high-dimensional spaces, estimating their distributions without any prior information is practically impossible, and hypotheses must be made that will simplify the likelihood.

The most relevant type of hypothesis is one of **conditional independence**: given three random variables, A, B, and C, we say that A is conditionally independent of C given B if:

$$p(a|b,c) = p(a|b). \tag{1.7}$$

This concept extends the one of **independence** of variables $p(a,b) = p(a)\, p(b)$, and can be interpreted as meaning that the influence of the random variable C on the random variable A is due only to the influence of C on B.

From there, one can derive the concept of Graphical Model Wainwright and Jordan (2008), which allows for an intuitive representation of the dependencies between random variables, and efficient inference algorithms such as Message-Passing to be defined. While we do not attempt a rigorous presentation of those concepts, we give in the following some examples of practical relevance.

1.4 Gaussian Vectors

One example of high-dimensional distribution in which no conditional independence exists *a priori* is the one of the Multivariate Gaussian, often referred to as Gaussian Vectors.

[1] Unfortunately, the standard notations for likelihoods and losses, which will be the topics of interest respectively in Chapter 2 and Chapters (7,8) are identical.

This model describes the joint distribution of n scalar variables $\boldsymbol{x} = (x_1, \ldots, x_n)$ with a probability density function:

$$p(\boldsymbol{x}) = \frac{1}{\sqrt{(2\pi)^n \det(\boldsymbol{\Sigma})}} e^{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\boldsymbol{x}-\boldsymbol{\mu})}, \tag{1.8}$$

where the vector $\boldsymbol{\mu}$ is equal to the average value of the random vectors, and the positive-definite symmetric matrix $\boldsymbol{\Sigma}$ is equal to their covariance:

$$\boldsymbol{\mu} = \mathbb{E}[\boldsymbol{x}]. \tag{1.9}$$

$$\boldsymbol{\Sigma} = \mathbb{E}\left[(\boldsymbol{x} - \boldsymbol{\mu})(\boldsymbol{x} - \boldsymbol{\mu})^T\right]. \tag{1.10}$$

Because of this, lines of equiprobability are ellipses, whose axes coincide with the eigenvectors of $\boldsymbol{\Sigma}$, see Figure 1.1. It is interesting to note the deep connection between the model of Gaussian Vectors and the dimensionality reduction (Maaten et al., 2009) method known as Principal Components Analysis, introduced by Pearson (1901). The objective of dimensionality reduction is to take data points living in $\mathbb{R}^n$, and map them to a low-dimensional manifold in such a way that "not too much information is lost during the mapping". While different meanings can be assigned to that statement, PCA considers the case where the mapping is a projection into a d–dimensional vector space, and the "information loss" is quantified by the sum of the square distances between each data-point and their projection. In that case, the optimal subspace is exactly the span of the largest d eigenvectors of the empirical covariance matrix, as these correspond to the directions in which data is most widely spread. This result is true for any random variable living in $\mathbb{R}^n$, not only Gaussian Vectors, but the intuition of the link between the covariance spectrum and the spread of data is more easily illustrated on these distributions.

It can be shown that the Maximum-Likelihood Estimator for the $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ parameters are, respectively, the empirical average and covariance, computed on the set of available samples $(\boldsymbol{s}^{(1)}, \ldots, \boldsymbol{s}^{(p)})$:

$$\boldsymbol{\mu}^{MLE} = \frac{1}{p} \sum_k \boldsymbol{s}^{(k)}. \tag{1.11}$$

$$\boldsymbol{\Sigma}^{MLE} = \frac{1}{p} \sum_k \left(\boldsymbol{s}^{(k)} - \boldsymbol{\mu}^{MLE}\right) \left(\boldsymbol{s}^{(k)} - \boldsymbol{\mu}^{MLE}\right)^T. \tag{1.12}$$

In Chapter 2, we show that this estimation, while unbiased, can lead to uncontrolled overfitting in cases where the number of samples is much smaller than the number of dimensions of the vector, in which case an optimal regularization exists that biases the estimate in a way that improves performance.

1.5 The Ising and Potts models

One of the main families of distributions in Statistical Physics, and in biological data analysis, is the one of Potts models (Potts, 1952), of which the extensively studied Ising model is a particular case. These distributions are defined on an ensemble of n "sites", which each take value in a discrete set $\{s_1, \ldots, s_q\}$. In protein analysis, each site is a position along the protein sequence, and the discrete values are the different "proteinogenic" amino acids that make up natural proteins (in which case, $q = 22$); in the Ising model, each site is a position in a lattice at which a spin–1/2 particle is present, and the $q = 2$ states correspond to the two possible orientations of the spin, up and down.

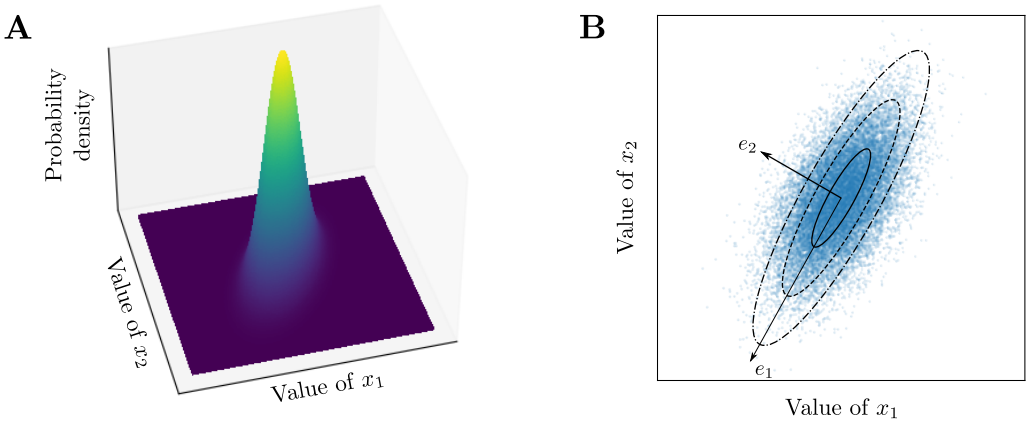


Figure 1.1: Schematic representation of the probability density function 1.8 of a 2–dimensional Multivariate Gaussian distribution (**A**) and random samples from that distribution (**B**). The eigenvectors of the covariance matrix e_1 and e_2 correspond to the orientation of the axes of the equiprobability ellipsoids, and the associated eigenvalues $\lambda_1 > \lambda_2$ correspond to the widths in those directions.

An interaction tensor $\boldsymbol{J}$ is then defined such that, for any two sites (i, j) and any two possible states (a, b), $\boldsymbol{J}[i, j, a, b]$ corresponds to the energy associated with having states a and b at sites i and j respectively (for $i = j$, the interaction matrix is diagonal and corresponds to local fields). The energy of configuration $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_n)$ is therefore obtained as:

$$H(\boldsymbol{\sigma}) = -\sum_{i=1}^{n}\sum_{j=1}^{n} \boldsymbol{J}[i, j, s_i, s_j], \tag{1.13}$$

which corresponds to summing over all pair of sites the interaction energy between the two corresponding states. It should be noted that in Physics, this tensor is usually assumed to depend only on whether the states a and b are identical or different (in the original Potts model, the state corresponded to three possible orientations of the spin along the (x, y, z) axes, and by symmetry interactions depended only on whether the spins were aligned); in protein analysis, this is not the case, as the interaction strength is allowed to depend on the exact amino-acids, and even to be asymmetric, see Cocco et al. (2018) for a review of those techniques.

Finally, the probability distribution over configurations is given by:

$$p(\boldsymbol{\sigma}) = \frac{1}{Z} e^{-H(\boldsymbol{\sigma})}, \tag{1.14}$$

where the *partition function* Z is used to ensure that this distribution is normalized and is defined as:

$$Z = \sum_{\boldsymbol{\sigma}} e^{-H(\boldsymbol{\sigma})}. \tag{1.15}$$

Similarly to how a Gaussian Vectors model is the maximum-entropy (least biased) model of a set of n scalars that reproduces one and two–points correlations (mean and covariance), Potts models are the least-constrained models of n discrete variables that

reproduce frequencies and correlations between each sites, defined as:

$$f_i(a) = \sum_{\boldsymbol{\sigma}} p(\boldsymbol{\sigma})\, \delta(\sigma_i = a). \tag{1.16}$$

$$f_{i,j}(a,b) = \sum_{\boldsymbol{\sigma}} p(\boldsymbol{\sigma})\, \delta(\sigma_i = a)\delta(\sigma_j = b). \tag{1.17}$$

While theoretical results on this model, including phase transitions (see Wu (1982) for a review), have been obtained, the question of the inference of the interaction tensor $\boldsymbol{J}$ from data remains difficult. In particular, computing the partition function Z involves a sum over all possible states, whose number grows exponentially with the number of sites and possible states in such a way that it becomes impossible to compute exactly as soon as those numbers become relevant for practical applications. To overcome this limitation, approximations to the exact inference problem have been proposed, and refining those methods remains an open problem (Morcos et al. (2011), Cocco et al. (2013), Ekeberg et al. (2013) to cite a few).

1.6 Restricted Boltzmann Machines

Restricted Boltzmann Machine models are defined by separating the sites in two groups, one "visible" layer $\boldsymbol{v}$ of n units and one hidden layer $\boldsymbol{h}$ of m units, and allowing connectivity only between neurons of different layers, as shown in Figure 1.2**A**. The probabilities and energy function in these models have the same form as the ones of an Ising model, that is:

$$p(\boldsymbol{v},\boldsymbol{h}) = \frac{1}{Z} e^{-H(\boldsymbol{v},\boldsymbol{h})}. \tag{1.18}$$

$$Z = \sum_{\boldsymbol{v},\boldsymbol{h}} e^{-H(\boldsymbol{v},\boldsymbol{h})}. \tag{1.19}$$

$$H(\boldsymbol{v},\boldsymbol{h}) = -\sum_i \boldsymbol{v}_i b_i^{(v)} - \sum_j \boldsymbol{h}_j b_i^{(h)} - \sum_{i,j} \boldsymbol{v}_i W_{ij} \boldsymbol{h}_j. \tag{1.20}$$

One of the main benefits of Restricted Boltzmann Machines is that they can be used as generative models: conditioned on the state of the visible layer, the energy 1.20 can be used to define a probability distribution on the hidden layer states which can then be sampled. Afterwards, a similar strategy can be used to obtain a new visible state, and this procedure, called Alternating Gibbs Sampling and illustrated in Figure 1.2**B**, defines a dynamics on the space of visible layer configurations.

While this structure (and the related Boltzmann Machines, which allow interactions within a given layer) has been considered for cognitive science as early as 1985 (see Ackley et al. (1985), and Rumelhart et al. (1986), Chapter 6), the inference procedure (more generally referred to as the training algorithm in the context of Machine Learning) was too complex for practical applications. After it was subsequently refined in Hinton (2002) and Tieleman (2008), it was met with major empirical success (see Larochelle and Bengio (2008), Salakhutdinov and Hinton (2009), Lee et al. (2009), Nair and Hinton (2010), Hinton (2012), Mohamed et al. (2012)). Numerous theoretical results have also been obtained on these models, a recent review of which can be found in Decelle and Furtlehner (2021). In particular, phase diagrams can be constructed that show the existence of a *compositional phase*, see Tubiana and Monasson (2017), in which each neuron in the hidden layer is associated with "intermediate level" features, such as strokes in the case of MNIST digit

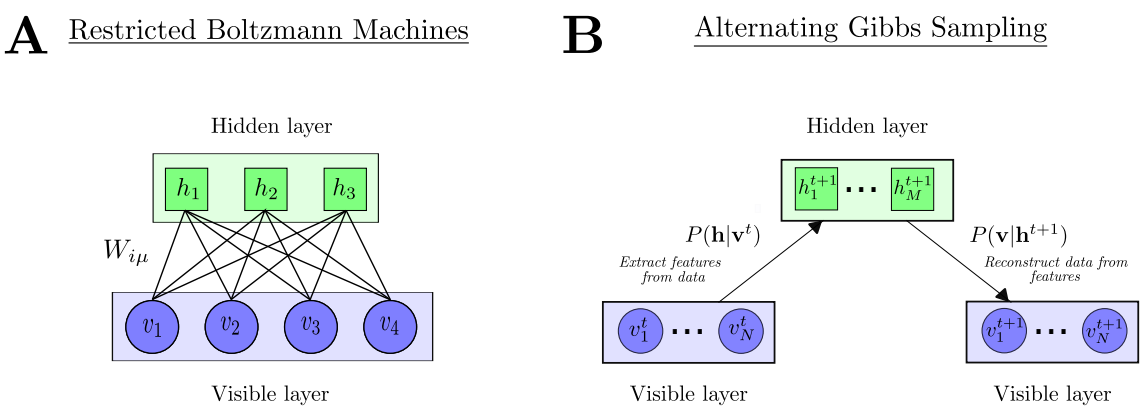


Figure 1.2: Illustration of the structure and sampling procedure for a Restricted Boltzmann Machine. **A**: The RBM presents itself as a special case of Ising model, in which the neurons are separated into two non-overlapping populations called the "visible" and "hidden" layer, and connections are allowed only between neurons of different layers. **B**: The Alternating Gibbs Sampling procedure allows to generate a dynamics on the visible layer configurations by, first, fixing the visible layer state to obtain a probability distribution on the hidden layer, then sampling from this new distribution a new hidden state, and finally using the newly sampled hidden state to define a probability distribution over visible states from which to sample, finishing one step of the AGS dynamics. Figure adapted from Roussel et al. (2021) with permission from the authors.

recognition (other typical behaviors are that each hidden unit encodes a single example from the dataset, or an "averaged" version of many similar examples).

These models are often used in practice as they are **generative**, *i.e.* can be used to output data such as images, protein sequences or sentences, following a distribution that was inferred from examples. Unfortunately, despite the elegance of the AGS sampling procedure, it often fails to exit local minima of the energy landscape (for example, outputting only slight variations around one of the training images); proposed solutions to this issue include stacking several RBMs (Bengio et al., 2013b), and introducing more involved dynamics in the hidden space (Roussel et al., 2021). Because of these limitations, new models such as Generative Adversarial Networks and Variational Autoencoders have been proposed as more flexible and powerful alternatives, which we present in more details in Chapter 5.

1.7 Markov Chains

A Markov Chain is a statistical model in which a system evolves in discrete time, and such that its state at time $t+1$ depends only on its state at time t[2]. This characteristic, often referred to as the **Markov property**, can be thought of as making the system *memoryless*: the past history of the system has no direct impact on its future, and the conditional dependency graph of such models is a chain.

If we further restrict the states $s \in \mathcal{S}$ of the system to be discrete, all information about the dynamics of the system can be represented using a single matrix $\mathcal{T}$, containing in position (i, j) the transition probability from state j to state i, and a vector $\boldsymbol{p}_0$ whose

[2]It is also possible to consider continuous-time models in which the future evolution of the system only depends on its current state, or systems evolving in discrete-time but whose states live in a continuous space; we will not consider such processes in any of the following.

i–th component contains the probability that the system is in state i at the first time-step. It is then possible to compute the probability of all states at any time as:

$$\boldsymbol{p}_t = \boldsymbol{T}^t \boldsymbol{p}_0. \tag{1.21}$$

Under certain conditions of irreducibility and aperiodicity of the transition matrix $\boldsymbol{T}$, the probability distribution vector is guaranteed to converge towards a stationary value $\boldsymbol{p}^*$ which is a fixed-point of the dynamics:

$$\boldsymbol{T}\boldsymbol{p}^* = \boldsymbol{p}^*.$$

If the number of possible states $|\mathcal{S}|$ of the system remains reasonably large, finding the stationary distribution is an easy task and can be accomplished either by iteratively multiplying p_0 by $\boldsymbol{T}$ (each time, requiring $|\mathcal{S}|^2$ operations), or by directly computing the eigendecomposition of $\boldsymbol{T}$ (requiring $O|\mathcal{S}|^3$ operations). Since the time of convergence towards the stationary distribution depends on the value of the eigenvectors, either method can be faster, depending on the exact values of the transition probabilities. However, both methods have memory space requirement of order $|\mathcal{S}|^2$, as they need to store at a given time the transition matrix, which might be unreasonable when the state space becomes large.

One example in which this is the case is the one of the PageRank algorithm, used by Larry Page and Sergey Brin as the basis for the Google search engine. The system which they were trying to model is the one of a "random surfer", who starts on a random page somewhere on the internet, then randomly moves to one of the pages that are linked from its current page; the stationary probability vector of this Markov Chain would be used to assess the relevance of each page in the World Wide Web graph. Since the number of web pages is so large that no computer could hold the transition matrix in memory, the natural solution to estimate this probability distribution was to rely on Monte-Carlo estimation, described in the following paragraph, and simulate the random-surfer process.

Monte Carlo method In practice, computing probabilities in high-dimension, might be too complicated to be done analytically. In those cases, one might exploit the deep relationship between integration and probabilities: the probability of an event $\mathcal{E}$ is simply the fraction of the volume of the universe Ω in which $\mathcal{E}$ is true: this is the Monte Carlo method.

An historic "application" of this idea is the one of Buffon's needle, in which a needle is dropped on a plane, marked with parallel lines, and one tries to estimate the probability that the needle does not lie across any of the bands. From direct analytical integration, one can derive this probability as a function of the length of the needle and the distance between the lines. This expression can then be used to obtain an approximation of π by performing that experiment of needle tossing in the real world. Nowadays, this kind of experiment is often performed through simulations, relying on the use of pseudo-random number generators such as the Mersenne Twister (Matsumoto and Nishimura, 1998) to generate the random draws necessary for the estimation process much more efficiently (and with less possible biases) than through a physical experiment. One example in which Monte Carlo methods allow for results that could not be obtained otherwise is the one of estimating the stationary probability vector of large Markov Chains, for which the connectivity matrix cannot be stored on physical memory while the probability vector itself can (since it scales linearly in the number of states instead of quadratically).

In some cases, even Monte-Carlo estimation remains impractical, and instead analytical approximation methods must be used, such as mean-field theory (Negele, 1982; Barabási et al., 1999; Opper and Saad, 2001).

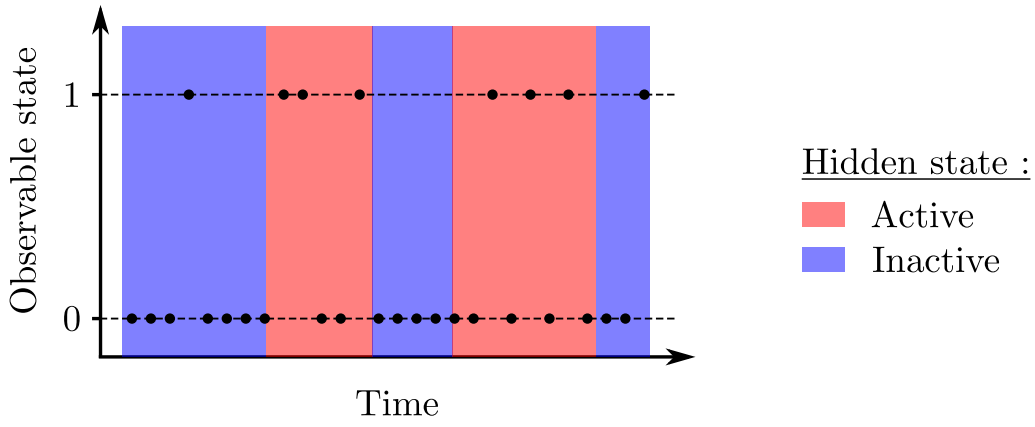


Figure 1.3: Example of a Hidden Markov Model with two hidden states (a communication line being either active or inactive), and two observable states (the current bit transmitted on the line being 0 or 1). When the line is inactive, the line transmits mostly 0, except on rare occasions where a fluctuation of the line potential accidentally transmits a 1. When the line is active, it transmits a signal which contains half of 0 and half of 1.

1.8 Hidden Markov Models

We now consider an extension of Markov Chains, called the Hidden Markov Models, in which a system has both a **hidden** state, evolving through a Markov process, and an **observable** state, conditioned only on the current value of the hidden state. For example, one could consider the case of a telecommunication line, able to transmit discrete binary sequences from the value of electrical tension in a copper wire; the current value of that binary signal constitutes the observable state. This communication line can be either **active**, when someone is using it to transmit a message, or **inactive** when no one is using it. In a period where the line is active, the observable state has equal probabilities of being 0 or 1, assuming the message was coded efficiently; when the line is inactive, the observable state will almost certainly be 0, but fluctuations in ground potential and other external factors might result in a small probability, *e.g.* 1% of a 1 being observed even if the line is inactive; this situation is represented in Figure 1.3. Hidden Markov Models have also been used in a wide range of other applications, from speech recognition (Varga and Moore, 1990) to analysis of molecular sequences in biology (Felsenstein and Churchill, 1996). The conditional dependency graph of a Hidden Markov Model is presented in Figure 1.4, and three quantities are needed to characterize the probability distribution: the initial hidden state distribution $\boldsymbol{\pi}$, the hidden state transition probabilities $\boldsymbol{T}$, and the emission probabilities of the observable states given the hidden state $\boldsymbol{E}$.

We will use this model in order to illustrate some of the questions that can be addressed using the formalism of Probabilistic Graphical Models, most notably detailing the procedure used to infer the hidden state sequence knowing the transition and emission probabilities, called Viterbi Decoding and a particular case of the Belief Propagation algorithm (Pearl, 1982; Kabashima and Saad, 1998), and the Baum-Welch algorithm, a particular case of Expectation Maximization (Dempster et al., 1977), used to infer the value of the probabilities.

Viterbi decoding An interesting, and practically important, problem concerning Hidden Markov Models consists in determining from a sequence of observed states which periods correspond to an active transmission, and which correspond to idle time, *i.e.* infer

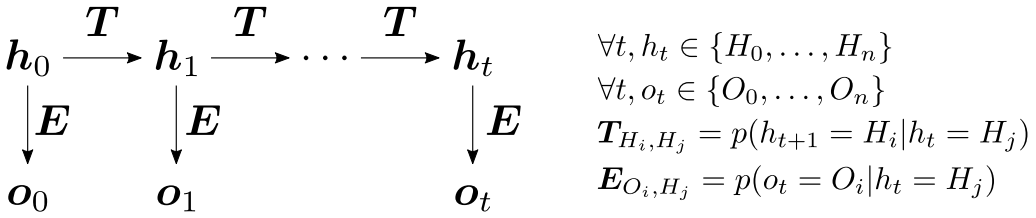


Figure 1.4: Graph of conditional dependencies in a Hidden Markov Model. The internal state h evolves according to a Markov Chain, *i.e.* the distribution of h_t is conditioned only on the previous internal state h_{t-1}; the observable state o_t is conditioned only on the internal state at the same time h_t.

the sequence of hidden states h_t from a sequence of observed states o_t.

To do so, let us begin by considering the probability of a particular hidden state h_t, conditioned on the sequence of observed states $\boldsymbol{o} = \{o_t, t = 1, \ldots, T\}$. From the Bayes rule and the conditional independences of our graphical model, we have:

$$\begin{aligned} p(h_t|\boldsymbol{o}) &= \frac{p(\boldsymbol{o}|h_t)\, p(h_t)}{p(\boldsymbol{o})} \\ &= \frac{p(o_0, \ldots, o_t|h_t)\, p(o_{t+1}, \ldots, o_T|h_t)\, p(h_t)}{p(\boldsymbol{o})} \\ &= \frac{p(o_0, \ldots, o_t, h_t)\, p(o_{t+1}, \ldots, o_T|h_t)}{p(\boldsymbol{o})} \\ &:= \frac{\alpha_t(h_t)\, \beta_t(h_t)}{p(\boldsymbol{o})} \\ &= \frac{\alpha_t(h_t)\, \beta_t(h_t)}{\sum_{h_t} \alpha_t(h_t)\beta_t(h_t)}, \end{aligned} \tag{1.22}$$

where $\alpha_t(h_t) = p(o_0, \ldots, o_t, h_t)$ is the probability of obtaining hidden state $\boldsymbol{h}_t$ at time t while emitting observable states $(o_0, \ldots, o_t)$, and $\beta_t(h_t) = p(o_{t+1}, \ldots, o_T|h_t)$ is the probability of emitting observable states $(o_{t+1}, \ldots, o_T)$ given that the system is in state h_t at time t[3]. The last equality comes from the normalization of conditional probabilities $\sum_{h_t} p(h_t|\boldsymbol{o}) = 1$.

Using once again the conditional independences expressed in the graphical model of Figure 1.4, one can show that these two quantities can be computed iteratively as:

$$\alpha_t(h_{t+1}) = \sum_{h_t} \alpha_t(h_t)\, T_{h_{t+1},h_t}\, E_{o_{t+1},h_{t+1}}. \tag{1.23}$$

$$\beta_t(h_t) = \sum_{h_{t+1}} \beta_{t+1}(h_{t+1})\, T_{h_{t+1},h_t}\, E_{o_{t+1},h_{t+1}}. \tag{1.24}$$

The computation of the α quantities requires a forward recursion, while the computation of β quantities requires a backwards recursion; the corresponding initialization conditions are $\alpha_0(h_0) = p(o_0|h_0)\pi_{h_0}$ and $\beta_T(q_T) = \mathbf{1}$. Having computed both of these quantities, one can deduce the conditional probabilities $p(h_t|\boldsymbol{o})$, and therefore deduce the most likely value of h_t by finding the value of h_t that maximizes that probability.

[3]We explicitly introduce the underscript t to underline that both α and β are defined at each time t, for all possible values $H_0, \ldots, H_n$ of the internal state h_t.

Baum-Welch algorithm Viterbi decoding made the major assumption that all parameters $\boldsymbol{E}, \boldsymbol{T}, \boldsymbol{\pi}$ of the model were known *a priori* and not **inferred** from examples. In practice, the transition probabilities are usually unknown, and they can be determined by using an Expectation Maximization procedure, known in the case of Hidden Markov Models as the Baum-Welch algorithm.

Doing so requires the computation of co-occurrence probabilities

$$\xi_t(h_t, h_{t+1}) = p(h_t, h_{t+1}|\boldsymbol{o}), \tag{1.25}$$

which can be obtained using the previously derived recursions as:

$$\xi_t(h_t, h_{t+1}) = \frac{\alpha_t(h_t)\, E_{o_{t+1},h_{t+1}}\, \beta_{t+1}(h_{t+1})\, T_{h_{t+1},h_t}}{p(\boldsymbol{o})}. \tag{1.26}$$

Denoting as $\gamma_t(h_t) = p(h_t|\boldsymbol{o})$ the probability of hidden state h_t conditioned on the entire sequence of observed states, we can write the EM algorithm as the alternance between two phases:

- Expectation phase: the α, β and γ quantities are computed, as well as the most likely sequence of hidden states $\boldsymbol{h}^*$ given the current estimation of the HMM parameters $\hat{\boldsymbol{T}}$, $\hat{\boldsymbol{E}}$ and $\hat{\boldsymbol{\pi}}$.
- Maximization phase: update the estimates for the parameters as:

$$\hat{\boldsymbol{T}}_{H_i,H_j} = \frac{\sum_{t=0}^{T} \xi_t(H_j, H_i)}{\sum_{t=0}^{T} \gamma_t(H_i)}. \tag{1.27}$$

$$\hat{\boldsymbol{E}}_{O_i,H_j} = \frac{\sum_{t=0}^{T} \gamma_t(H_i)\, \mathbf{1}_{o_t=O_j}}{\sum_{t=0}^{T} \gamma_t(H_i)}. \tag{1.28}$$

$$\hat{\boldsymbol{\pi}}(H_i) = \gamma_0(H_i). \tag{1.29}$$

Formally, the Baum-Welch algorithm as described here works on a single sequence of observable states. It is however straightforward to extend it to the case of multiple sequences, assumed to follow a common underlying distribution (*i.e.* sharing the same values of the $\boldsymbol{T}$, $\boldsymbol{E}$ and $\boldsymbol{\pi}$ parameters). The recursions of the Expectation phase are performed independently for each sequence in the batch, and the sums over time indices during the maximization phase are replaced by sum over both time, and sequence index.

1.9 Overfitting and regularization

Until now, we have only considered the ideal case in which we were able to compute the true expected values of all quantities required for the inference process. In practice, this is often an unrealistic constraint, and the amount of samples that are available for learning is finite, introducing non-zero variance in the expectations, and therefore in the inferred models. Because of this, one might observe the so-called **overfitting** phenomenon, in which the model very accurately fits the training data but generalizes poorly to previously unseen examples. To illustrate this idea, we consider in Figure 1.5 the case of a Neural Network classifier (more details on those in Chapter 4) which constructs an overly complex decision boundary in order to fit every single training data point; when new examples are added, it turns out that some of those points were **outliers**, which do not accurately represent the distribution that they were drawn from, and the inferred model does not achieve nearly as good results on these new points.

In order to address overfitting, a common solution is to add a **regularization** penalty in the likelihood used for the inference. Two very common choices for this penalty are the L_1 and L_2 norm of the parameters being inferred; intuitively, L_1 penalties are used to encourage sparsity in the solutions, but usually lead to MAP equations that cannot be solved analytically, while L_2 penalties are usually easier to analyze theoretically and encourage smoothness of the energy landscape, which is usually a desirable property to avoid overfitting (in the analogy of Figure 1.5, the L_2 norm of the weight-vector would correspond to the total curvature of the boundary).

The addition of this regularization loss can be seen as a form of **prior** on the possible values of the parameters. Let us illustrate this idea in the simple case of inference of the meanμ of a one-dimensional Gaussian variable: p samples $(s_1, \dots, s_p)$ are drawn from the true underlying distribution. The likelihood of these samples for a given underlying mean μ (assuming a variance σ equal to 1) is:

$$p(s_1, \dots, s_p|\mu) = \frac{1}{(2\pi)^{p/2}} e^{-\frac{1}{2}\sum_{k=1}^{p}(s_k-\mu)^2}. \tag{1.30}$$

If we now introduce a Gaussian prior on the value of the mean μ:

$$p(\mu) = \sqrt{\frac{\gamma}{(2\pi)}} e^{-\frac{\gamma}{2}\mu^2}, \tag{1.31}$$

where γ determines the magnitude of μ that we expect (or, conversely, the strength of the regularization that forces μ towards 0), we can obtain the joint probability of $\boldsymbol{s}$ and μ. Finally, we can also derive the posterior probability:

$$p(\mu|\boldsymbol{s}) = \sqrt{\frac{1}{(2\pi\tilde{\sigma}^2)}} e^{-\frac{1}{2\tilde{\sigma}^2}(\mu-\tilde{\mu})^2}, \tag{1.32}$$

where we introduced the following quantities:

$$\tilde{\sigma} = (p+\gamma)^{-1}. \tag{1.33}$$

$$\tilde{\mu} = \frac{p}{p+\gamma}\,\overline{s}. \tag{1.34}$$

$$\overline{s} = \frac{1}{p}\sum_{k=1}^{p} s_k. \tag{1.35}$$

The Maximum A Posteriori inference can then be formulated as finding the maximum (with respect to μ) of the log-likelihood per sample:

$$\ell(\mu) = \underbrace{\frac{1}{2}(\mu^2 - \overline{s}\mu)}_{\text{Unconstrained likelihood}} + \underbrace{\frac{\gamma}{2p}\mu^2}_{L_2 \text{ regularization}}, \tag{1.36}$$

so that the Gaussian prior imposed on the value of μ manifests itself as an additional L_2 penalty in the log-likelihood. Since the prior is of the same family distribution as the conditional probability, a situation referred to as a *conjugate prior*, it has the same effect as introducing a new (specially weighted) sample in the dataset (in this case, equal to 0), and therefore has the effect of biasing the inference towards $\mu = 0$.

In Chapter 2, we consider an extension of this computation, the problem of regularization in Maximum a Posteriori Inference of multivariate Gaussian Models, and show the existence of an optimal regularization in this setting. In Chapter 4, we present the

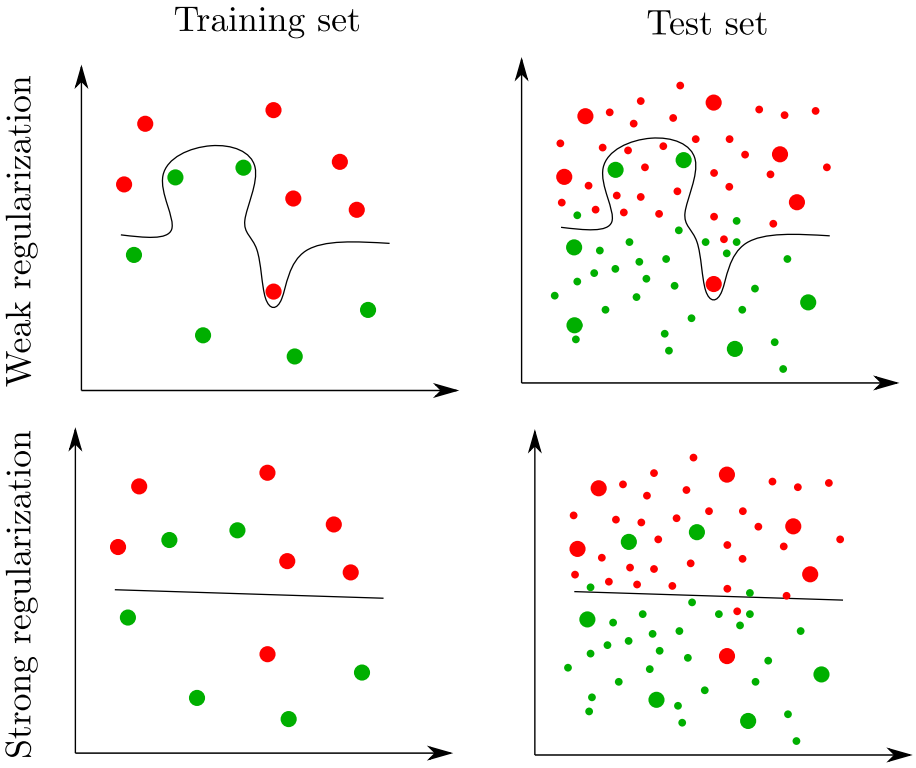


Figure 1.5: Illustration of the overfitting phenomenon. In the case of weak regularization, the model is able to fit perfectly the training set by constructing a very irregular decision boundary. However, the underlying data distribution was more regular, and when additional examples are added, they are incorrectly classified. On the other hand, a strongly regularized network performed poorly on the training set, but finds a very regular decision boundary which generalizes better to the new examples of the test set than the weakly regularized model.

SVM model, which can be seen as an L_2–regularized version of the Perceptron, which we present in the same Chapter. Finally, we note that more meaningful regularization terms can be chosen to enforce specific characteristics of the solutions, such as invariances, that are known from high-level knowledge about the task; this is the approach we took for the elaboration of the model we study in Chapter 8, where the "direct" and "inverse" model losses impose a specific structure on the visual representation which makes it easier for the recurrent part of the model to learn a resetting behavior.

Chapter 2

Regularization in Gaussian Model inference

Abstract

Understanding the role of regularization is a central question in Statistical Inference. Empirically, well-chosen regularization schemes often dramatically improve the quality of the inferred models by avoiding overfitting of the training data. We consider here the particular case of L_2 and L_1 regularizations in the Maximum A Posteriori (MAP) inference of generative pairwise graphical models. Based on analytical calculations on Gaussian multivariate distributions and numerical experiments on Gaussian and Potts models we study the likelihoods of the training, test, and 'generated data' (with the inferred models) sets as functions of the regularization strengths. We show in particular that, at its maximum, the test likelihood and the 'generated' likelihood, which quantifies the quality of the generated samples, have remarkably close values. The optimal value for the regularization strength is found to be approximately equal to the inverse sum of the squared couplings incoming on sites on the underlying network of interactions. Our results seem largely independent of the structure of the true underlying interactions that generated the data, of the regularization scheme considered, and are valid when small fluctuations of the posterior distribution around the MAP estimator are taken into account. Connections with empirical works on protein models learned from homologous sequences are discussed.

2.1 Introduction

Data-driven modeling is now routinely used to address hard challenges in an increasing number of fields of science and engineering for which first-principle approaches have limited success. Applications include the characterization and design of complex materials (Schmidt et al., 2019), shaped by the pattern of strong and heterogeneous interactions between their microscopic components. Performance of data-driven models strongly depends on the choice of their hyperparameters, such as the architecture, and the strengths of the regularization penalties. These parameters are generally set through empirical procedures, such as cross-validation with respect to a goodness-of-fit estimator. Unfortunately, this common approach often offers no insight about why these values of the parameters are optimal, and may not guarantee that the obtained models are well-behaved with respect to other estimators. This paper reports some efforts to address these issues for the specific case of L_p-norm regularization and probabilistic graphical models.

Probabilistic graphical models rely on the inference of the set of conditional dependencies between the variables under study, which, in turn, may be used to generate new configurations of these variables (MacKay, 2003). Regularization allows the graph of pairwise conditional dependence to satisfy some properties of interests, such as to be sparse or to have dependence factors bounded from above. Among the huge variety of applications of those models, substantial efforts have been devoted over the past decades to applications to the modeling of proteins based on homologous, i.e. evolutionary related sequence data. Unveiling the relations between the functional or structural properties of a protein and the sequence of its amino acids is a difficult task. Graphical model-based modeling consists of inferring a graph of effective interactions between the amino acids, which reproduce the low-order (1- and 2-point) statistics in the sequence data; for reviews, see (Cocco et al., 2018) for protein modelling and (Chau Nguyen et al., 2017) for general inference of graphical models with discrete variables. In practice, for proteins with few hundreds of amino acids, tens of millions of interaction parameters have to be inferred. To avoid overfitting, regularization of those interactions, often based on pseudocounts, or L_1- and L_2–norms are generally introduced, with intensities varying with the optimality criteria chosen by the authors (Barton et al., 2014; Haldane and Levy, 2019). For instance, Ekeberg et al. chose regularization strength scaling linearly with the number of data (sequences) (Ekeberg et al., 2013, 2014) to maximize the quality of structural predictions. Hopf et al. chose linear scaling with the dimension of the data (sequence length) and with the number of possible amino-acid types (generally, $q = 20$) for predicting the fitness effects resulting from mutations along the sequence (Hopf et al., 2017). The rationale for these scalings and what they tell us about the underlying properties of the protein system remains unclear. In addition, whether these scalings are appropriate for generating new data points, i.e. for the design of new protein sequences having putative properties is not known, and other regularization schemes have been proposed (Barrat-Charlaix et al., 2021)

In the following, we propose to study the role of regularization in the inference process, replacing Potts models by Multivariate Gaussian models in order to make the problem analytically tractable in some limiting cases. We show that two natural definitions for the optimal values of the regularization strength are in practice very close to one another, and that their common value can be related to the amplitude of the ground-truth interactions, in agreement with experimental observations. Our paper is organized as follows. In Section 2, we introduce the Gaussian model and the regularizations of interest. Numerical results are reported in Section 3. Section 4 is devoted to the analytical studies of the poor and

excellent sampling limits. Last of all, some conclusions and perspectives are drawn in Section 5.

2.2 Gaussian Vectors Model and Regularization

2.2.1 Expression of likelihood in the large–size limit

In order to be able to model distributions over n–dimensional vectors, we consider first the multidimensional Gaussian distribution, often referred to as Gaussian Vectors or Spherical Model. In the following, we will only consider the case of centered Gaussian Vectors, for which the mean value of each component vanishes and the probability density is given by:

$$p(\boldsymbol{x}) = \frac{1}{\sqrt{(2\pi)^n \det(\boldsymbol{C}^{tr})}} e^{-\frac{1}{2}\boldsymbol{x}^T (\boldsymbol{C}^{tr})^{-1} \boldsymbol{x}} \ , \tag{2.1}$$

where $\boldsymbol{C}^{tr}$ is the $n \times n$–dimensional covariance matrix. Alternatively we may define the underlying data distribution through an interaction matrix $\boldsymbol{J}^{tr}$, which represents the interaction strength between the variables (vector components). This interaction matrix $\boldsymbol{J}^{tr}$ is related to the true covariance matrix $\boldsymbol{C}^{tr}$ of the data through

$$\boldsymbol{C}^{tr} = (\mu^{tr} \boldsymbol{I} - \boldsymbol{J}^{tr})^{-1} \ , \tag{2.2}$$

where μ^{tr} was introduced to impose the spherical normalization constraint $Tr(\boldsymbol{C}^{tr}) = n$. Denoting as $(j_1^{tr}, \ldots, j_n^{tr})$ the eigenvalues of $\boldsymbol{J}^{tr}$, the normalization condition can be written, in the large n limit, as

$$1 - \frac{1}{n} \sum_{k=1}^{n} \frac{1}{\mu^{tr} - j_k^{tr}} = 0. \tag{2.3}$$

As the covariance matrix is non-negative we are looking for the unique value of μ^{tr} in $[\max_k\{j_k^{tr}\}, +\infty[$ that satisfies this equation.

In the following, we will be interested in inferring the interaction matrix $\boldsymbol{J}^{tr}$ from an empirical approximation $\boldsymbol{C}^{emp}$ of the correlation matrix obtained using $p = \alpha\, n$ samples $(\boldsymbol{x}^1, \ldots, \boldsymbol{x}^p)$ as:

$$\forall (i,j) \in [1,n]^2, \ \boldsymbol{C}^{emp}_{i,j} = \frac{1}{p} \sum_{k=1}^{p} \boldsymbol{x}_i^k \boldsymbol{x}_j^k \ . \tag{2.4}$$

We define the posterior likelihood of any interaction matrix $\boldsymbol{J}$ given the empirical covariance matrix $\boldsymbol{C}^{emp}$,

$$p(\boldsymbol{J}|\boldsymbol{C}^{emp}) = e^{-n\, E(\boldsymbol{J})} \ , \tag{2.5}$$

where the energy function $E(\boldsymbol{J})$ reads

$$E(\boldsymbol{J}) = -\frac{\alpha}{2} Tr(\boldsymbol{J}\boldsymbol{C}^{emp}) + \alpha \log Z(\boldsymbol{J}) + \frac{\gamma}{4} Tr(\boldsymbol{J}^2). \tag{2.6}$$

In the expression above the first two terms correspond to the standard likelihood of a given Gaussian Model given the empirical covariance, while the last term expresses a penalty on the L_2 norm of the inferred interaction matrix. The strength of this regularization is controlled by the parameter γ.

Symbol	Quantity
$\boldsymbol{I}$	The identity matrix
n	Dimension of the Gaussian Vectors
p	Number of samples
α	Sampling ratio p/n
γ	The strength of the L_2 penalty
$\boldsymbol{J}$	Dummy variable standing for an interaction matrix
$\boldsymbol{C}$	Dummy variable standing for a covariance matrix
$\boldsymbol{J}^{tr}$	True interaction matrix of the underlying model
$\boldsymbol{C}^{tr}$	True covariance matrix of the underlying model
$\boldsymbol{C}^{tr,rot}$	True covariance matrix, in the diagonalizing basis of $\boldsymbol{C}^{emp}$
c^{tr}	An eigenvalue of the true covariance matrix
μ^{tr}	Lagrange multiplier imposing the spherical constraint on $\boldsymbol{J}^{tr}$
$\boldsymbol{C}^{emp}$	Empirical covariance matrix obtained from $p = \alpha n$ samples
c^{emp}	Eigenvalue of the empirical covariance matrix
$\boldsymbol{J}^*$	Interaction matrix obtained from Maximum A Posteriori inference
j^*	Eigenvalue of the MAP inferred interaction matrix
μ^*	Lagrange multiplier imposing the spherical constraint on $\boldsymbol{J}^*$

Table 2.1: All quantities used in the inference procedure. Please note that the empirical covariance matrix $\boldsymbol{C}^{emp}$ and its eigenvalues are stochastic quantities for a given underlying interaction model $\boldsymbol{J}^{tr}$ (since they depend on the exact samples drawn). Additionally, we will assume the eigenvalues c to be ordered from largest to smallest, and denote with a lower–index k both c_k^{emp} (the k–th largest eigenvalue of $\boldsymbol{C}^{emp}$) and j_k^* the corresponding eigenvalue of $\boldsymbol{J}^*$ (see eqn. 2.10).

The partition function $Z(\boldsymbol{J})$ of the so-called spherical spin model reads

$$\begin{aligned} \log Z(\boldsymbol{J}) &= \int_{\boldsymbol{x}\in\mathbb{R}^n} \delta(\boldsymbol{x}^2 = n)\, e^{\frac{1}{2}\sum_{i\neq j} x_i J_{ij} x_j} \\ &= n \min_{\mu} \left(\frac{\mu}{2} - \frac{1}{2n} \log(\det(\mu \boldsymbol{I} - \boldsymbol{J})) \right) \end{aligned} \tag{2.7}$$

to the dominant order in n. The parameter μ can be interpreted as a Lagrange multiplier, introduced to impose the spherical constraint $Tr(\boldsymbol{C}) = n$, which corresponds exactly to the normalization condition (2.3) but with the eigenvalues of the true interaction matrix j^{tr} replaced by the ones of $\boldsymbol{J}$.

Our goal will be to minimize the energy (2.6) with respect to the interaction matrix $\boldsymbol{J}$; the matrix $\boldsymbol{J}^*$ minimizing the energy will be called inferred matrix and will be our primary object of study. We also define μ^* the Lagrange multiplier imposing the spherical constraint on this inferred model, and $\boldsymbol{C}^*$ the covariance matrix of the inferred model. For reference, we define in Table 2.1 all the different quantities that we will be considering and their associated notations.

2.2.2 Maximum A Posteriori estimator of the interaction matrix

When γ is equal to 0, the regularization disappears and the Maximum Likelihood estimation of $\boldsymbol{J}^*$ is exactly equal to the one computed from the empirical covariance $\boldsymbol{C}^{emp}$; when γ goes to infinity, the regularization becomes so strong that the inferred interaction matrix is exactly equal to $\boldsymbol{0}$; in the general case of finite γ, we find $\boldsymbol{J}^*$ by computing $\frac{\partial E}{\partial \boldsymbol{J}}(\boldsymbol{J}^*)$,

which yields the Maximum A Posteriori (MAP) equation:

$$\gamma \boldsymbol{J}^* - \alpha \boldsymbol{C}^{emp} + \alpha(\mu^* \boldsymbol{I} - \boldsymbol{J}^*)^{-1} = 0. \tag{2.8}$$

According to equation (2.8) the inferred interaction matrix $\boldsymbol{J}^*$ is diagonal in the same vector basis as the empirical covariance matrix $\boldsymbol{C}^{emp}$. It is therefore possible to rewrite this equation in terms of the eigenvalues (respectively, j^*, c^{emp}) of those matrices[1]:

$$\gamma\, j^{*2} - (\gamma\mu^* + \alpha c^{emp})\, j^* + \alpha(\mu^* c^{emp} - 1) = 0. \tag{2.9}$$

Since the discriminant $\Delta = (\alpha c^{emp} - \gamma\mu^*)^2 + 4\alpha\gamma \geq 0$, the eigenvalue $j^*(c^{emp})$ always exists in $\mathbb{R}$ and is found to be equal to:

$$j^*(c^{emp}) = \frac{1}{2\gamma}\left(\alpha c^{emp} + \gamma\mu^* - \sqrt{(\alpha c^{emp} - \gamma\mu^*)^2 + 4\alpha\gamma}\right). \tag{2.10}$$

It should be noted here that this is in fact an auto-consistent equation: μ^* is used to compute the eigenvalues j^*, which in turn are used to compute μ^*. In order to solve it, we consider μ^* to be a free parameter and make the expression of the inferred eigenvalues depend on two variables $j^*(c^{emp}, \mu^*)$. Introducing the corresponding expression into the normalization condition 2.3, we find that μ^* is the only root[2] of the residual function:

$$Res^{norm}(\mu) = 1 - \frac{1}{n}\sum_k \frac{1}{\mu - \frac{1}{2\gamma}\left(\alpha c_k^{emp} + \gamma\mu - \sqrt{(\alpha c_k^{emp} - \gamma\mu)^2 + 4\alpha\gamma}\right)}. \tag{2.11}$$

In practice, the optimization of this residual is performed numerically in Python using the Van Wijngaarden-Dekker-Brent method Brent (2013), implemented within the SciPy package Virtanen et al. (2020). After obtaining the value of μ^*, the inferred interaction matrix $\boldsymbol{J}^*$ is obtained by computing its spectrum through equation 2.10 and changing the basis back from the inference basis (which diagonalizes the empirical covariance $\boldsymbol{C}^{emp}$) to the original basis (in which the true interaction $\boldsymbol{J}^*$ was defined).

2.2.3 Likelihoods of the training, test, and generated sets

In order to be able to compare the quality of the inferred interaction matrix $\boldsymbol{J}^*$ as a function of the different parameters of the system (namely, α, γ and the true interaction matrix $\boldsymbol{J}^{tr}$) the first interesting quantity to define is the training likelihood:

$$L_{train} = \frac{1}{p}\sum_{k=1}^{p}\left[\frac{1}{2}\sum_{i,j} J_{ij}^*\, x_i^k x_j^k - \log Z(\boldsymbol{J}^*)\right], \tag{2.12}$$

which directly quantifies how well the examples of the training set are fit by the MAP estimator $\boldsymbol{J}^*$. By performing the summation over the sample index k, the likelihood can be rewritten as a function of the empirical covariance matrix $\boldsymbol{C}^{emp}$:

$$L_{train} = \frac{1}{2}\sum_{i,j} J_{ij}^* C_{ij}^{emp} - \log Z(\boldsymbol{J}^*). \tag{2.13}$$

[1] Because of equation 2.8, we know that to each eigenvalue of the empirical covariance matrix corresponds exactly one eigenvalue of the inferred interaction matrix.

[2] It can easily be shown that $\partial j^*(c^{emp})/\partial\mu^*$ is always positive; since $j^*(c^{emp}, \mu) < \mu$, we have that $Res(\mu)$ is well-defined for all values of μ; $\partial Res(\mu)/\partial\mu$ is always positive and therefore $Res(\mu)$ is monotonically increasing from $-\infty$ when $\mu \to -\infty$ to 1 when $\mu \to +\infty$, ensuring the unicity of the root.

A similar reasoning can be performed, this time considering the case where an infinite number of samples are drawn from the true underlying distribution (meaning that C^{emp} is replaced by $\boldsymbol{C}^{tr}$), corresponding to the average test error on samples independent of the training ones. This leads to the definition of the test likelihood:

$$L_{test} = \frac{1}{2}\sum_{i,j} J^*_{ij} C^{tr}_{ij} - \log Z(\boldsymbol{J}^*). \tag{2.14}$$

Finally, one can also consider the likelihoods of a 'generated set' of examples drawn using the inferred interaction matrix, with respect to this same inferred interaction matrix $\boldsymbol{J}^*$:

$$L_{gen} = \frac{1}{2}\sum_{i,j} J^*_{ij} C^*_{ij} - \log Z(\boldsymbol{J}^*). \tag{2.15}$$

It is possible to rewrite the "generated" likelihood using the MAP equation:

$$\begin{aligned} L_{gen} = \frac{1}{2}\sum_{i,j} J^*_{ij} C^*_{ij} - \log Z(\boldsymbol{J}^*) &= \frac{1}{2}\sum_{i,j} J^*_{ij} \frac{1}{\mu \boldsymbol{I} - \boldsymbol{J}^*}\Big|_{ij} - \log Z(\boldsymbol{J}^*) \\ \overset{(2.8)}{=} \frac{1}{2}\sum_{i,j} J^*_{ij}(\boldsymbol{C}^{emp}_{ij} - \frac{\gamma}{\alpha}\boldsymbol{J}^*_{ij}) - \log Z(\boldsymbol{J}^*) &= L_{train} - \frac{\gamma}{2\alpha}\sum_{i,j} {J^*_{ij}}^2. \end{aligned} \tag{2.16}$$

This form of the generated likelihood can be interpreted as a form of bias-variance trade-off: if an increase in the magnitude of the couplings is necessary to better fit the training set, it will increase the variance of the generated data and consequently decrease the generated set likelihood.

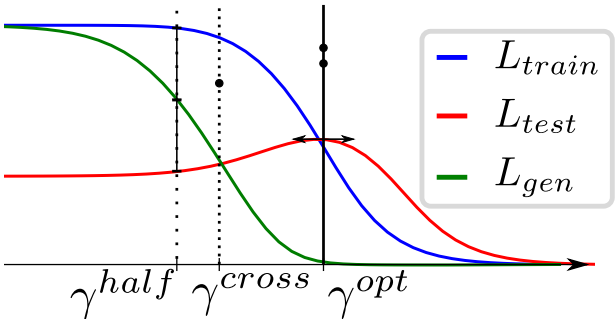
2.2.4 Generic dependence of the likelihoods upon regularization strength

Figure 2.1 is a sketch of the typical behaviours expected for the three log-likelihoods defined above as the regularization strength γ is varied:

- For weak regularization *i.e.* γ close to zero MAP inference is unconstrained, and the inferred covariance coincides with the empirical one. The value of the training likelihood is large, as the details of the training set are fitted. Consequently, the inferred model has poor generalization capability, and the test log-likelihood has a low value. This is a situation of *overfitting*. Generated data look like training data, so the generated likelihood is large.

- For strong regularization, *i.e.* large γ the regularization term in the energy becomes more important than the likelihood term, so that the MAP estimator $\boldsymbol{J}^*$ tends to zero; this is a case of *under fitting*, as the training, test, and generated likelihoods will be low. When γ goes to infinity, the three likelihoods converge to a common value,
$$L(\gamma \to \infty) = -\frac{n}{2}. \tag{2.17}$$

- In-between those two regimes, *i.e.* for intermediate values of γ the training likelihood is monotonically decreasing with γ, reflecting the increasing bias towards small couplings, and so is the generated likelihood. The test likelihood displays a non-monotonic evolution, and reaches a maximum for some regularization penalty γ^{opt}. While the presence of γ biases the inference, it also reduces its variance, and



- $\frac{L_{train}(\gamma^{half}) - L_{gen}(\gamma^{half})}{L_{gen}(\gamma^{half}) - L_{test}(\gamma^{half})} = 1$
- $L_{test}(\gamma^{cross}) = L_{gen}(\gamma^{cross})$
- $\partial_\gamma L_{test}(\gamma^{opt}) = 0$

Figure 2.1: Sketch of the expected behaviours of the likelihoods vs. regularization γ, and definitions of the three values of interest: γ^{half}, for which the generated likelihood is exactly in-between the train and test ones; γ^{cross}, for which the test and generated likelihoods are equal; γ^{opt}, for which the test likelihood is maximal. The difference between optimal and crossing likelihoods is strongly exaggerated for illustration purposes, as in practice they are found to be extremely close to each other in almost all circumstances.

hence allows for better generalization of the model to unseen examples. While the test likelihood always remains smaller than the training likelihood (as should be expected, the model cannot generalize better than it fits the available data), the test and generated likelihoods may cross at a certain value γ^{cross}, Figure 2.1. We also define the regularization γ^{half} for which the generated likelihood is half way between the train and test ones.

In the following we will study, through numerical experiments and analytical calculations the behaviour of these three regularization strengths of interest, and their dependence on the model defining parameters (number p of samples compared to the size n, structure of the coupling matrix, ...).

2.3 Numerical experiments

2.3.1 Gaussian Vectors Model

In order to study the dependence of $\gamma^{opt}, \gamma^{cross}, \gamma^{half}$ with the different parameters, we implemented the MAP inference procedure in Python (the code is available on GitHub).

The general procedure is as follows: first, an interaction matrix $\boldsymbol{J}^{tr}$ is randomly generated, according to an underlying distribution (see next subsections for details on the distributions we considered); then, a certain number $p = \alpha n$ samples are drawn from the Gaussian Vectors model with interactions $\boldsymbol{J}^{tr}$, and from those samples an empirical covariance matrix $\boldsymbol{C}^{emp}$ is derived; this matrix is then diagonalized, and the spectrum of the MAP interaction estimator $\boldsymbol{J}^*$ is computed through eqn. (2.10); the training and generated set likelihoods are computed directly using those eigenvalues, while the test likelihood requires the inversion of the diagonalization basis change in order to obtain the expression of $\boldsymbol{J}^*$ in the same basis as $\boldsymbol{C}^{tr}$.[3]

[3]Those two basis *a priori* coincide if and only if $\alpha \to \infty$.

Case of random quenched couplings

The condensation phase transition. We assume that the underlying interaction matrix is drawn from the Gaussian Orthogonal Ensemble, *i.e.* all its components are drawn at random and independently from a centered Gaussian distribution:

$$\forall\, i, j\,, \quad J_{ij}^{tr} \sim \mathcal{G}(0, \frac{\sigma}{\sqrt{n}}). \tag{2.18}$$

The presence of this $1/\sqrt{n}$ normalization ensures that the energy is extensive with n. The model is "infinite range" because all spins are interacting with all other spins with similar strengths, controlled by the parameter σ. As shown in Kosterlitz et al. (1976) the model exhibits a condensation phase transition when σ crosses the critical value $\sigma_c = 1$. For $\sigma > \sigma_c$ one eigenvalue of the covariance matrix scales linearly with n, while all others remain finite. This transition can be intuitively understood as follows. Since the interaction matrix $\boldsymbol{J}^{tr}$ has Gaussian entries, its eigenvalue distribution follows Wigner's semi-circle law, and ranges from -2σ and 2σ. As σ increases from small values, the value of the Lagrange multiplier μ imposing the spherical constraint becomes closer and closer to its lower-bound 2σ, and the gaps closes (in the infinite n limit) when $\sigma = \sigma_c$. For $\sigma > \sigma_c$ μ remains equal to 2σ, and the corresponding top eigenvector of $\boldsymbol{J}^{tr}$ gives rise to an extensively large eigenvalue in $\boldsymbol{C}^{tr}$. More precisely, when σ is larger than σ_c, the maximum eigenvalue of $\boldsymbol{C}^{tr}$ is equal to

$$c_{max}^{tr} = n \times \left(1 - \frac{1}{2\pi\sigma^2} \int_{-2\sigma}^{2\sigma} \frac{\sqrt{4\sigma^2 - j^2}}{2\sigma - j} dj\right) = n\, \left(1 - \frac{1}{\sigma}\right). \tag{2.19}$$

In this situation, the model generates configurations that are effectively constrained close to a subspace of dimension 1.

Evolution of the log-likelihoods with γ. Figure 2.2 shows the behaviours of the log-likelihoods with varying γ, for different regimes of low and high sampling fractions α. Vertical lines locate the three values of γ of interest. The overall shape of the curves agree with the expected behaviours sketched in Figure 2.1.

For small γ (overfitting regime), the value of the training likelihood is very large, irrespective of the value of α as the weak regularization allows the inference procedure to fit the training set without bias. The test loss, however, strongly varies α. For low sampling (small α) $\boldsymbol{C}^{emp}$ is essentially uncorrelated with $\boldsymbol{C}^{tr}$, and the test likelihood will be very low. If α is large, $\boldsymbol{C}^{emp}$ is almost equal to $\boldsymbol{C}^{tr}$, and the test likelihood will be very close to its training counterpart, both being very high. In all cases the generated and the training log-likelihoods coincides.

When γ is very large, the regularization term in the energy pushes the MAP estimator $\boldsymbol{J}^*$ towards 0. In this *underfitting* regime, all log-likelihoods tend to the same limit value, see eqn. (2.17).

For intermediate γ, we observe that the location of the maximum of the test likelihood, γ^{opt}, is very close to the value of the regularization strength γ^{cross} for which it crosses the generated log-likelihood. This unexpected results holds in most circumstances as reported in Figure 2.3, but small discrepancies can be observed at low sampling ratio α. Detailed analytical calculations for the Gaussian Vectors Model in Section 2.4 will allow us to confirm this numerical observation, and, in addition, to approximate their common value as a function of the sampling ratio α and of the "true" interaction matrix $\boldsymbol{J}^{tr}$:

$$\gamma^{opt} \simeq \gamma^{cross} \simeq \frac{n}{\sum_{i,j} (J_{ij}^{tr})^2}\ . \tag{2.20}$$

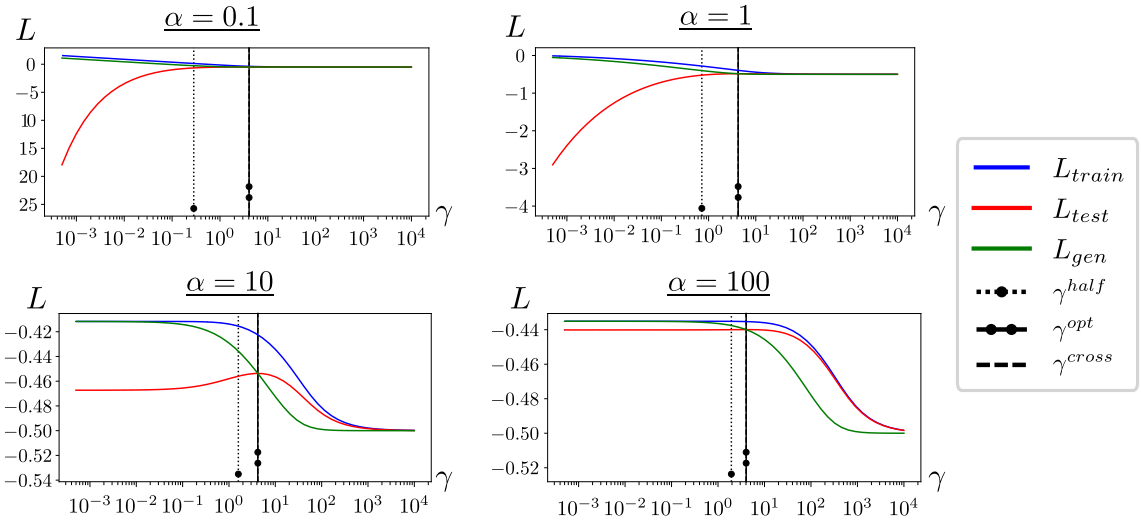


Figure 2.2: Evolution of the four likelihoods (normalized by n) as functions of the regularization strength γ for four different values of the sampling ratio α. In all cases, both training and generated likelihoods are monotonically decreasing, while the test likelihood is first increasing then decreasing; the training and test likelihoods never cross, while the generated and test likelihoods cross for a value of the regularization extremely close to the optimum of L_{test}.

Let us notice that γ^{half}, the regularization penalty at which the generated likelihood is the mean of the train and test ones, seems to approximately fullfill the following equality

$$\sum_{i,j} J^*_{ij}(\gamma^{half})\, C^{tr}_{ij} = \sum_{i,j} J^{tr}_{ij}\, C^*_{ij}(\gamma^{half})\ . \tag{2.21}$$

2.3.2 Numerical estimation of the regularization strengths

In order to compute the values of γ^{opt} and γ^{cross} as precisely as possible, we derived two *residuals*, *i.e.* functions of γ which are equal to 0 respectively when the test likelihood is optimal, or when the test and generated likelihoods are equal. Similarly to how μ^* was determined when solving the MAP equation, the roots of those residuals will be minimized using standard convex optimization routines to obtain high precision estimates of the optimal and crossing regularizations.

This approach is easilly illustrated in the case of the crossing regularization γ^{cross}. According to eqn. (2.42) the following function $Res^{cross}(\gamma)$ has its root equal to γ^{cross}:

$$Res^{cross}(\gamma) := \alpha \frac{\langle \boldsymbol{J}^*(\gamma)\,(\boldsymbol{C}^{emp} - \boldsymbol{C}^{tr})\rangle}{\langle \boldsymbol{J}^*(\gamma)^2\rangle} - \gamma. \tag{2.22}$$

For the estimation of the optimal regularization, the computation is more involved and relies on finding the derivative of L_{test} with respect to γ. Indeed, γ is equal to γ^{opt} when

$$Res^{opt}(\gamma) := \frac{\partial L_{test}}{\partial \gamma} \tag{2.23}$$

is equal to 0.

This derivative can be computed as:

$$\begin{aligned}\frac{\partial L_{test}}{\partial \gamma} &= \frac{1}{2}\sum_{i,j}\frac{\partial J^*_{ij}}{\partial \gamma}C^{tr}_{ij} - \frac{\partial \log Z(\boldsymbol{J}^*)}{\partial \gamma} \\ &= \frac{1}{2}\sum_{k}\frac{\partial j^*_{k}}{\partial \gamma}C^{tr,rot}_{k,k} - \frac{\partial \log Z(\boldsymbol{J}^*)}{\partial \gamma},\end{aligned} \tag{2.24}$$

where $\boldsymbol{C}^{tr,rot}$ is the true correlation matrix after changing the basis to the inference basis in which $\boldsymbol{C}^{emp}$ is diagonal.
We begin by computing

$$\begin{aligned}\partial_\gamma j^*_k &= \partial_\gamma\left[\frac{1}{2\gamma}\alpha c_k + \gamma\mu^* - D_k\right] \\ &= A_k\partial_\gamma\mu^* + B_k - \frac{j^*_k}{\gamma},\end{aligned} \tag{2.25}$$

where we introduced

$$D_k = \sqrt{(\alpha c^{emp}_k - \gamma\mu^*)^2 + 4\alpha\gamma}, \tag{2.26}$$

$$A_k = \frac{1}{2}\left(1 - \frac{\gamma\mu^* - \alpha c^{emp}_k}{D_k}\right), \tag{2.27}$$

$$B_k = \frac{1}{\gamma}\left(\mu^* A_k - \frac{\alpha}{D_k}\right). \tag{2.28}$$

Then, we have that

$$\partial_\gamma \log Z = n\partial_\gamma\mu^* - \frac{1}{2}\sum_k \frac{\partial_\gamma\mu^* - \partial_\gamma j^*_k}{\mu^* - j^*_k}. \tag{2.29}$$

Finally, we can compute $\partial_\gamma\mu$ by first noting that:

$$\frac{1}{2}\sum_k \frac{1}{\mu^* - j^*_k} = 1, \tag{2.30}$$

hence

$$\sum_k \frac{\partial_\gamma\mu^* - \partial_\gamma j^*_k}{(\mu^* - j^*_k)^2} = 0, \tag{2.31}$$

and therefore

$$\partial_\gamma\mu = \left[\sum_k \frac{\partial_\gamma j^*_k}{(\mu^* - j^*_k)^2}\right] / \left[\sum_k \frac{1}{(\mu^* - j^*_k)^2}\right] \tag{2.32}$$

$$\partial_\gamma\mu^* = \left[\sum_k \frac{A_k\partial_\gamma\mu^* + B_k - j^*_k/\gamma}{(\mu^* - j^*_k)^2}\right] / \left[\sum_k \frac{1}{(\mu^* - j^*_k)^2}\right] \tag{2.33}$$

$$\partial_\gamma\mu^*\left[\sum_k \frac{1 - A_k}{(\mu^* - j^*_k)^2}\right] = \left[\sum_k \frac{B_k - j^*_k\gamma}{(\mu^* - j^*_k)^2}\right] \tag{2.34}$$

which finally yields:

$$\partial_\gamma\mu^* = \left[\sum_k \frac{B_k - j^*_k/\gamma}{(\mu^* - j^*_k)^2}\right] / \left[\sum_k \frac{1 - A_k}{(\mu^* - j^*_k)^2}\right]. \tag{2.35}$$

Putting together eqns. (2.24) to (2.35) yields an explicit expression for the derivative of L_{test} with respect to γ, which is exactly the residual $Res^{opt}(\gamma)$ whose root gives the value of γ^{opt}.

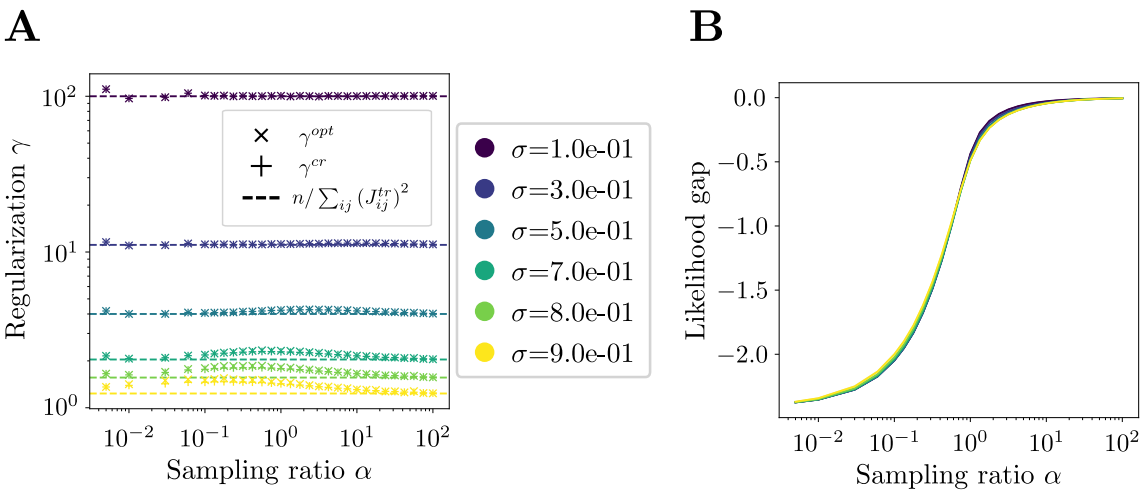


Figure 2.3: Gaussian Vectors Model with L_2 regularization. **A**: Evolution of the regularizations γ^{opt} and γ^{cross} as functions of the sampling ratio α for different values of the interaction dispersion σ, see eqn. (2.18). The theoretical prediction for γ^{cross}, represented here as a dashed line for each value of σ, is given in eqn. (2.20) and derived in Section 2.4. **B**: Evolution of the likelihood gap $\Delta L = L_{test}(\alpha, \gamma^{opt}(\alpha)) - L_{train}(\alpha = \infty, \gamma = 0)$ as a function of α for the same values of σ as panel **A**. As expected, this gap vanishes as α goes to infinity, meaning that the optimal inferred model (obtained with non-zero regularization) fits the data perfectly in the limit of infinite samples. While the gaps are identical between different values of the interaction strength, we were not able to determine the expression for this evolution.

Other types of underlying interactions

The empirical coincidence between γ^{opt} and γ^{cross} reported above extends to other choices of the coupling matrices. As an illustration we consider the case where the underlying interaction matrix $\boldsymbol{J}^{tr}$ is structured, instead of being randomly drawn. In particular, we present in Figure 2.4 two examples, and show that the presence of structure does not significantly alter our previous observations:

- in panel **A**, the interaction matrix is band-diagonal, meaning that the coefficients are given by

$$\forall\,(i,j)\ s.t.\ |(i-j) \mod n| < \frac{w}{2}\,, \quad J^{tr}_{ij} \sim \mathcal{G}\left(0, \frac{\sigma}{\sqrt{w}}\right)\,, \tag{2.36}$$

  where w is the width of the non-zero band, $\mathcal{G}$ is the Gaussian distribution, and $[n]$ represents the 'modulo n' operation. This means that sites are arranged on a ring, with interactions only between w nearest neighbors, and the value of those non-zero interactions are drawn randomly from a Gaussian distribution.

  This model can be related to the random Schrödinger operator in dimension 1, an object extensively studied in the context of Anderson localization, see Anderson (1958). As observed numerically by Casati et al. (1990) and later rigorously proved (see Bourgade (2018) for an overview), a phase transition can be observed when $w \sim \sqrt{n}$ between a regime (small w) where the eigenvectors of $\boldsymbol{J}^{tr}$ are localized *i.e.* decay exponentially with distance, and another where they are extended (large w).

  Our particular choice of scaling of the individual entries of those band matrices is such

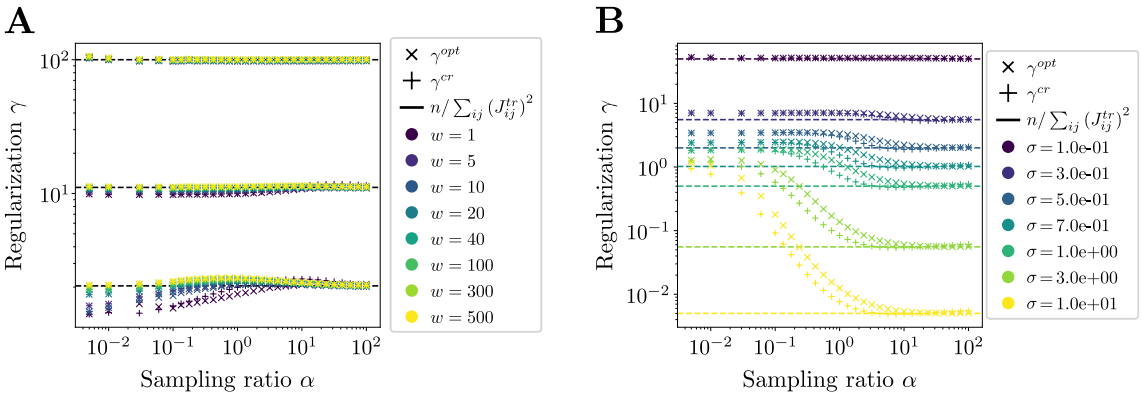


Figure 2.4: Evolution of the regularizations of interest for two different cases of structured interaction matrices $\boldsymbol{J}^{tr}$. **A**: case of a random band matrix. **B**: case of a deterministic, uniform 1-dimensional chain. In both cases, the observation that the crossing and optimal regularizations are of the same order of magnitude remains true, and so does the prediction for their value in the $\alpha \to \infty$ regime.

that $\frac{\sum_{i,j}(J^{tr}_{ij})^2}{n}$ remains constant, and so do the expected values of the regularizations of interest.

- in panel **B**, $\boldsymbol{J}^{tr}$ is a deterministic matrix corresponding to a unidimensional chain:

$$\forall\,(i,j)\,,\quad J^{tr}_{ij} = \begin{cases} 0 \text{ if } i=j \text{ or } |(i-j) \bmod n| > 1 \\ \sigma \text{ if } |(i-j) \bmod n| = 1 \end{cases}, \tag{2.37}$$

meaning that sites are again arranged on a ring, this time with fixed positive interactions between direct neighbors only. This particularly simple model does not exhibit any phase transition.

We find that changing the underlying model of interaction does not significantly impact the phenomenology that we previously observed for infinite-range Gaussian interactions: an optimal regularization still exists for all values of the sampling ratio α.

L_1 regularization

While the L_2 penalty is often used in practice, and encourages smoothness of the energy landscape, it is not the only possible choice. In many cases, it can be interesting to infer sparse interactions models, which is usually done by using an L_1 regularization: in a protein, amino acids which are very distant in the sequence can end up close in the folded structure, and therefore interact strongly so that one has to *a priori* allow interactions between all sites along the sequence; however, in three-dimensional space, each site is close only to a very small fractions, so that the inferred interaction matrix should be sparse. The inference procedure in this case is less straight-forward than for the L_2 case, and analytical solutions cannot be obtained in the general case. Instead, one relies on the so-called "Graphical Lasso" method Friedman et al. (2008), which iteratively solves Lasso problems for each column of the interaction matrix using coordinate descent Wright (2015) until convergence, implemented in Scikit-learn Pedregosa et al. (2011).

We show in Figure 2.5 that the behavior of the likelihoods remains qualitatively similar to what we observed in the case of L_2 regularization, despite the difference between the two noticeable regularizations being much higher than previously; similarly, the equality

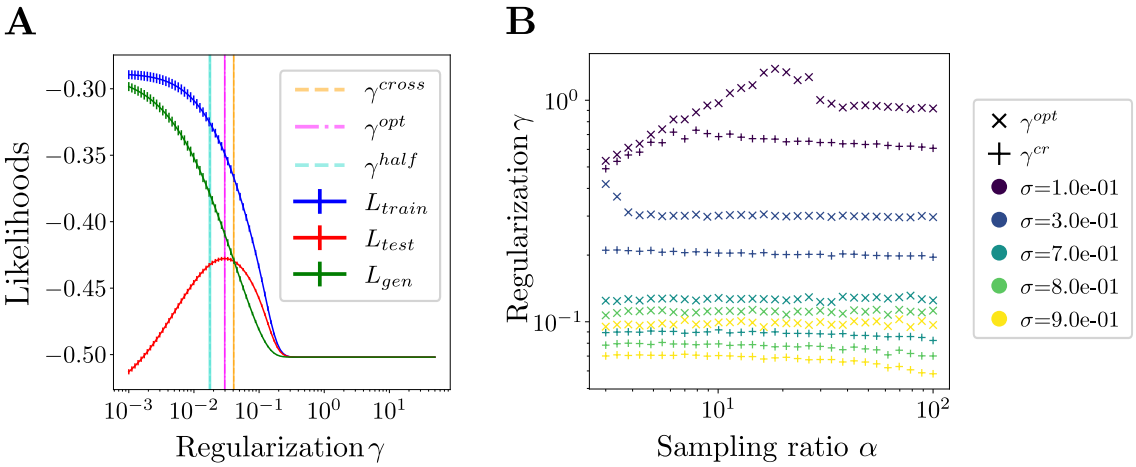


Figure 2.5: **A**: Typical evolution of the likelihoods as a function of the strength of the L_1 regularization. The existence of a finite optimal regularization, as well as the crossing between test and generated likelihoods, remains true as in the L_2 case. **B**: Evolution of the crossing and optimal regularizations as a function of the sampling ratio α. While the two noticeable regularizations are no longer equal, they remain of a similar order of magnitude.

(2.21) that was observed at γ^{half} remains close to being true, albeit less closely followed than in the L_2 case. A detailed analysis of this inference procedure could both shed light on the difference between the two, and give us a theoretical prediction for the optimal regularization in this regime, but this remains to be done in future work.

2.3.3 Potts Model

Generation of synthetic data and energy model

We now consider a discrete-valued graphical model, in which each (categorical) variables may take one out of q values. The energy of a configuration $\boldsymbol{x}$ is given by

$$E(\boldsymbol{x};\boldsymbol{h},\boldsymbol{J}) = -\sum_{i<j} J_{ij}(x_i,x_j) - \sum_i h_i(x_i)\ , \tag{2.38}$$

The local fields $\boldsymbol{h}$ and the couplings $\boldsymbol{J}$ are, respectively, q–dimensional vectors and $(q\times q)$–dimensional matrices. The corresponding partition function is

$$Z(\boldsymbol{h},\boldsymbol{J}) = \sum_{\{x_i=1,2,\ldots,q\}} e^{-E(\boldsymbol{x};\boldsymbol{h},\boldsymbol{J})} \tag{2.39}$$

We start by drawing the components of $\boldsymbol{h}^{tr}$ and $\boldsymbol{J}^{tr}$ that from Gaussian distributions of zero mean and standard deviations σ_h^2 and σ_J^2. All components of the h vectors and J matrices are chosen at random and independently from each other.

Next, each element of the Gaussian matrix J_{ij}^{tr} is multiplied by a connectivity indicator equal to 0 or 1, which identifies, respectively, the absence or the presence of an edge between the variables i and j in the coupling network. In practice, we choose this connectivity at random, following the prescription of the so-called Erdös-Rényi (ER) random graph ensemble. For each pair i,j of variables we chose to insert an edge in the interaction graph with probability d/n, and to have no connection with probability $1-d/n$; $d/n\times(n-1)$ is therefore the average degree of each variable in the connectivity graph.

In our simulations, we vary

- the size (number of variables), n; here $n = 25, 50, 100, 150$;
- the number of Potts states, q (here $q = 10, 20$);
- the probability d/n to include edges in the ER graph. Different values of d were tested only for $n = 25$, for which the computation were faster: $d = 1.25, 2.5, 7, 10$.

For each system, a number p of data point, ranging from 10^2 to 10^5 were generated by Markov Chain Monte Carlo sampling. Intuition about the sampling level can be obtained by comparing p with the number of parameters to infer from the data, $n \times q + \frac{1}{2} n(n-1) \times q^2$. The parameters defining the Gaussian distribution to generate fields and coupling are here kept constant as n varies: $\sigma_h^2 = 5$, $\sigma_J^2 = 1$.

Behaviours of the train, test, and generated log-likelihoods

Once the data are generated through Monte Carlo sampling of the Gibbs distribution associated to the energy (2.38) we infer the model parameters $h_i(x), J_{ij}(x, x')$ using two methods. The first one is the Pseudo-Likelihood Method (PLM), a non-Bayesian inference method that bypass the (intractable) computation of the partition function Z (Ravikumar et al., 2010; Ekeberg et al., 2013). The second one is the so-called Adaptive Cluster Expansion (ACE) algorithm, which recursively computes better and better approximations for the cross-entropy of the data (and $\log Z$) (Cocco and Monasson, 2011; Barton et al., 2016), combined with color compression (Rizzato et al., 2020).

The inference is done with a L_2-norm regularization on the couplings (intensity γ) and on the fields (intensity γ_h). We expect regularization to be much less needed for the fields, because single-site frequencies are much better sampled than pairwise frequencies. We therefore fix the ratio between the regularization of fields and couplings, setting $\gamma_h = \gamma/(10\, n)$, and vary γ.

In Figure 2.6, we show the average log-likelihoods (normalized by n) of the data in the training set, in the test set (same size as the training set) and the generated data set. Model parameters were inferred with the PLM procedure, and log-likelihoods (and the log partition function) were computed with the Annealed Importance Sampling method. For small regularization γ we observe a strong overfitting effect as expected, with similar values for L_{train} and L_{gen}, much above L_{test}. For intermediate regularization values, the test and generated log-likelihoods are similar as the number p of samples available for the inference increases, while the size n is kept fixed. This result is compatible with a weak dependence of γ^{cross} upon α, as found for the Gaussian Vectors Model. For large γ, L_{gen} may get smaller than L_{test}, a signature of very strong underfitting.

Dependence of optimal regularizations on system and data set sizes

We assess the quality of the inference through the Kullback-Leibler (KL) divergence of the inferred probability distribution from the ground-truth probability distribution,

$$D_{KL} = \sum_{\boldsymbol{x}} \frac{e^{-E(\boldsymbol{x};\boldsymbol{h}^*,\boldsymbol{J}^*)}}{Z(\boldsymbol{h}^*,\boldsymbol{J}^*)} \log \left[\frac{e^{-E(\boldsymbol{x};\boldsymbol{h}^*,\boldsymbol{J}^*)}}{Z(\boldsymbol{h}^*,\boldsymbol{J}^*)} \Big/ \frac{e^{-E(\boldsymbol{x};\boldsymbol{h}^{tr},\boldsymbol{J}^{tr})}}{Z(\boldsymbol{h}^{tr},\boldsymbol{J}^{tr})} \right] . \tag{2.40}$$

Again, we estimate the partition functions entering the definition above with Annealed Importance Sampling.

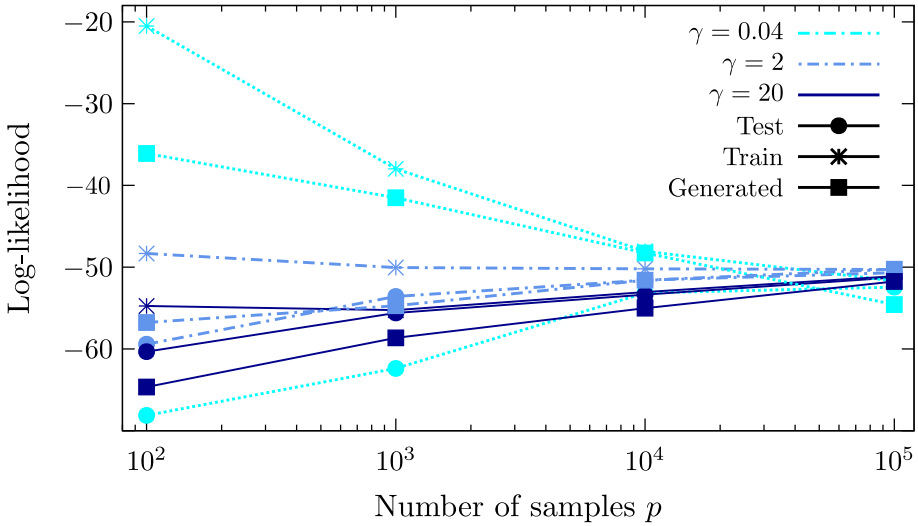


Figure 2.6: Average log-likelihoods of test (circular markers), train (cross markers) and generated (square markers) data vs. number p of samples for different regularization strengths γ (one color and line style for each). Results were averaged over 20,000 sequences for each reported value of γ and p. Parameters: $n = 50$, $q = 10$.

Dependence on the size n. We first study if and how the optimal regularization parameter γ changes when we the system size n is increased, while the average connectivity in the graph is fixed by choosing $p = 2.5/n$; we also fix the number of Potts states to $q = 10$. In Figure 2.7 we show the KL divergence for models inferred at different γ for various n and p. The optimal regularization γ^{opt} seems to be roughly equal to 0.5 in all the considered cases, independently of p (with some inaccuracy for very poor sampling, *i.e.* $p = 100$). We have also checked that this optimal value of γ does not seem to depend on q, by repeating the same numerical experiments for $q = 20$ Potts states with similar results, see Figure 2.8.

These two results are in very good agreement with the theoretical prediction reported in eqn. (2.20), that is, $\gamma^{opt} \simeq \gamma^{cross} \simeq \frac{1}{d} = 0.4$ for the parameters chosen in Figures 2.7 and 2.8. Indeed, in ER graphs, the average number of interacting neighbours is equal to d (on average), independently of n (and p). In addition, since each variable can take one out q symbol values, the number of variables j interacting with i in the sum at the denominator in eqn. (2.20) is independent of q.

Dependence on the structural connectivity of the interaction graph. We then study how the optimal regularization depends on the connectivity of the graph. For this reason we keep the graph size fixed ($n = 25$), and build different ER models with different densities varying d, see section 2.3.3. Once data are generated we infer the model parameters $\boldsymbol{h}^*, \boldsymbol{J}^*$ for different γ and sample sizes p. Results are reported in Figure 2.9, and show a clear dependence on the structural parameter d. We observe that the scaling factor is approximately inversely proportional to the number of neighbors on the interacting graph. This result is in excellent agreement with the outcome of the expected theoretical scaling reported in eqn. (2.20).

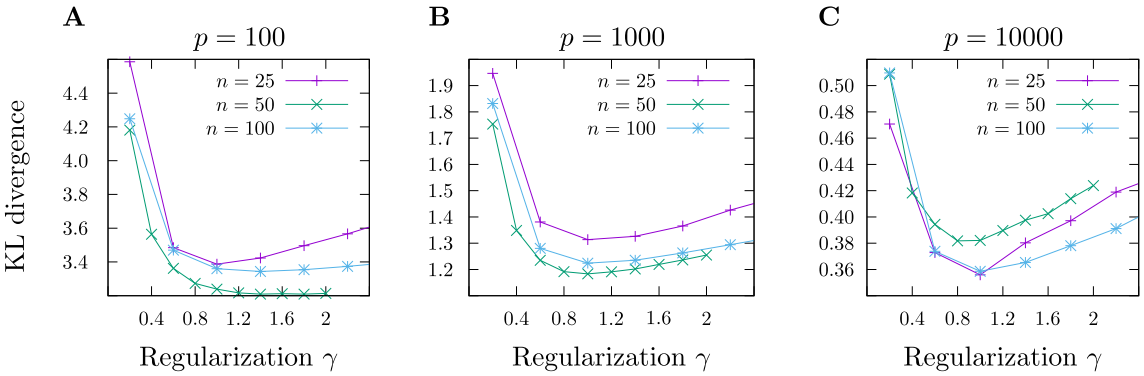


Figure 2.7: Kullback-Leibler (KL) divergence between the inferred models and the ground truth for different graph (n) and sampling (p) sizes as a function of the regularization on the couplings (γ). The y-axis was arbitrarily rescaled between the different curves to allow for easier comparison. Parameters: $d = 2.5$, $q = 10$.

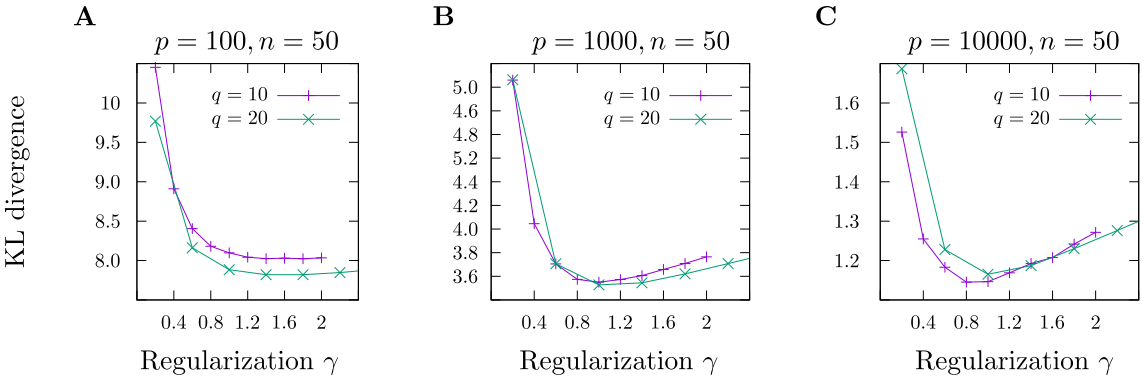


Figure 2.8: Kullback-Leibler (KL) divergence between the inferred models and the ground truth for different numbers q of Potts states and p of data points, as a function of the regularization on the couplings (γ). The y-axis was arbitrarily rescaled between the different curves to allow for easier comparison. Parameters: $d = 2.5$, $n = 50$.

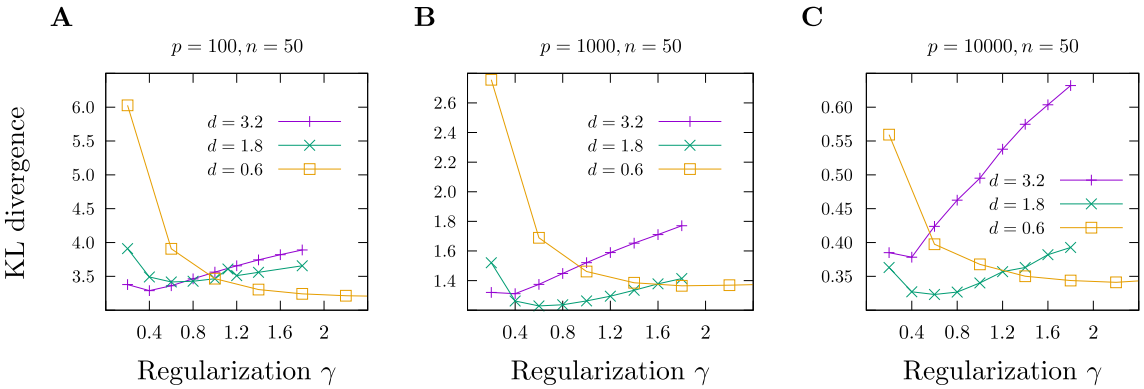


Figure 2.9: Kullback-Leibler (KL) divergence between the inferred models and the ground truth for different average number of edge per site (numbers reported in the panels, obtained by varying d), as a function of the regularization (γ) used during inference. The y-axis was arbitrarily rescaled between the different curves to allow for easier comparison. Parameters: $n = 25$, $q = 10$.

2.4 Analytical calculations at low and high sampling ratios

While finding the exact value for the regularization strengths of interest as functions of the model parameters is out of reach we show in this section how this calculation can be done in the case of the Gaussian Vectors Model for very low and high values of the sampling ratios.

2.4.1 Asymptotic behavior of γ^{cross}

The crossing regularization γ^{cross} is defined through

$$L_{test}(\gamma^{cross}) = L_{gen}(\gamma^{cross}) \ . \tag{2.41}$$

Replacing L_{gen} in the equation above with its expression in eqn. (2.16) and using the definitions (2.13,2.14) of the train and test log–likelihoods we obtain

$$\gamma^{cross} = \alpha \ \frac{L_{train}(\gamma^{cross}) - L_{test}(\gamma^{cross})}{\sum_{i,j} {J^*_{ij}}^2} = \alpha \ \frac{\sum_{i,j} J^*_{ij}(C^{emp}_{ij} - C^{tr}_{ij})}{\sum_{i,j} {J^*_{ij}}^2} \ . \tag{2.42}$$

$\alpha \to \infty$ regime

We derive below an asymptotic prediction for γ^{cross} in the large sampling regime $\alpha \to \infty$. We begin by considering the $\alpha \gg 1$ limit of the matrix-form MAP eqn. (2.8):

$$\boldsymbol{J}^* = \mu \boldsymbol{I} - (\boldsymbol{C}^{emp})^{-1}. \tag{2.43}$$

We consider the distribution of the empirical covariance matrix $\boldsymbol{C}^{emp}$ conditioned to the "true" correlation matrix $\boldsymbol{C}^{tr} = (\mu^{tr} - \boldsymbol{J}^{tr})^{-1}$, known as the Wishart distribution Wishart (1928), and defined for $p > n$ as

$$p_{\boldsymbol{J}^{tr}}(\boldsymbol{C}) \propto e^{n\frac{\alpha}{2}\mathcal{F}(\boldsymbol{C})} \ , \quad \mathcal{F}(\boldsymbol{C}) = \frac{\alpha - 1}{2} \log\det(\boldsymbol{C}) - \frac{\alpha}{2} \operatorname{Tr}\left((\mu^{tr} - \boldsymbol{J}^{tr})\boldsymbol{C}\right) \tag{2.44}$$

where we omit $\boldsymbol{C}$-independent normalization factor. For large α, we can perform a saddle-point approximation of this density around its maximum $\boldsymbol{C}^{tr}$:

$$p_{\boldsymbol{J}^{tr}}(\boldsymbol{C} = \boldsymbol{C}^{tr} + \Delta\boldsymbol{C}) \propto e^{n\frac{\alpha}{2}\Delta\boldsymbol{C}^\dagger \frac{\partial^2 \mathcal{F}}{\partial \boldsymbol{C} \partial \boldsymbol{C}}(\boldsymbol{C}^{tr})\Delta\boldsymbol{C}}. \tag{2.45}$$

A straightforward calculation leads to

$$\frac{\partial^2 \mathcal{F}}{\partial C_{i,j} \partial C_{a,b}}(C^{tr}) = \frac{\partial^2 \log\det \boldsymbol{C}}{\partial C_{i,j} \partial C_{a,b}}(\boldsymbol{C}^{tr}) = -(C^{tr})^{-1}_{a,i} \, (C^{tr})^{-1}_{b,j} \ . \tag{2.46}$$

We deduce from eqn. (2.45) that $(\boldsymbol{C}^{tr})^{-1} \times \Delta\boldsymbol{C} = \boldsymbol{U}/\sqrt{n\alpha}$, where $\boldsymbol{U}$ is distributed as an uncorrelated Gaussian matrix, whose entries have zero means and unit standard deviation. Therefore, using eqn. (2.43), we have

$$\boldsymbol{J}^* \simeq \mu \boldsymbol{I} - (\boldsymbol{C}^{tr} + \Delta\boldsymbol{C})^{-1} = \mu \boldsymbol{I} - \left(\boldsymbol{I} - \frac{\boldsymbol{U}}{\sqrt{\alpha n}}\right) {\boldsymbol{C}^{tr}}^{-1}. \tag{2.47}$$

This expression for the inferred coupling matrix can be inserted in eqn. (2.42) for γ^{cross}. Carrying out the averages over $\boldsymbol{U}$ appearing in $\boldsymbol{J}^*$ and $\boldsymbol{C}^{emp}$ we obtain

$$\gamma^{cross} = \frac{n}{\sum_{i,j}(J^*_{ij})^2} \overset{\alpha\to\infty}{\simeq} \frac{n}{\sum_{i,j}(J^{tr}_{ij})^2}. \tag{2.48}$$

The stronger the interactions in our underlying model, the weaker the regularization that needs to be applied during inference. One way of intuitively understanding this statement is that stronger interactions will *a priori* generate samples (and therefore MAP estimates) with less undesirable variance, and therefore require less smoothing from the regularization.

$\alpha \to 0$ regime

We now consider the case of very poor sampling. The lowest value of the sampling ratio, $\alpha = \frac{1}{n}$, is reached with a single sample $\boldsymbol{s}$ ($p = 1$). The empirical covariance matrix is then easily written as

$$\boldsymbol{C}^{emp} = \boldsymbol{s}\boldsymbol{s}^\dagger := n\,\boldsymbol{u}\boldsymbol{u}^\dagger. \tag{2.49}$$

One eigenvalue of $\boldsymbol{C}^{emp}$ is non-zero, and is fixed to n to enforce the spherical constraint[4]. In other words, the normalized vector $\boldsymbol{u} = \boldsymbol{s}/\sqrt{n}$ is the unique non-zero eigenvector of $\boldsymbol{C}^{emp}$.

Eigenvalues of $\boldsymbol{J}^*$. The inferred coupling matrix reads, according to eqns. (2.8) and (2.49),

$$\boldsymbol{J}^* = [j^*(n) - j^*(0)]\,\boldsymbol{u}\boldsymbol{u}^\dagger + j^*(0)\,\boldsymbol{I}\ , \tag{2.50}$$

where the eigenvalues $j^*(c^{emp})$ are given by eqn. (2.10). Using $\alpha = \frac{1}{n}$ and expanding in powers of $\frac{1}{n}$, we find

$$j^*(0) = -\frac{1}{n\gamma\mu^*} + \frac{2}{n^2\gamma^2\mu^{*3}} + O(n^{-3}). \tag{2.51}$$

$$j^*(n) = \frac{1}{2\gamma}\left[1 + \gamma\mu^* - \sqrt{(\gamma\mu^* - 1)^2 + \frac{4\gamma}{n}}\right] + O(n^{-3}). \tag{2.52}$$

The latter expression can be divided into two cases, depending on whether $\gamma\mu$ is larger or smaller than 1:

$$j^*(n) = \begin{cases} \mu^* - \dfrac{1}{n(1-\gamma\mu^*)} + \dfrac{\gamma}{n^2(1-\gamma\mu^*)^3} + O(n^{-3}) \text{ if } \gamma\mu < 1 \\ \dfrac{1}{\gamma} - \dfrac{1}{n(\gamma\mu^* - 1)} + \dfrac{\gamma}{n^2(\gamma\mu^* - 1)^3} + O(n^{-3}) \text{ if } \gamma\mu > 1. \end{cases} \tag{2.53}$$

which, together with the normalization condition

$$\frac{n-1}{\mu^* - j^*(0)} + \frac{1}{\mu^* - j^*(n)} = n, \tag{2.54}$$

yields that:

$$\mu^*(\gamma) = \begin{cases} \gamma^{-1/2} \text{ if } \gamma < 1 \\ 1 \text{ if } \gamma\mu^* > 1. \end{cases} \tag{2.55}$$

[4] In numerical experiments on finite size n, this constraint is enforced by hand, by rescaling the empirical covariance $\boldsymbol{C}^{emp}$ to have a trace exactly equal to n. Note that, in the $n \to \infty$ limit and for $\sigma < 1$, this rescaling is not necessary. For σ larger than 1, however, the norm of $\boldsymbol{s}$ fluctuates strongly, as $|\boldsymbol{s}|^2$ follows a chi-square distribution.

Expression for γ^{cross}. We then express the terms appearing in the expression of γ^{cross}, see eqn. (2.42), in terms of the eigenvalues $j^*(0), j^*(n)$:

$$\sum_{i,j}(J^*_{ij})^2 = j^*(n)^2 + (n-1)\, j^*(0)^2 \ , \tag{2.56}$$

$$\sum_{i,j} J^*_{ij}\, C^{emp}_{ij} = n\ [j^*(n) - j^*(0)] + n\, j^*(0) \ , \tag{2.57}$$

$$\sum_{i,j} J^*_{ij}\, C^{tr}_{ij} = n\ [j^*(n) - j^*(0)]\ \theta + n\, j^*(0) \ , \tag{2.58}$$

where we introduced the matrix element

$$\theta = \frac{1}{n}\sum_{i,j} \boldsymbol{u}_i \boldsymbol{C}^{tr}_{ij} \boldsymbol{u}_j \ . \tag{2.59}$$

Let us consider this quantity in more details. On average over the sample $\boldsymbol{s}(=\sqrt{n}\boldsymbol{u})$, we have:

$$\langle \boldsymbol{u}_i \boldsymbol{u}_j \rangle = \frac{1}{n}\boldsymbol{C}^{tr}_{ij}, \tag{2.60}$$

and thus

$$\langle \theta \rangle = \frac{1}{n^2}\sum_{i,j} \boldsymbol{C}^{tr\,2}_{ij} = \frac{1}{n^2}\sum_k c^{tr\,2}_k. \tag{2.61}$$

Generally, due to the constraint that $\sum_k c^{tr}_k = n$, we find that $\langle\theta\rangle$ is bounded from below by $1/n$ (when all eigenvalues of $\boldsymbol{C}^{tr}$ are equal to 1), and from above by 1 (when a single eigenvalue of $\boldsymbol{C}^{tr}$ is equal to n, and all the other eigenvalues are equal to 0). It should be noted that, while the result $\langle\theta\rangle > 1/n$ is true on average, the value of θ for an individual sample can be arbitrarily close to 0. This possibility will be discussed below.

Analytical expressions for the average value of θ can be obtained in the case of random quenched interactions considered in Section 2.3.1 by explicitly integrating over the semi-circle eigenvalue distribution, with the results

$$\langle\theta\rangle = \begin{cases} \frac{1}{n(1-\sigma^2)} & \text{if} \quad \sigma < \sigma_c = 1 \ , \\ \left(1 - \frac{1}{\sigma}\right)^2 & \text{if} \qquad \sigma > \sigma_c \ , \end{cases} \tag{2.62}$$

see Figure 2.10A. The last equation comes from the fact that, when σ is larger than 1, the sum in eqn. (2.61) is dominated by the single macroscopic eigenvalue of $\boldsymbol{C}^{tr}$, see eqn. (2.19). For the rescaled samples we used in practice, the average value of θ still goes to 1 as σ increases, but with a gap closer to $\sim \sigma^{-1/2}$.

Let us now summarize the different cases that can be met, see Figure 2.10B:

- If σ is below 1, the system is in a disordered phase, and strong regularization is needed. We find that
  - if $\theta > 1/n$, the crossing regularization is

$$\gamma^{cross} = \frac{n\theta}{n\theta - 1}, \tag{2.63}$$

    which is larger than 1. This corresponds to a situation where the sample is slightly informative, and strong regularization is necessary to avoid overfitting.

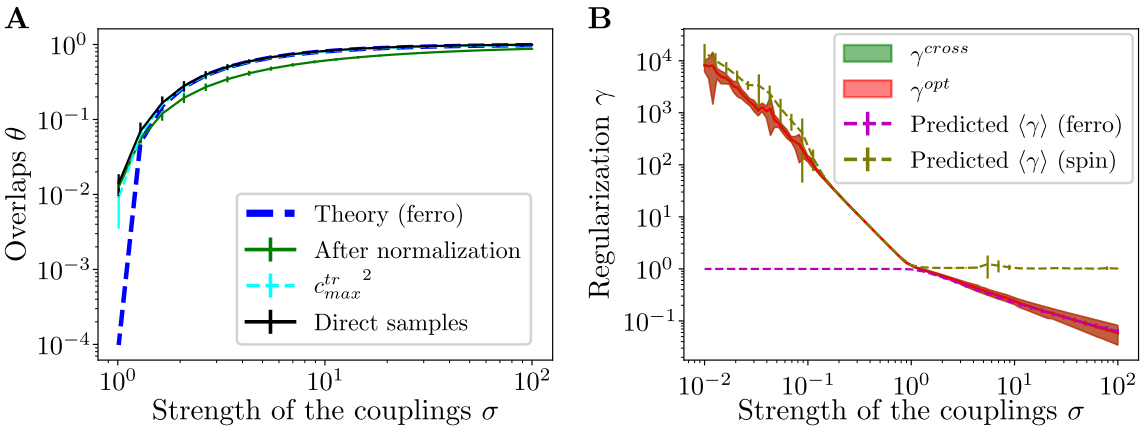


Figure 2.10: Properties of the inference in the low sampling regime $\alpha = 1/n$ and for the random quenched coupling model. **A**: Value of the overlap θ as a function of the scale σ of the interactions in the ferromagnetic regime $\sigma > 1$, see theoretical prediction for $\langle\theta\rangle$ for non rescaled samples in eq. (2.62). For normalized samples the overlap converges towards 1 more slowly. **B**: Comparison between the values of γ^{opt} and γ^{cross} found numerically and the predictions in eqns, (2.63) and (2.64), applied to the empirical distribution of "rescaled samples" overlaps. In both panels error bars represent the variations across 10 choices of the true underlying interaction matrix of the θ, and γ is averaged over 100 random draws from the Gaussian model distributioh.

  - if $\theta < 1/n$, the two likelihoods never cross, and the optimal regularization appears to be infinite. This corresponds to a situation in which the randomly drawn sample is counter-informative, so that the null answer is better than taking it into account.

- If σ is above 1, the system is in the ferromagnetic phase, so that a single sample conveys significant information about the entire distribution. In that case, we find that for a given (rescaled) sample the crossing regularization is given by

$$\gamma^{cross} = (1-\theta)^2, \tag{2.64}$$

  which vanishes when $\sigma \to \infty$.

In all cases where σ is either very small or very large, the optimal regularization varies strongly from sample to sample.

2.4.2 Asymptotic behavior of γ^{opt} for $\alpha \to 0$

While the $\alpha \to \infty$ limit of the optimal regularization is hard to obtain (in particular, because the test likelihood's derivative with respect to γ vanishes uniformly), the computation of γ^{cross} can be carried out in the low ratio regime, $\alpha = 1/n$.

We start from the definition of γ^{opt}:

$$\frac{\partial L^{test}}{\partial \gamma}(\gamma^{opt}) = 0 = \frac{\partial}{\partial \gamma}\left[\frac{1}{2}\sum_{i,j} J^*_{ij} C^{tr}_{ij} - \log\, Z(\boldsymbol{J}^*)\right] \tag{2.65}$$

From eqns. (2.7) and (2.50), we have

$$\log Z(\boldsymbol{J}^*) = \frac{n}{2}\mu^* - \frac{1}{2}\left[(n-1)\log(\mu^* - j^*(0)) + \log(\mu^* - j^*(n))\right] \tag{2.66}$$

so that

$$\frac{\partial \log Z(\boldsymbol{J}^*)}{\partial \gamma} = \frac{n-1}{2}\frac{\partial_\gamma j^*(0)}{\mu^* - j^*(0)} + \frac{1}{2}\frac{\partial_\gamma j^*(n)}{\mu^* - j^*(n)} \ . \tag{2.67}$$

In addition, differentiating eqn. (2.58) we get

$$\frac{\partial}{\partial \gamma}\sum_{i,j} J^*_{ij} C^{tr}_{ij} = n\left[\frac{\partial j^*(n)}{\partial \gamma} - \frac{\partial j^*(0)}{\partial \gamma}\right]\theta + n\frac{\partial j^*(0)}{\partial \gamma} \tag{2.68}$$

We now need to evaluate the derivatives $\partial_\gamma j^*(0)$ and $\partial_\gamma j^*(n)$. From eqns. (2.51) and (2.55), at the first order in n, we have

$$\frac{\partial j^*(0)}{\partial \gamma} = \frac{1}{n\gamma^2\mu^*} + \frac{\partial_\gamma \mu^*}{n\gamma\mu^{*2}} = \begin{cases} \frac{1}{n\gamma^2} & \text{if} \quad \gamma > 1 \ , \\ \frac{1}{2n\gamma^{3/2}} & \text{if} \quad \gamma < 1 \ . \end{cases} \tag{2.69}$$

Similarly, eqns. (2.52) and (2.55) yield

$$\frac{\partial j^*(n)}{\partial \gamma} = \begin{cases} -\dfrac{1}{\gamma^2} & \text{if } \gamma > 1 \ , \qquad (2.70a) \\ -\dfrac{1}{2\gamma^{3/2}} & \text{if } \gamma < 1 \ . \qquad (2.70b) \end{cases}$$

We may now conclude our calculation of γ^{opt}:

- If $\gamma > 1$,

$$\frac{\partial L^{test}}{\partial \gamma} = \frac{1}{2\gamma^2}(1 - n\theta) + \frac{1}{2\gamma^2(\gamma - 1)}. \tag{2.71}$$

  and therefore this derivative vanishes for

$$\gamma^{opt} = \frac{n\theta}{n\theta - 1}, \tag{2.72}$$

  which is the same result as found from the γ^{cross} computation in equation (2.63).

- If $\gamma < 1$,

$$\frac{\partial L^{test}}{\partial \gamma} = \frac{n}{4\gamma^{3/2}}\left[(1-\theta) - \sqrt{\gamma}\right] \ , \tag{2.73}$$

  whose root is given by

$$\gamma^{opt} = (1-\theta)^2 \ , \tag{2.74}$$

  in full agreement with the result shown in eqn. (2.64).

Therefore, the analytical expressions of γ^{opt} and γ^{cross} coincide in the undersampled regime (single sample), which provides further support to our conjecture that the values of those two regularizations are equal or very close, as suggested by numerical experiments. Unfortunately, the computation of γ^{opt} in the oversampled regime ($\alpha \to \infty$) is more complicated, and we were not able to prove that its value converges to the limit found for γ^{cross} in eqn. (2.48).

2.5 Conclusion

In this work we provided both analytical and numerical evidence for the optimal value of a L_2 penalty term in the likelihood used for Maximum A Posteriori inference of graphical models. In addition to showing that a non-zero optimal regularization always exists, we find a remarkable empirical coincidence between two optimality criteria: the maximization of the test log-likelihood, and the condition that test and generated likelihoods are equal, a natural requirement for a generative model, see Figure 2.1. This equality suggests that, while weaker regularizations might give the impression of higher quality generated data (through higher generated likelihoods), stronger regularizations should actually be employed to achieve the best possible model, and the perceived increase in generated likelihood is actually a form of overfitting.

Analytical expressions for the crossing and optimal regularizations could be obtained in the limiting regimes of poor or good sampling. In the latter case, we obtain an explicit expression for the optimal regularization strengths in terms of the average inverse squared couplings between the variables, see eqn. (2.20). This prediction remains remarkably accurate over a wide range of parameter value, and even for case of categorical variables (Potts model), while it was established analytically in the case of the Gaussian multivariate model. This result suggest that our study could also be applied to other interesting classes of models, such as Restricted Boltzmann Machines, an extension of Ising/Potts models in which multi-body interactions can be introduced. More generally, it has been known for a long time that Neural Networks benefit from regularization, with extensive research being led on the exact regularization scheme to apply for different tasks (see for example Wan et al. (2013); Zaremba et al. (2015); Louizos et al. (2018); Haarnoja et al. (2018); Bartlett et al. (2021)); all approaches exhibit some form of "bias-variance trade off", *i.e.* a phenomenon in which increasing the strength of the regularization reduces the variance of the estimator (*e.g.* by increasing the smoothness of the solutions) but in doing so biases the inference towards a particular subset of solutions; because of this, an optimal value of the regularization exists that balances those two effects, very similarly to what we observed in our simplistic model.

In terms of modeling protein from sequence data our results suggest that the optimal γ should neither be proportional to $\frac{p}{n}$ nor to q, as proposed in previous works (Ekeberg et al., 2014; Hopf et al., 2017), but is related to the inverse sum of the squared couplings incoming onto residues, see eqn. (2.20). In particular, our prediction is that the optimal value for γ scales inversely proportional to the number of interacting neighbors on the dependency graph. However, some caution must be brought to this conclusion. The sample size p is not clearly defined for real proteins. The presence of phylogenetic correlations between sequences make the assumption of independent data points only approximate at best. In practice the choices $\gamma = 0.01\frac{p}{n}$ (Ekeberg et al., 2014) and $\gamma = 0.01\, q$ (Hopf et al., 2017) are qualitatively similar when the number of sequences exceed the protein length by a factor 20, which is not unreasonable for a substantial number of protein families.

Last of all, let us recall that we focused in this work only on Maximum A Posteriori inference, which can be seen as the null temperature limit of Bayesian inference. It is natural to wonder whether our result hold for when sampling the posterior probability at inverse temperature β:

$$p_\beta(\boldsymbol{J}) \propto e^{-\beta\left[\frac{\gamma}{4}Tr(\boldsymbol{J}^2)-\frac{\alpha}{2}Tr(\boldsymbol{J}\boldsymbol{C}^{emp})+\alpha\log Z(\boldsymbol{J})\right]}\,. \tag{2.75}$$

While an in-depth study of the different sampling strategies is out of the scope of this work (see Rubinstein and Kroese (2016) for a general overview), we report below numerical and analytical preliminary steps aiming at characterizing this posterior distributions.

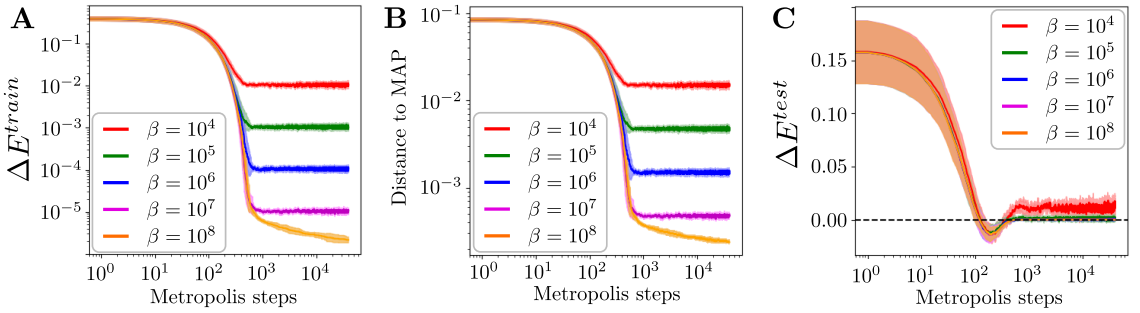


Figure 2.11: Evolution of the train energy, distance to MAP estimator and test energy as a function of the number of Metropolis steps for different values of the temperature. The energies are given relative to the ones of the MAP. For low temperatures and long enough times, the sampled solutions have very close energies to the MAP estimator. At intermediate times, the test energy of the sampled solutions can get lower than the one of the MAP. Higher temperature allow the system to stay in states of higher energy, which are further from the MAP. Figure obtained with $n = 20$, $\alpha = 5$, $\sigma = 0.5$, $\gamma = 5$ (larger than the optimal regularization $\gamma^{opt} = 1/\sigma^2 = 4$).

We performed some preliminary experiments using a simple Metropolis-Hastings algorithm Metropolis and S. Ulam (1949) which consists in starting from a random point in the distribution, proposing a small modification and accepting it with probability $p = \min(1, \exp(-\beta \Delta E))$ depending on the associated change in energy, ΔE. In our case, we start from a symmetric Gaussian matrix in which all the entries above the diagonal are independent and have the same mean and variance as the MAP estimator[5], and the modifications we propose are the addition of small amplitude, sparse, Gaussian matrices. Since increasing the temperature (hence decreasing β) can be seen as a way of letting the system explore areas of higher energy, the matrices sampled at higher temperatures will be further away from the MAP solution, which we illustrate in Figure 2.11**A** and **B** respectively. While the energy used for sampling is computed using the empirical covariance matrix $\boldsymbol{C}^{emp}$, it is also interesting to consider the evolution of a "test" energy, computed using the true covariance matrix $\boldsymbol{C}^{tr}$, which will help quantify the generalization property of these solutions. While at long time scales the test energy converges to a value very close to the one of the MAP estimator, there exists an intermediate regime in which the sampled matrices achieve better test energy than the MAP estimator, as seen in Figure 2.11**C**. Notice, however, that the values of the inverse temperature β considered in the simulations are large compared to the canonical inverse temperature, n, defined in the posterior probability over $\boldsymbol{J}$, see eqn. (2.5). The results reported above therefore imply that weak fluctuations of the posterior do not modify the properties of the MAP estimator.

[5]This initial choice only affects convergence time, as the Metropolis sampling procedure loses information on the initial conditions after a transient regime.

Chapter 3

Computational models of neurobiology

Abstract

In order to better situate the following work within its broader context, we present here an introduction to computational neuroscience. After a brief history of the study of individual neurons, detailing how the evolution of the membrane potential constitutes the main support for information representation within the brain. We then focus on the "mesoscopic" scale of neuroscience, at which the interactions between a large number of neurons allows for the emergence of collective phenomena, similar to the ones studied in the field of Statistical Physics, which in particular are used to process information. We define the concept of "cognitive manifold", a subset of the possible states of the neural population (usually of low dimension) which are effectively observed during behavior, which will be central to the theory of Recurrent Neural Integrators we consider in Chapter 7. Finally, we present a few examples of neural circuit models which are relevant not only to behavioral neuroscience, but also to cognitive science and consequently to Machine Learning: vision, hearing, spatial navigation and abstract reasoning.

3.1 Individual neuron models

One of the first major results in modern Neuroscience was the discovery of the neurons as the basic building blocks of nervous systems, made possible by the invention of a novel tissue staining technique by Camillo Golgi and its further refinement by Santiago Ramon y Cajal, which allowed staining of cells in their entirety. This particularity proved to be critical to the analysis of the connectivity within the brain, as it allowed visualization of the filamentary extensions of neurons, which were not observable using previously developed techniques due to their small size and relative transparency. Both scientists were awarded in 1906 the Nobel Prize in Physiology and Medicine for their work.

Although many families of neurons exist, with different physiological and information processing properties, most neurons are comprised of three parts: the cell body or **soma**, which contains the nucleus and therefore most of the RNA material and protein synthesis activity; the dendrites, which receive information from neighboring neurons and relay it to the soma; the axon, which allows for passing information to downstream neurons through the **synapses** that can be found at its termination and connect to the **dendrites** of other neurons. Those synaptic terminals contain vesicles, which respond to the arrival of an electrical signal from the soma (such as the action potentials, see next paragraphs) by fusing with the membrane of the axon and releasing the neurotransmitters they contain in the **synaptic cleft**, which will then bind to specific receptors on the dendritic membrane of the downstream neurons. Some neurotransmitters, such as glutamate, will increase activity of the downstream neuron, creating an *excitatory* synapse), while others such as GABA will decrease it and create an **inhibitory synapse**.

The way information is represented at the level of a single neuron is through the value of the **membrane potential**: at any given time, the concentrations of positive and negative ions inside the neuron and in the extracellular fluid that surrounds it are *a priori* different. This difference in concentration is made possible by the fact that the neuron membrane is a bilayer lipidic film that is mostly impermeable to charged molecules, and in practice acts as a capacitor. The conductance of this capacitor can be modulated through a wide variety of mechanisms, mostly related to the presence of **ion channels** that allow ionic flow through the lipid membrane in certain conditions: the channels can be opened or closed based on a number of different variables such as the value of the membrane potential, or the concentration of neurotransmitters in the extracellular cleft (*e.g.* for channels in the dendritic receptors, allowing the flow of information between neurons via electrical to chemical transduction).

The interplay between these different mechanisms has been studied in great details, most notably by Hodgkin and Huxley (1952), who performed experiments on the giant axon of *Doryteuthis pealeii* squids, an axon so large that it allowed easy experimentation with voltage clamp electrodes and whose function had already been characterized by Young (1938). Their study provides a detailed account of how the presence of a wide variety of different membrane conductances allows for the apparition of **action potentials**, brief and transient depolarization of the membrane followed by a period of hyper-polarization, whose shape is highly stereotyped for a given neuron.

While conductance models are extremely precise and predict individual neuron properties with great accuracy, the large number of parameters to infer and of differential equations to solve make them ill-adapted to many practical situations. In that case, simpler models have to be used, such as the **Integrate-and-Fire** models, originally introduced by Lapicque (1907) and further refined by Brette and Gerstner (2005). This model was developed very early in the history of modern Neuroscience, only a few decades after the

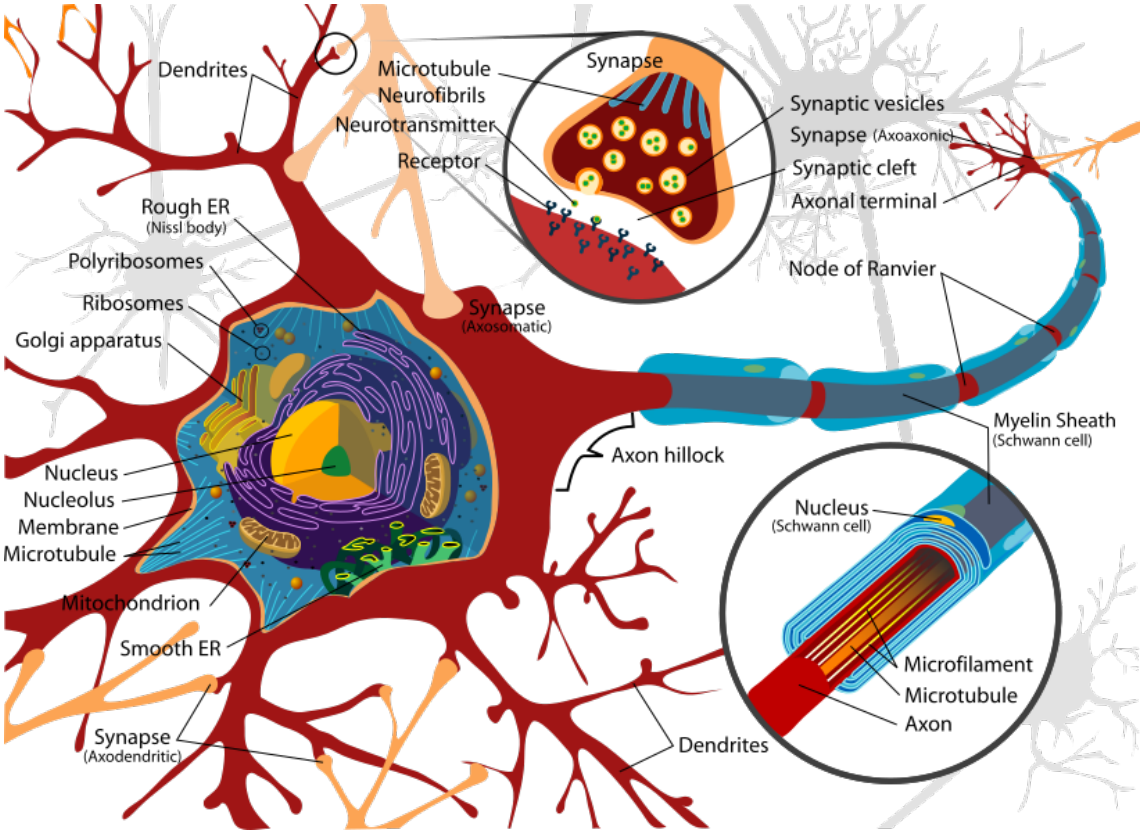


Figure 3.1: Illustration of the different components of a neuron. The body of the neuron, or *soma*, centralizes most of the metabolic functions; the lipidic membrane separating the intra and extracellular media can act as a capacitor, and acquire a potential; by opening or closing ion channels on the surface of the membrane, the potential acquires non-trivial dynamics, responsible for information propagation along the axons as "action potentials", short and highly stereo-typical shapes of membrane potential evolution; when the action potentials reach the axon terminal, neurotransmitters are released in the synaptic cleft in-between the axon of the upstream neuron and the dendrites of the downstream neuron; depending on the type of synapse, these neurotransmitters will impact differently the electrochemical properties of the downstream neuron, allowing information transmission between neurons; combining a large number of neurons, and tuning their synapses correctly, many computations can be realized. This diagram is part of the public domain.

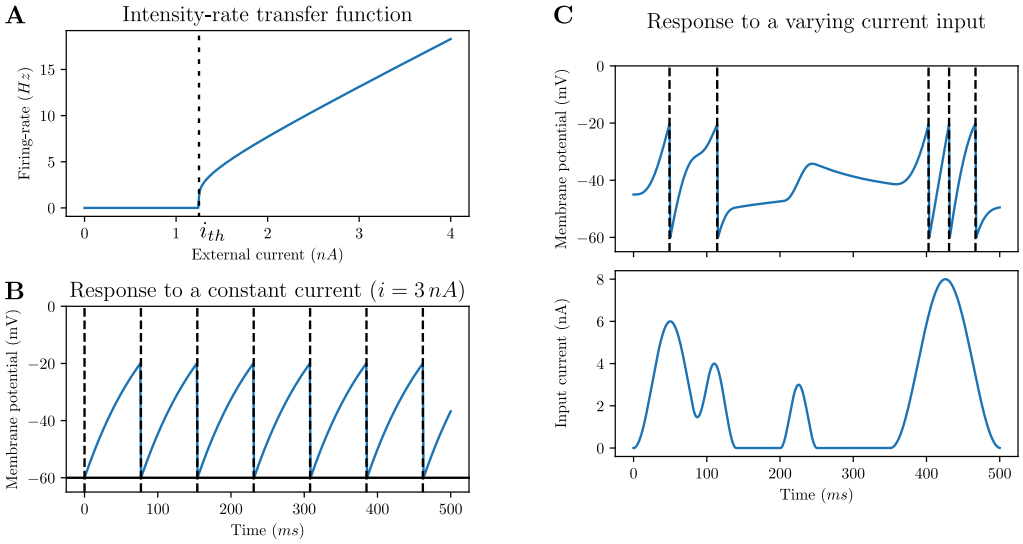


Figure 3.2: Summary of the properties of a Leaky Integrate and Fire neuron model. **A**: transfer function mapping the input current i to the firing-rate, as computed in equation 3.3. When the input current is below a certain value i_{th}, the neuron never fires, as the membrane potential never reaches the threshold for emitting an action potential. Above that threshold, the firing-rate is an almost linear function of i, so that this transfer function can be approximated by $max(i - i_{th}, 0)$. **B**: response of the membrane potential to a constant input current, showing a regular firing pattern where action potentials are emitted at frequency $r(i)$ (dashed vertical lines). **C**: response of the membrane potential to a varying input current $i(t)$. The input is chosen to model the arrival of different action potentials from afferent neurons, resulting in different responses (no firing, firing of one or several action potentials in response).

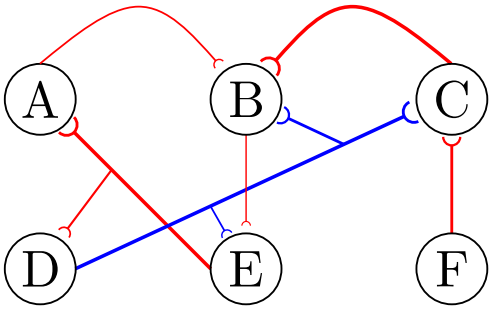
discovery of neurons, and far before the precise mechanisms of action potential generation were uncovered; whenever those details are not critical to the phenomenology, Integrate-and-fire models provide an easy to simulate and analytically tractable alternative to the Hodgkin model. The assumption of this model is that the membrane potential v evolves continuously below a certain **firing threshold** V_{th}; when the threshold is reached, the neuron will fire an action potential (represented as a Dirac delta), and the potential will be reset to a lower value E_{reset}; finally, the sub-threshold evolution of the potential is modeled as an exponential decay at rate τ, which depends on the membrane conductance, towards a resting potential E_0 and subject to a forcing by an external current i:

$$\tau \frac{dv}{dt} = (E_0 - v) + R\,i. \tag{3.1}$$

In that model, one can easily compute a **transfer function**, mapping the value of the external current i to the frequency at which the neuron emits an action potential: assuming i is constant[1], and $v(t = 0) = E_{reset}$, we can solve the sub-threshold dynamics as:

$$v(t) = E_0 + R\,i + (E_{reset} - E_0 - R\,i)\,e^{(-t/\tau)}, \tag{3.2}$$

[1]In the following, we will consider varying input currents $i(t)$; for simplicity, however, we can assume this input current to be constant for a short duration, equal to i_t from t to $t + 1$, so that $r(t)$ is also constant on this short duration.



$$\boldsymbol{W} = W_{i \leftarrow j} = \begin{pmatrix} 0 & 0 & 0 & 0 & 10 & 0 \\ 5 & 0 & 10 & -8 & 0 & 0 \\ 0 & 0 & 0 & -10 & 0 & 8 \\ 0 & 0 & 0 & 0 & 6 & 0 \\ 0 & 4 & 0 & -5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Figure 3.3: Illustration of the mapping between a network of neurons, connected through synapses of different strengths, and the corresponding weight-matrix $\boldsymbol{W}$.

which reaches the firing threshold in a time t^{isi}, the interspike interval, equal to:

$$t^{isi}(i) = \tau \ln \left(\frac{R\, i + E_0 - E_{reset}}{R\, i + E_0 - E_{th}} \right), \tag{3.3}$$

so that action potentials are emitted at a rate $r(i) = 1/t^{isi}(i)$.

In practice, even this simplified model can become too complex to simulate when considering large numbers of neurons; in that case, one can make use of **rate-based models**, which will be the basis of most of the work in this dissertation. Rate models are defined in discrete time, and assume that the state of the neuron at each time-step is the **firing-rate**, in Hertz, at which it emits action potentials. These rates can be thought of either as the average across a mesoscopic timescale, justifying the move from continuous to discrete time in the model, or across a large enough assembly of neurons. The dynamics of the neuron state can then be summarized by defining only its **transfer function** $r(i)$, which will allow for a very succinct and computationally efficient description of dynamics at the macroscopic scale (see Equation 3.4). Unfortunately, the use of such models entails a loss of detail with respect to the conductance-based models described previously, and some subtle phenomena such as Spike-Timing Dependent plasticity (Markram et al., 1997) can not be reproduced. Another approach consists in switching to the McCulloch-Pitts model, described in more details in Section 4.1, which considers the neuron to always be in a binary state, active or inactive, which also evolves in discrete time.

3.2 Neural circuits models

Until now, we have assumed that our individual neurons received input currents i, but did not give any details on how such a current is provided to the cell. The simplest way possible, which was historically used to establish the models we described earlier, was to consider neurons isolated from the rest of the brain and provide the input current through an electrode, controlled by the experiment. In practice, however, this current comes from other neurons through synapses, and those connections are extremely important to understand the function being implemented by an assembly of neurons. In practice, we will describe the couplings between neurons through a **weight-matrix** $\boldsymbol{W}$, whose components at index (i, j) describes the strength of synaptic coupling from neuron j to neuron i, as illustrated in Figure 3.3.

If we now consider the state of a population of n rate-based neurons as a vector $\boldsymbol{r}$, the dynamics of evolution of this vector can be written as:

$$\boldsymbol{r}_{t+1} = f(\boldsymbol{W}\boldsymbol{r}_t), \tag{3.4}$$

where f is the transfer function of the neurons in the assembly[2]. Such a structure of alternating a linear mapping with a pointwise non-linearity is the basis of Deep Learning, and in the following we will investigate numerous special cases of such structures.

Constraints on the weight-matrix Without any additional assumption, this framework of representing the synaptic couplings through a n–dimensional matrix allows for arbitrary, not biologically relevant connectivity graphs. First, each neuron is allowed to project synapses onto all neurons, which is not realistic since all of those have to be located near the extremity of the (unique) axon; to better represent this constraint, limits on the number of non-zero weights per column of $\boldsymbol{W}$, or even realistic models of axon growth and synapse creation such as the small-world networks (Bassett and Bullmore, 2006) could be introduced. Second, neurons are allowed to have outgoing connections of both signs, meaning that they can act as excitatory towards some neurons and inhibitory towards others; such a behavior is in contradiction with the so-called **Dale's Law**, which states that all outgoing synapses from a given neuron must be of the same type, resulting in fixed-sign columns in the weight-matrix $\boldsymbol{W}$. Finally, in order to be able to perform certain computations and analogies with Statistical Physics, most notably when trying to reproduce the spike correlations by using an Ising model, $\boldsymbol{W}$ is assumed to be symmetric. In our analysis of Recurrent Neural Integrators of Chapter 7, we will relax the hypothesis of symmetry, which as we will see introduces some theoretical complications which can be solved by considering the **Singular Value Decomposition** of $\boldsymbol{W}$; we will also show that the introduction of Dale's Law into a Gradient Descent procedure can be done straightforwardly, and only marginally modifies the behavior of the system, an approach similar to the one of Song et al. (2016). We will not consider additional structural constraints.

3.3 Neural integrators and stable manifolds dynamics

In Chapter 7, we will consider models of non-linear Recurrent Neural Networks tasked with performing the integral of several input signals in parallel. The motivation for this study lies in the fact that such networks have been found to be associated with several important cognitive tasks, such as eye fixation in-between saccades (Cannon et al., 1983; Cannon and Robinson, 1987; Arnold and Robinson, 1991; Seung, 1996; Arnold and Robinson, 1997), evidence integration (Wong and Wang, 2006; Wong et al., 2007; Ganguli et al., 2008), and motor control (Gallego et al., 2017; Feulner and Clopath, 2021).

Let us for now focus on the case of oculomotor control. In order to be able to hold a **memory** of the current value of eye position, the associated neural circuit has to construct a continuous manifold of fixed-points, within which each position corresponds to a value of the angular position of the eye. Additionally, one could expect this manifold of fixed-points to be stable with respect to the dynamics of the network, so that small perturbation in the state of the network will be corrected; such a manifold is often referred to as a **line attractor** in the case where the network stores the value of a single variable. Seung (1996) showed that in the case of a linear recurrent network, the existence of this line attractor is related to the existence of an eigenvector with eigenvalue 1 in the weight-matrix (in which case the stable manifold is given by that eigenvector) and proposed a solution for approximating this dynamics in non-linear networks. In Chapter 7, we extend this approach to stable manifolds of arbitrary dimension, allowing for integration of any number of input signals within a single network, and provide an abstract energy function whose minima implement this dynamic for arbitrary non-linearities.

[2]This function could be different for each neuron without any major consequences.

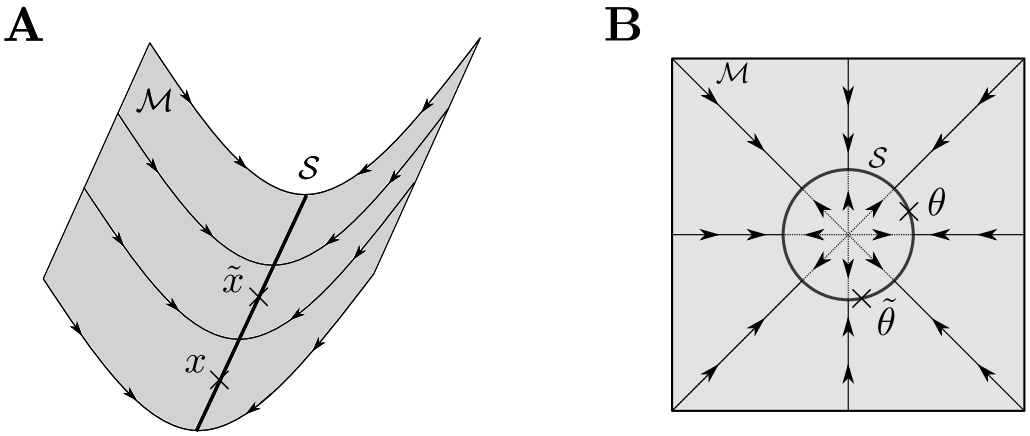


Figure 3.4: Two examples of attractor manifolds. In both cases, we represent in gray the manifolds $\mathcal{M}$ of all possible network states (defined by the weight-matrix $\boldsymbol{W}$, the neuron non-linearities, and the initial state), and as a bold line the manifold of stable fixed-points $\mathcal{S}$. The arrows represent the direction of evolution of the system under the network dynamics. **A** represents the case of a **line attractor**, in which all states converge to a 1–dimensional line; such an attractor can be used to store the value of a single scalar x as the position of the system along the line $\mathcal{S}$. **B** represents the case in which the manifold $\mathcal{S}$ forms a closed loop; such an attractor naturally exhibits a periodicity, making it particularly suitable for storing an angle θ.

While for oculomotor stabilization it is desirable to have all points in the line attractor be fixed-points of the dynamics, in other cases (for example, when evidence integration is required) it might be interesting to impose a particular dynamics within this line attractor: if the input to the network corresponds to the current "evidence" towards a particular decision, it might be interesting to introduce a form of **decay** $\gamma < 1$ in the integral, so that the evidence is integrated only within a finite time-window. If the position x along the line attractor corresponds to the current value of the integral, we expect that the network will be at position $x_{t+1} = \gamma x_t$ after one step of the network dynamics. In the case $\gamma = 1$, this corresponds to a fixed-point, but any γ between 0 and 1 will correspond to a relaxation towards a fixed-point at coordinate $x = 0$, meaning that in the absence of evidence, the network comes back to its default state. This idea is a special case of the "computation through latent variable dynamics" framework (Sussillo, 2014), which postulates that the brain performs computations, and more generally cognitive tasks, through the temporal dynamics of firing; considering the brain as a Recurrent Neural Network (see Section 5.3), the dynamical system that describes the evolution of the internal state under the influence of the external inputs entirely characterizes the function of the network. This idea is deeply related to the work presented in Chapter 7.

This idea of dynamics within a stable manifold can be further extended to accommodate more exotic dynamics. One such example is the one of ring integrators (Kim et al., 2017), hypothesized to represent the orientation of the head in behaving animals. Assuming that the head rotation is constrained to a plane, which is usually behaviorally relevant, and that angular velocity can be obtained from sensory neurons (*e.g.* through vestibular or visual inputs), one could be interested to construct a neural network in which the stable manifold has a ring structure, so that states corresponding to angles close to $-\pi$ and $+\pi$ are close in that manifold, and angles are integrated with a periodicity of 2π as two angles differing by 2π correspond to the same position in space. One way to con-

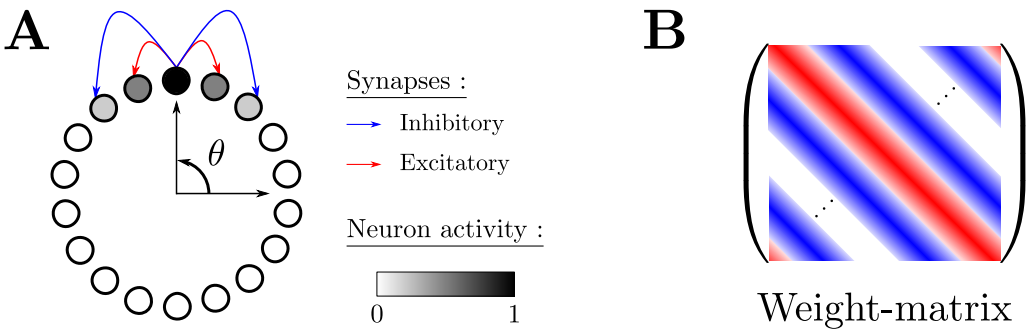


Figure 3.5: Schematic representation of a ring attractor (**A**) and its connectivity matrix $\boldsymbol{W}$ (**B**). The neurons in the network are regularly organized on a circle, with excitatory connections (red) towards their closest neighbors, and inhibitory (blue) towards others; we represent the weight-matrix as translation invariant, since such a solution ensures the existence of periodic solutions. The stable states $\boldsymbol{s}_\theta$ of the associated dynamics are "bumps" of activity centered on each neuron (the neuron θ is maximally activated, and its neighbors are less and less activated the further from θ they are. These bumps of activity can be translated into the corresponding angle by a single layer network, and external inputs (not represented here) from a velocity sensor $\delta\theta$ can be projected to all neurons in the network in such a way that the next internal state at which the network stabilizes is $\boldsymbol{s}_{\theta+\delta\theta}$.

struct such a model is presented in Figure 3.5, where neurons are arranged in a circle, and connections between neighboring neurons are excitatory while connections towards more distant neurons are inhibitory. In this connectivity structure, fixed-points of the dynamics correspond to a localized "bump" of activity (a few consecutive neurons along the ring are active), and the coordinate along the manifold of stable point is simply the location of the center of that bump[3]. Given the circular organization of neurons, that position naturally exhibits a periodicity, and assuming that connections toward the head-velocity neurons are correctly tuned the position within the attractor will faithfully encode the position of the head in physical space.

3.4 Examples of biological neural circuits

While the microscopic behavior of individual neurons has been studied and mostly understood for a relatively long time, their organization into macroscopic "circuits", responsible for performing specific information processing tasks, remains an active area of research to this day. This section will aim at giving a brief overview of the different domains of investigation, and providing reference to some relevant literature.

3.4.1 Visual Pathway

We will begin this introduction by introducing the Visual information processing system, as it was one of the main motivations for the emergence of Artificial Neural Networks, both the Perceptron 4.1 algorithm and the more recent Convolutional Neural Networks 5.1. This

[3]This is a special case of a cyclic matrix $\boldsymbol{W}$, in which each row is obtained by a lateral translation of the previous one; the eigenvectors of these matrices are linear combinations of the discrete sine and cosine function, and the particular value of the weights used here is such that the eigenstates are localized (*i.e.* are non-zero only on a subset of all neurons).

system has been extensively studied since the 1930s, with Nobel Prizes in Physiology and Medicine awarded to Haldan K. Hartline, Ragnar Granit and George Wald in 1967, and to David H. Hubel and Torsten Wiesel in 1981.

The first step in the visual system is the retina, which has been shown to contain two main types of photo-receptors: the rods, which are sensitive to light intensity starting at a few individual photons (Baylor et al., 1979) and show little sensitivity to the wave-lengths of those photons, are useful primarily for black-and-white vision, in particular at low luminosity; the cones, which require a much higher number of incoming photons to activate, are much more sensitive to wave-length, and in the humans are divided into three populations, which respond maximally to red, green and blue light respectively (Brown and Wald, 1964), and are especially important for color vision. Both types of photo-receptors translate light into membrane potentials using ion channels that can be opened or closed by specific proteins, called opsins, which change configuration upon interaction with incoming photons. The axons of those receptors then connect to horizontal and bipolar retinal cells, which in turn project to retinal ganglion cells. Those ganglion cells are the first in the visual pathway to encode information through action potentials instead of continuous membrane potentials, and some of them have been shown to have easily characterizable **receptive fields** consisting of two concentric circles, one in which the neuron is excited and one where it is inhibited; depending on which circle is the smallest, the neuron is classified as "on-center" or "off-center". Those ganglion cells then project to the central nervous system, more precisely the Visual Cortex, an area organized in several layers denoted V1 to V6[4], which has been studied by Hubel and Wiesel (1962) in anesthetized cats. They found that the receptive field of these neurons were different from those of the ganglion neurons, reacting to either "simple" or "complex" features of the visual signals, such as orientation or movement, and proposed the hypothesis that cortical neurons **integrated** information from their afferent neurons to construct more complex representations, a hypothesis which has profoundly shaped the understanding of information processing in nervous systems.

3.4.2 Sound

The first step in the audio processing pathway relies on sensory neurons, the **hair cells**, located within the organ of Corti in the cochlea of mammals. Those neurons contain mechanically gated ion channels, allowing them to transform information about movement of the basilar membrane, caused by incoming sound waves, into variations of electric potential. We show in Figure 3.6 an example of cochleogram, *i.e.* a visual representation of the membrane potential of an assembly of hair cells in response to an incoming sound. Each of the neurons is sensitive to a particular range of frequencies, which is used to organize the neurons in the plot.

The electrical outputs of the hair cells are then projected onto the Auditory Cortex, in which a wide range of functions is performed, such as pitch perception, sound localization and speech recognition. The study of these topics, broadly referred to as psychoacoustics, remains particularly challenging, in particular from the point of view of Artificial Intelligence. Systems to analyze and produce sound remain experimentally difficult to construct, despite recent advances in practical implementations (see Ling et al. (2015) for a review) as well as theoretical understanding (Eggermont, 2001; McDermott and Simoncelli, 2011; Kell et al., 2018).

[4] Although the nomenclature suggests a purely feedforward system in which one area provides inputs to the following, which is usually how this system is modeled in Artificial Neural Networks, backward connections are also present.

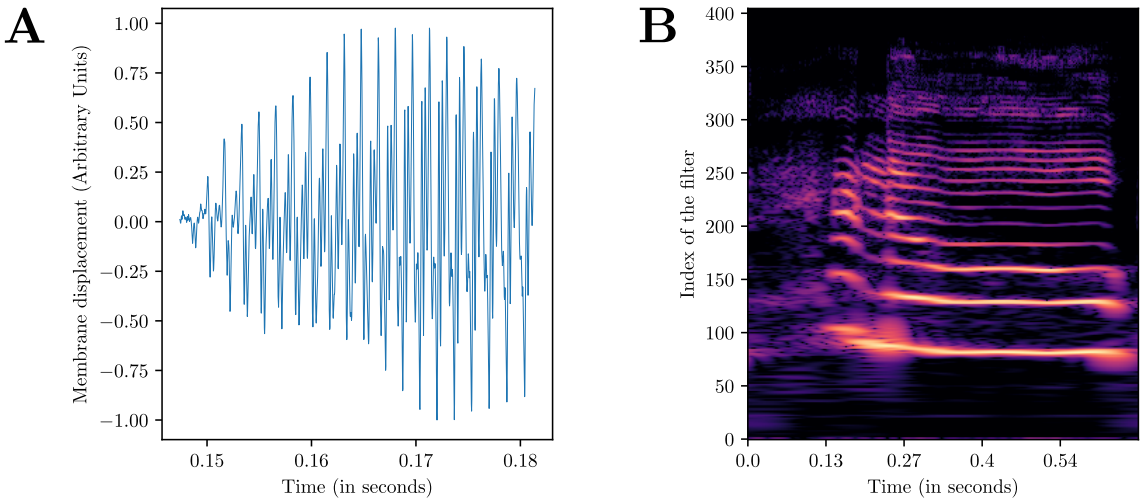


Figure 3.6: Illustration of the preprocessing step used to transform the raw displacement signal obtained from a microphone, see panel **A**, into its cochleogram, see panel **B**. This illustration was realized using a self-recorded waveform of the word "Hello", processed into human-perception tuned cochleograms using the python implementation of Josh McDermott's Matlab code, available on GitHub.

Another example worth mentioning here is the one of the zebra finch songbird, in particular its ability to communicate via learned vocalizations (Zeigler and Marler, 2004) through a well-identified and thoroughly studied "song-circuit" (Hahnloser et al., 2002; Hamaguchi and Mooney, 2012; Markowitz et al., 2015; Okubo et al., 2015; Ono et al., 2016). While this organism has seen its genome entirely sequenced (Warren et al., 2010), understanding the precise mechanisms at play in this circuit remains extremely challenging. It should be noted that song-generation mechanisms are relevant far beyond the domain of psychoacoustics as they are an example of sequence generation and storage, two concepts that can be applied for example to spatial navigation (generating the sequence of muscle inputs necessary to walk, or storing the optimal trajectory between two positions in an environment).

3.4.3 Spatial navigation

Cognitive maps Tolman (1948) introduced a fundamental concept for the understanding of spatial navigation, the one of a **cognitive map**, mental representation of the spatial structure of an environment. Such a representation is thought to be the basis for "wayfinding" behaviors in both mammals and humans (Prescott, 1996; Golledge, 2003), most notably Path Integration (McNaughton et al., 2006; Etienne and Jeffery, 2004), the task of keeping track of displacement across a sequence of movements which will be the focus of our work presented in Chapter 8. By design, this definition of cognitive maps is extremely loose, and makes no assumption about the structure of the representation (first-person or top-down view, euclidean or topological metric, goal-specific or "general").

The question of how such maps can be constructed remains relevant to this day (Epstein et al., 2017), as it is known that their elaboration relies on the merging of many, qualitatively different modalities (Maaswinkel and Whishaw, 1999): some of them are **proprioceptive** (*e.g.*head-direction signal (Taube et al., 1990), velocity signal coming from speed cells (Kropff et al., 2015), reafferent copies of performed actions (Iacoboni et al., 2001), memories of past trajectories (Cooper et al., 2001)), in the sense that they

are generated by the agent itself, while others are **allocentric** and instead correspond to measurements performed on the environment (*e.g.*visual (Etienne et al., 1996), auditory (Rossier et al., 2000), olfactory (Deschênes et al., 2012)), and "fusing" these sensors is not a trivial task. Understanding the contributions of those two types of information is a major challenge in psychology, as it is *a priori* impossible to decouple them in real world experiments (a movement in physical space always has the same impact on environmental cues). However, developments in Virtual Reality techniques (Campbell et al., 2018) have recently made it possible, and Chen et al. (2019) have performed such an experiment: a mouse is moving on a rotating ball, and a screen projects a movie of a virtual environment in which the mouse's position is updated by its movement on the ball; the experimenters are then able to control how the real movement impacts the virtual one (in particular, making it so that a real movement produces either a larger or smaller than expected virtual displacement), and to assess the influence of such changes on both place and grid cells (by plotting the firing fields in the "real-world" and "virtual world" coordinates, which are no longer identical). Their results suggest that some cells are mostly constructed from visual cues as changes in the movement gains do not impact them, while other rely more on self-motion cues (respectively, place and grid cells, see next paragraph for details). In Chapter 8, we argue that constructing direct-inverse models of the environment (predicting the next state of allocentric sensors given their current state and the proprioceptive signals) yields a promising approach to performing this fusion, in particular when combined with Path Integration as an incentive to propagate information between time-steps.

Place and grid cells A widely accepted possible neural substrate for cognitive maps is the Hippocampus (O'Keefe and Nadel, 1978), as it has been found to contain a specific type of neurons, referred to as **place cells**, which become active when an agent is located at a specific position in the environment (Humphreys et al., 1998). One particularly interesting property of place cells is that, in different environments, the same neuron can encode for different positions, a phenomenon known as remapping (Fyhn et al., 2007), and inferring the environment from neuronal activity is a relevant task (Posani et al., 2017). Recently, Battista and Monasson (2020) derived a theoretical analysis of the capacity-resolution trade off in the storage of multiple semi-continuous maps with continuous attractors.

On top of place cells, Moser et al. (2008) found evidence for another type of neurons displaying spatially-structured firing fields, which are called the **grid cells**. For those cells, instead of observing spikes only in a specific region of space, firing happens at numerous, regularly spaced positions, arranged in a regular triangular lattice. Such an organization could be beneficial to represent distance, without any particular relationship to orientation. Another interpretation for the properties of grid cells, coming from studies on the "Successor Representation" of Reinforcement Learning, (see Gershman et al. (2012), Stachenfeld et al. (2014), Momennejad et al. (2017), Gershman (2018)), is that they could allow for efficient hierarchical representations of policies used for spatial navigation. We illustrate the spatial structure of firing of those two types of neurons in Figure 3.7.

3.4.4 Abstract reasoning

One of the theorized advantages of using grid cells for Spatial Navigation is that such patterns could provide a *transferable* model: if such firing fields are indeed constructed mostly from self-motion signals, they will *a priori* function in a never-before-seen environment, and allow very efficient "few-shots" learning. In other words, the underlying **structure** of euclidean space has been represented by the Hippocampus, and can be used to efficiently understand a new **instance** of that structure. It has been experimentally shown that the

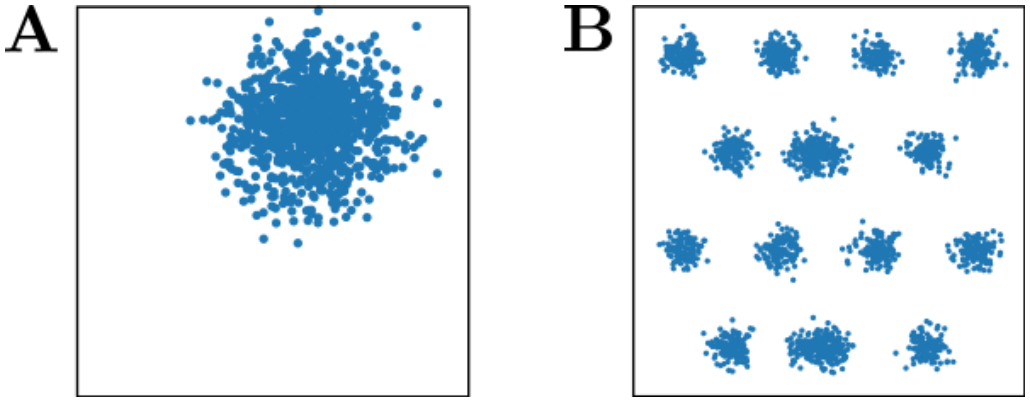


Figure 3.7: Schematic representation of the firing fields for a place cell (**A**) and a grid cell (**B**). The black square represents the boundaries of the environment, and each dot represents the position at which a spike was emitted. Both firing fields are spatially structured, but the grid cell firing fields additionally present spatial periodicity.

Hippocampus is indeed relevant for high-level cognition tasks (Aronov et al., 2017; Doeller et al., 2010), and theoretical models have been proposed that show how to generalize this form of structure abstraction to relational memory (Whittington et al., 2020).

Another brain region heavily involved in decision-making is the Prefrontal Cortex (PFC), which has been shown by Romo et al. (1999) to be involved in working memory, *i.e.* allows information to be conserved on a short time-scale relevant to performing an action. The PFC is also involved in the processing of external rewards (Liu et al., 2007), making it a likely candidate for the neural substrate of Reinforcement Learning (more details in Chapter 6), as well as in the processing of behaviorally relevant contextual cues (Mante et al., 2013; Freedman et al., 2003). Finally, PFC is also related to behavioral adaptation (Passingham, 1993; Wise and Murray, 2000; Feulner and Clopath, 2021) and the so-called "learning-to-learn" phenomenon (Yang et al., 2019), in which an agent is able to learn a new task faster if it has already been trained to perform similar tasks, an idea related to the one of Multi-Goal Reinforcement Learning in which the agent is expected to construct a Universal Value Function Approximator (Schaul et al., 2015), allowing generalization of learned behaviors to new contexts (*e.g.* navigating towards states that were never used as goals during training).

Chapter 4

Deep Learning I: the Multi-Layer Perceptron

Abstract

Now that the basics of computational modeling of neurobiological processes were introduced in Chapter 3, we will focus on a particularly simple but historically relevant model, the Perceptron, introduced in the 1950s by Rosenblatt (1958) for the study of the salamander retina. Notably, an algorithm exists for "training" such a model in order to represent a given "binary predicate" on a set of data, under certain hypothesis, which we present in the following. We then explain how to relax the hypotheses of this model, and extend it until we arrive to a general mathematical definition of Machine Learning, before providing an example of back-propagation for optimization of a 2–layers network. We introduce the concept of overfitting in the context of Deep Learning, and finally consider the idea of **transfer learning** as a way to benefit from high-quality representations.

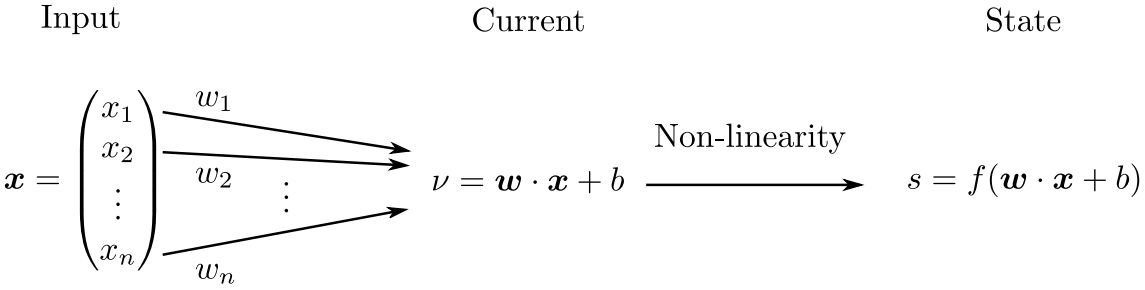


Figure 4.1: Schematic representation of a general artificial neuron. In all generality, the inputs to the neuron are represented as a vector $\boldsymbol{x}$; these inputs are then mapped to a scalar **current** through an affine transformation, parametrized by a vector $\boldsymbol{w}$ and a scalar b (which will be omitted in the following, see footnote 3); the state is finally obtained from the pre-activation current through application of a non-linearity. This type of input-state mapping is designed to mimic rate-based models of neurons mentioned in Chapter 3, and MCP neuron corresponds to a special case where the activation function is the "sign" function.

4.1 The McCulloch-Pitts Neuron

The objects that we are going to consider in this section were originally constructed in an effort to understand the salamander retina, and more precisely how an input image (*i.e.* a matrix of "gray-levels" between 0 and 1) could be processed into a logical statement (*e.g.* the image has at least one black pixel, the image contains a dog, etc...).

The McCulloch-Pitts (MCP) neurons, first proposal to tackle this task, take a vector as input, requiring to flatten the image before being able to process it[1]. Therefore, the most general setting consists not in image predicates, but in predicates over an observation space[2].

The computation of an MCP neuron can be summarized as follows:

- the input vector $\boldsymbol{x}$ is affinely mapped to a scalar **current** $\nu = \boldsymbol{w} \cdot \boldsymbol{x} + b$;[3]
- the sign of the current is returned, making the output being ± 1.

The MCP neurons are therefore a predicate on images, parametrized by a set $\theta = (\boldsymbol{w}, b)$ of parameters. An important question to ask is the following: what is the set of predicates that can be represented, *i.e.* have the same value at any point of the observation space, by a McCulloch Pitts neuron? A simple geometric consideration will prove useful here: the predicate for a given point in the observation vector space is exactly its position with respect to the so-called *Decision Hyperplane* H_θ. Therefore, the set of representable predicates is exactly the "linearly separable" ones, such that there exists a hyperplane separating all "True" observations from the "False" ones.

[1] This particular point, although irrelevant in non-spatially structured problems, is crucial when it comes to understanding the dramatic improvements that Convolutional Networks (see Chapter 5.1) brought to Computer Vision.

[2] Strictly speaking, this space would need to be an Inner Product Space. However, in practice, any subset of R^n with either the canonical dot product or any Kernel function will do.

[3] In practice, the bias b is often omitted through the following experimental trick: one dimension is added to the vectors $\boldsymbol{x}$ and $\boldsymbol{w}$, so that $\tilde{\boldsymbol{x}} = (x_0, \ldots, x_n, 1)$ and $\tilde{\boldsymbol{w}} = (w_0, \ldots, w_n, b)$. Since the additional component is the same for all inputs $\boldsymbol{x}$ and equal to 1, we have $\tilde{\boldsymbol{w}} \cdot \tilde{\boldsymbol{x}} = \boldsymbol{w} \cdot \boldsymbol{x} + b$, and with this rewriting we only have one vector parameter to consider.

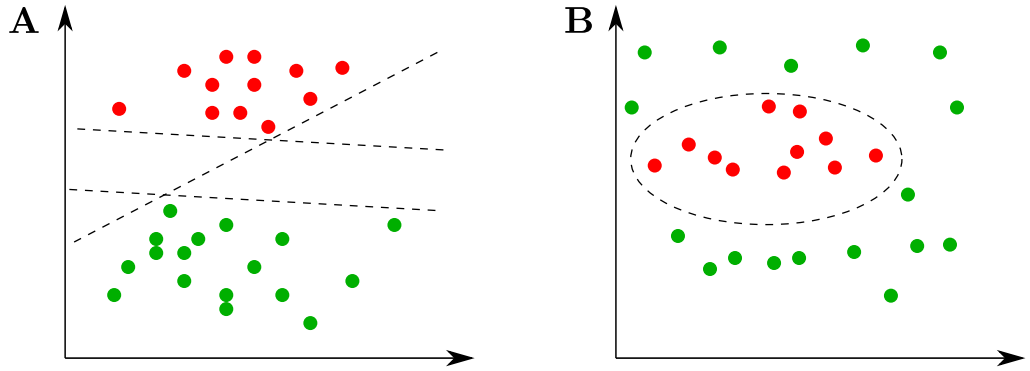


Figure 4.2: Illustration of linear separability on a two-dimensional input space. **A**: A non-empty set of affine hyperplanes separate the red and green points: the MCP neurons with these decision hyperplanes correctly represent the predicate "the point belongs in the green set". **B**: No hyperplane exists that separates the two sets: a non-linear decision boundary is necessary.

4.2 The Perceptron learning rule

For now, we have only discussed the representation properties of the MCP neurons, but we did not consider the problem of learning: how does one set the parameters θ of a neuron to encode a specific predicate?

With a single binary neuron, the only possible problem is binary classification, and it is solved by the perceptron algorithm. This algorithm has a very natural geometric interpretation, which we illustrate in Figure 4.3: starting from $\boldsymbol{w}_0 = 0$, the decision boundary of the neuron is iteratively moved by choosing one example $(\boldsymbol{x}, y)$ in the training set, computing the output of the neuron with its current weights $\overline{y} = \boldsymbol{w}_t \cdot \boldsymbol{x}$ and updating the boundary as

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \frac{1}{2}(y - \overline{y})\boldsymbol{x} \tag{4.1}$$

until all the examples are correctly classified. Variants of the training algorithm might choose the example at random from the entire training set, or make sure that all examples are used at least once before using any example a second time. Alternatively, the Perceptron algorithm (and its multi-layer extensions) can be used in an **online** setting, in which the network sequentially updates itself based on incoming samples (Saad and Solla, 1995; Saad, 1999). These modifications have little to no impact on the solutions and convergence in the general case.

One important property of this algorithm is that in the case of linearly separable inputs, it is guaranteed to converge in finite time. To prove this, let us first denote as $\boldsymbol{w}^*$ a vector of $\mathbb{R}^n$, that we choose of unit norm, that separates the two classes, and γ the associated **margin**, *i.e.* a strictly positive number such that:

$$\forall\, (\boldsymbol{x}, y) \in \text{Training set}, \quad y(\boldsymbol{w}^* \cdot \boldsymbol{x}) > \gamma. \tag{4.2}$$

We can then show that, at any step:

$$\begin{aligned} \boldsymbol{w}_{t+1} \cdot \boldsymbol{w}* &= (\boldsymbol{w}_t + \frac{1}{2}(y - \overline{y})\boldsymbol{x}) \cdot \boldsymbol{w}* \\ &= \boldsymbol{w}_t \cdot \boldsymbol{w}* + \frac{1}{2}(y - \overline{y})\boldsymbol{x} \cdot \boldsymbol{w}* \\ &> \boldsymbol{w}_t \cdot \boldsymbol{w}* + \gamma \quad \text{if } y \neq \overline{y} \end{aligned} \tag{4.3}$$

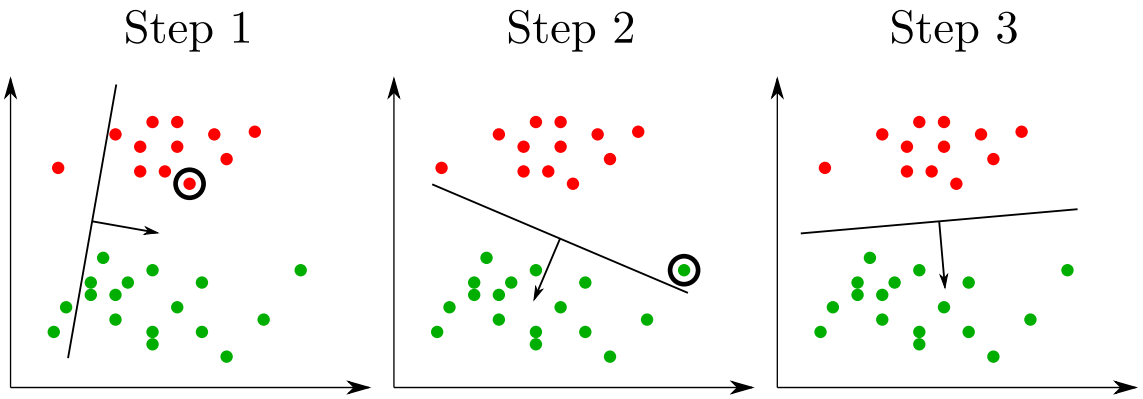


Figure 4.3: Illustration of the Perceptron learning algorithm. At each step, one of the incorrectly classified examples is chosen, and the decision boundary is moved in the appropriate direction to better classify it. Training stops when all examples are correctly classified, which is guaranteed to happen in finite time.

By induction, after k updates of the weight vector, we have $\boldsymbol{w}_k \cdot \boldsymbol{w}* > k\gamma$. Since $||\boldsymbol{w}^*|| = 1$, we obtain a lower bound on the norm of $\boldsymbol{w}_k$:

$$||\boldsymbol{w}_k|| = ||\boldsymbol{w}_k||\,||\boldsymbol{w}^*|| \geq \boldsymbol{w}_k \cdot \boldsymbol{w}* > k\gamma. \tag{4.4}$$

Then, we have that

$$\begin{aligned} ||\boldsymbol{w}_k||^2 &= ||(\boldsymbol{w}_{k-1} + \frac{1}{2}(y - \overline{y})\boldsymbol{x})||^2 \\ &= ||\boldsymbol{w}_{k-1}||^2 + ||\boldsymbol{x}||^2 + \underbrace{(y - \overline{y})\boldsymbol{x} \cdot \boldsymbol{w}_k}_{\leq 0 \text{ since } y \neq \overline{y}} \\ &\leq ||\boldsymbol{w}_{k-1}||^2 + ||\boldsymbol{x}||^2 \\ &\leq ||\boldsymbol{w}_{k-1}||^2 + R^2, \end{aligned} \tag{4.5}$$

where $R = \max ||\boldsymbol{x}||$. Therefore, through another simple inductive proof, we find that

$$kR^2 \geq ||\boldsymbol{w}_k||^2 > k^2\gamma^2. \tag{4.6}$$

These two equalities can only be satisfied as long as $k < R^2\gamma^2 = K$, meaning that the algorithm will perform at most K updates before all points are correctly classified. Interestingly, this bound is independent of the choice of $\boldsymbol{w}^*$ at the beginning of the proof, and only the value of γ remains. Therefore, one can choose $\boldsymbol{w}^*$ in order to maximize the margin γ to obtain the tightest convergence bound possible. The largest this margin, the faster the convergence, but as long as it remains positive the convergence time will remain finite. If the margin becomes negative, *i.e.* the problem becomes non-linearly separable, other methods can be adopted, such as the Support Vector Machines that we will now describe.

4.3 Support Vector Machines

While it is reasonable that an algorithm such as the Perceptron does not achieve good performance on non-linearly separable data, the absence of guarantees on its convergence is a major practical limitation. One way to understand this phenomenon is that the Perceptron updates are only concerned with correctly classifying the current example they are

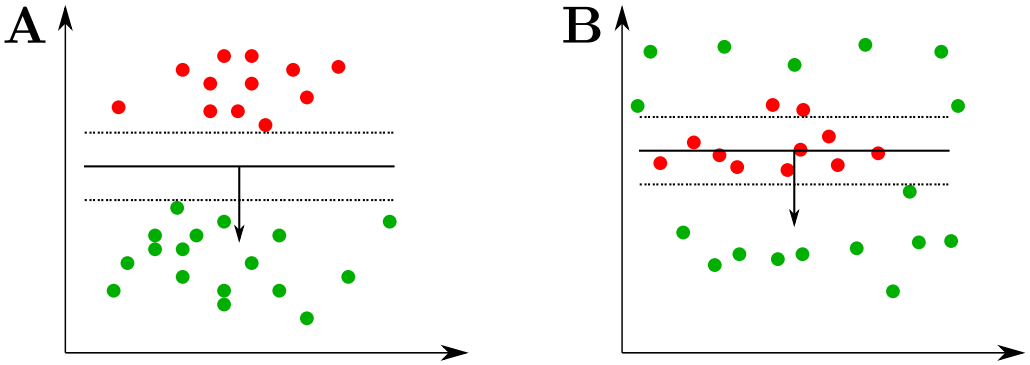


Figure 4.4: Illustration of the SVM classification in two different situations: **A**: the two categories are linearly separable, in which case perfect classification is possible and SVM returns the hyperplane that maximizes the classification margin, *i.e.* the distance between the points and the hyperplane. **B**: if the two categories are not linearly separable, SVM is still guaranteed to converge to a solution, but the resulting classifier will make errors and might not perform very well.

presented with, which can lead to instabilities in the case where correct classification of all examples simultaneously is impossible. The Support Vector Machines algorithm operates in a slightly different setting, by maximizing the classification margin, *i.e.* the signed distance between the hyperplane and the samples. While the corresponding problem can be exactly solved through Quadratic Linear Programming, in order to preserve consistency with the rest of this introduction, we chose to focus on the update procedure as defined on the weights $\boldsymbol{w}$, which is a Gradient Descent on the loss

$$\mathcal{L} = \lambda||\boldsymbol{w}||^2 + C\Big\langle \max(0, 1 - y(\boldsymbol{w} \cdot \boldsymbol{x}))\Big\rangle. \tag{4.7}$$

This loss introduces a $L2$–regularization term in order to avoid that the loss could be made arbitrarily small by increasing the norm of $\boldsymbol{w}$ without changing its orientation. As illustrated in Figure 4.4, this new algorithm is guaranteed to converge to a solution even if the dataset is not linearly separable. However, the solution is only guaranteed to minimize the margin, not to correctly classify the examples. In that case, the use of Kernel functions to map the data to a new space in which they are closer to being linearly separable is highly recommended; since the SVM algorithm is guaranteed to converge anyway, it becomes possible to train SVM on large numbers of randomly generated kernels and select the ones that yield the best results.

4.4 Fully-connected Neural Networks

What happens if we now want to consider non-linearly separable predicates? Two directions exist, and as we shall see, are tightly related:

- Combine several layers of MCP neurons to make the decision boundaries non-linear. A layer is simply a set of independent MCP neurons taking the same inputs but representing different predicates. The state of a layer can be used as input to another layer, allowing the construction of so-called "deep" architectures.
- Switch to a probabilistic version of MCP neurons. Instead of using a Heaviside function to output a binary predicate, one can use any smooth strictly increasing

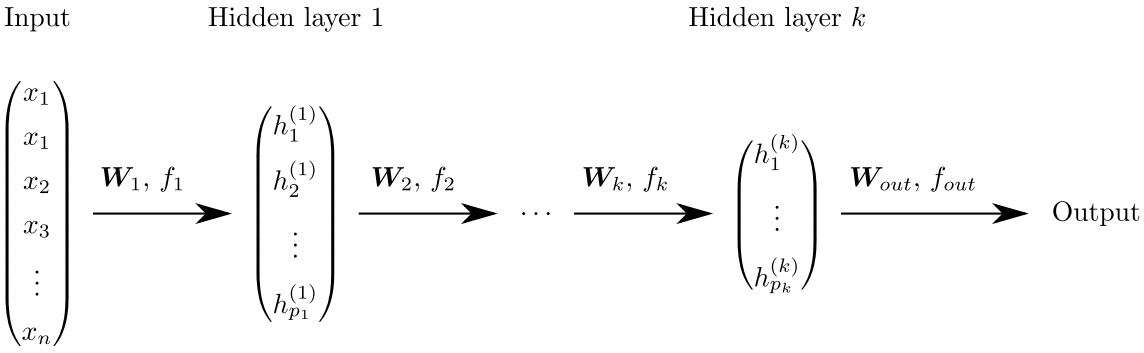


Figure 4.5: Example of Multi Layer Perceptron architecture. The input constitutes the first layer of the network; a set of artificial neurons, acting on the input, constitutes the first **hidden layer**, and the parameter vectors for each of those neurons are grouped into a weight-matrix $\boldsymbol{W}$; additional layers can be added, each time acting on the neurons of the previous one, until the network output. The non-linearity between each layer is primordial, as otherwise all intermediate layers act as a single linear transformation. Adding more layers can be seen as allowing to construct more and more complex decision boundaries, or to map the inputs to simpler and simpler spaces (see Figure 4.6).

function of a real parameter that takes values in $[0, 1]$ to represent a probability estimate of the predicate being true.

First, let us note that the mapping from the input to the state of the so-called "first hidden layer" is effectively done to allow for linear separation by the second hidden layer (that takes the first hidden layer state as input). The main advantage of this "reparametrization" of the input space is that learning algorithms can be adapted to train several layers at once, hence removing the need for hand-crafting (or brute grid-searching) a mapping that make the examples linearly separable. As more layers are added, more complicated decision boundaries can be created, increasing the set of representable predicates (Raghu et al., 2017). However, this increase in expressivity comes at the price of a higher chance of overfitting (see Section 1.9), and Lecun et al. (1989) show that removing unnecessary neurons in overparametrized networks can lead to better performance.

It should also be noted that neurons with activation functions different from the Heaviside can also be used in the hidden layers of a Deep Neural Network. In that case, it is usually preferred to use activations that are not necessarily confined in $[0, 1]$, such as the Rectified Linear Units (ReLU), as they allow to represent both linear and logical behaviors in the same neuron.

4.5 Linear head vs. Softmax head

Until now, we have only been concerned with the question of representing one binary predicate on an input. However, using more than one neuron and more complicated activation functions, more complicated questions can be answered.

The simplest example is that of regression, *e.g.* retrieving the mean color of an image. In that case, the expected output of the network would be a vector of length three, each component indicating the state of the corresponding RGB channel, which can easily be represented by a layer of 3 linear neurons, *i.e.* neurons in which the activation function is the identity. We should mention here that neural networks perform best if they work with input and outputs centered and of or order of magnitude 1. Therefore, since RGB

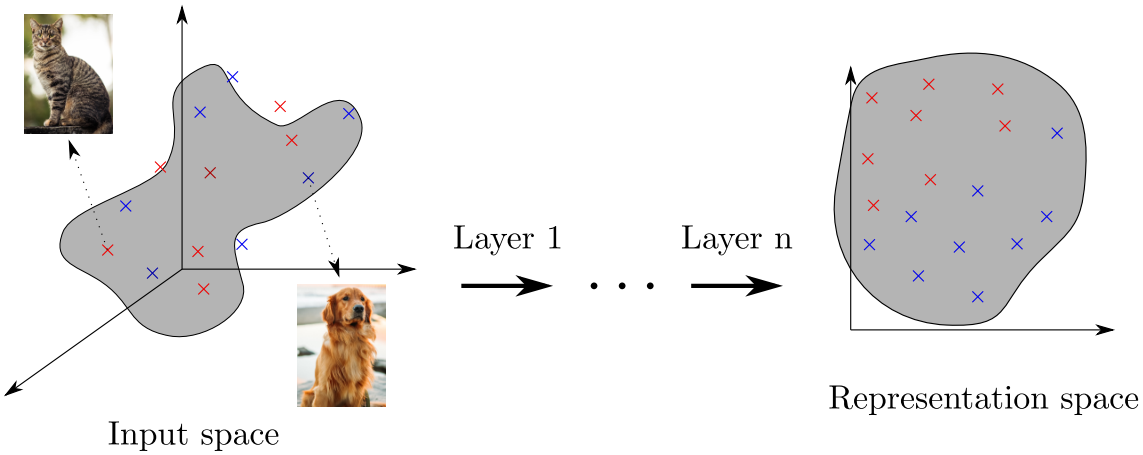


Figure 4.6: Illustration of Multi-Layer Perceptron as a procedure for generating disentangled representations. In the input space, here the space of images, data points lie close to a low-dimensional manifold (Goldt et al., 2020), and points corresponding to different categories are not linearly separable; a single-layer perceptron would not be able to perform the classification. However, performing a succession of non-linear mappings, it is possible to map all example to a new space in which they are linearly separable, so that a perceptron acting on this new representation can perform the classification.

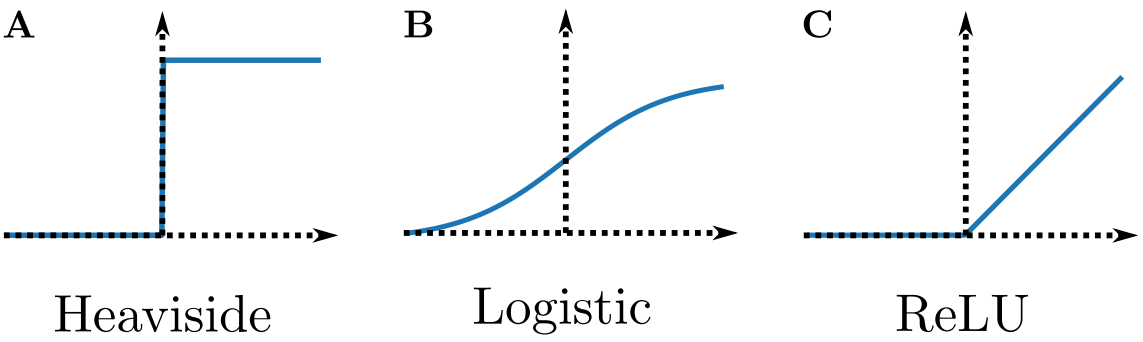


Figure 4.7: Three different activation functions. Due to the fact that they will always be preceded by a parametrizable affine transformation, the horizontal scale and position are irrelevant. **A**: Heaviside unit-step function, equal to the indicator of positive numbers. It can only encode binary information. **B**: The logistic function $(1+e^{-x})^{-1}$, that transforms an arbitrary input into a number between 0 and 1. **C**: The Rectified Linear Unit activation, equal to the identity for positive number and zero for negative ones. This kind of activation functions, able to represent both linear and logical behaviors, have proved extremely useful for Deep Learning.

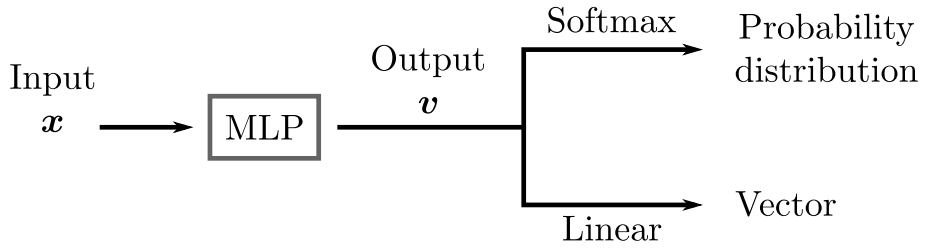


Figure 4.8: Two possible output heads. The linear head simply returns an unconstrained vector, it is perfectly adapted for regression tasks where the output is continuous in nature. The softmax head returns a probability vector over a discrete set, which is more useful in situations where the expected output is categorical.

components are usually in $[0, 256]$, it is recommended to perform a rescaling before using the network.

Another important class of problems is that of multi-class classification, *e.g.* predicting the category of an image from the MNIST handwritten digits dataset. If the state of one neuron is able to represent the probability of a statement being true, it is now possible to use several neurons in parallel to answer tasks of finding "the most likely true predicate" among a given list. Therefore, on a problem with N categories, one can train a layer of N neurons, one for each category, and simply use a "max" over the response of these neurons to output the right category. However, as we shall soon see, differentiability of computations is of paramount importance, therefore making the max function unsuitable for use in deep networks. A widely used solution is to use a so-called "Softmax Layer": instead of forcing the activation of each neuron in the output layer to be in $[0, 1]$, we let them be any real number, and normalize the layer as a whole through

$$\forall i, a_i \leftarrow \frac{\exp a_i}{\sum_k \exp a_k}. \tag{4.8}$$

This global parametrization forces the sum of outputs to be equal to 1, making a probabilistic interpretation in terms of "one-versus-all" probabilities possible.

It should be noted that although neural networks can output only either a vector or a vector of probabilities, these outputs can be used for much more interesting tasks than categorization and regression when incorporated in a suitable setup. For example, the probability vector could be used to represent a stochastic policy in Reinforcement Learning, subsequent outputs of a categorical layer could be used to represent a sentence (each categorical symbol being a word), etc...

4.6 Parametric families of functions and Gradient Descent

Now that we have introduced the first class of Deep Neural Network architectures, the Multi Layer Perceptron, it is time to for us to introduce a more general mathematical framework, largely inspired by Denker et al. (1987). A Machine Learning problem is usually described in terms of the following elements:

- A **dataset** $\{(\boldsymbol{x}_k, \boldsymbol{y}_k); k \in [1, p], \forall k, x \in X, y \in Y\}$ of inputs and corresponding outputs. The simplest example, to which most problems can formally be mapped, is the one where both the input and output spaces are finite-dimensional vector spaces $X = \mathbb{R}^{d_x}$, $Y = \mathbb{R}^{d_y}$, but in some circumstances these spaces can have additional

structure (images have a spatial organization and live in a low-dimensional submanifolds of the space of possible pixels (Goldt et al., 2020); time-series, such as the value of temperature as a function of time, have causality in the temporal direction; words can be mapped to numeric, unambiguous, "tokens", but the values of those numbers, in particular their ordering, is arbitrary).

- A **model**, *i.e.* parametrized family of functions $\mathcal{F} = \{f_\theta; \theta \in \Theta\}$, such that for any value of the parameters θ, f_θ is a function that goes from the input space X to the output space Y. Similarly to the input and output spaces, the parameter space Θ can formally be considered to be $\mathbb{R}^n$. This does not make any restrictions on the family of functions $\mathcal{F}$: they could be $\mathcal{F} = \mathbb{M}_{d_x,d_y}(\mathbb{R})$ the linear mappings from X to Y; a group of d_y perceptrons, each characterized by a vector of size $(d_x + 1)$; or even more advanced neural network structures such as the ones described in Section 5. The choice of the family of functions $\mathbb{F}$ has to be guided by the structure of the data: if the family does not contain any function that faithfully represents the input-output relationships of the dataset, the results at the end of the training procedure will not be satisfying; on the other hand, if the family is too large and contains many functions that can perfectly reproduce the dataset, for example if the number of parameters n becomes much larger than the size of the dataset p, the learning procedure might result in overfitting (see Figure 1.5).

- A **criterion** $\mathcal{C}$, that maps two elements of the output space Y to a scalar, and which is used to quantify how similar those two elements are[4]. This criterion is then used to derive a **loss function**:

$$\mathcal{L}(\theta) = \frac{1}{p} \sum_{k=1}^{p} \mathcal{C}\left(y_k, f_\theta(X_k)\right). \tag{4.9}$$

  The objective of the **training** procedure is then to tune the values of the parameters θ in order to minimize the loss function $\mathcal{L}$, which can be thought of as the equivalent of maximizing a likelihood, and is usually done through some variant of Gradient Descent procedure[5]. Although many algorithms exist (SGD (Bottou, 2010; Ruder, 2017), Newton (Battiti, 1992), Adam (Kingma and Ba, 2017), etc...), they all serve a common purpose: minimizing a function with access only to local evaluations of its gradient and possibly its Hessian[6]. It should be noted that these procedures have theoretical guarantees only in the case where the function to minimize is convex; this is not the case for Neural Networks loss functions, which are often riddled with either local minima or local saddle-points (Dauphin et al., 2014), which will either make training unreliable or slow, as the second-order properties of the energy surfaces are crucial for the speed of convergence (LeCun et al., 1991a,b). In practice the choice of the algorithm and its parameters is often crucial for the success of the training. Interestingly, despite the obvious numerical limitations of such methods, they are also thought to bring increased resistance to overfitting (Advani and Saxe, 2017), possibly by encouraging convergence to solutions of small spectral norms (Arora

[4] Once again, a standard choice for the criterion is the euclidean distance between two vectors of $\mathbb{R}^{d_y}$, but other criteria might be more adapted to specific cases.

[5] Some particular cases exist, for example linear regression, in which the optimization can be performed analytically; they are exceptions in the field of Deep Learning.

[6] Optimization schemes relying on Hessian information, also called second-order optimizers, are often impractical for Deep Learning as the number of parameters can reach millions, so that the Hessian cannot be represented on the physical memory of a computer.

et al., 2019). It should also be noted that Gradient Descent will, in the general case, not converge to the global minima of the loss function; instead, it should rather be thought of as defining a distribution over parameter space, with higher probabilities associated to regions of low error (Levin et al., 1990).

The backpropagation algorithm Because most computations used for processing information inside a neural network use linear algebra, and because of the underlying hardware implementations, it is extremely efficient to compute both the output and its gradient with respect to all parameters of the network in a single forward-backward pass using the chain-rule of multi-variate analysis, commonly referred to as the "backpropagation" algorithm in the context of Deep Learning. Let us illustrate this idea by considering a two-layers binary classifier, whose equations can be written as:

$$\begin{cases} \boldsymbol{h} & = ReLU(\boldsymbol{W}\boldsymbol{x}) \\ p & = \sigma(\boldsymbol{d} \cdot \boldsymbol{h}) \end{cases}, \tag{4.10}$$

where $\sigma(x) = (1+\exp^{-x})^{-1}$ and $ReLU$ maps each component of a vector to the maximum of said element and 0. The parameters of this network are the components of $\boldsymbol{W}$ and $\boldsymbol{d}$, so that our Gradient Descent training will require us to compute the derivatives of the loss function $\mathcal{L}$ with respect to these parameters. Since we are training a classifier, we will use the Negative Log-Likelihood loss, which for a single example $(\boldsymbol{x}, y)$ reads:

$$\mathcal{L} = -\left[y_k * log(p_k) + (1. - y_k) * log(1 - p_k)\right]. \tag{4.11}$$

The back-propagation algorithm computes the derivatives as follows:

- Compute the derivative of $\mathcal{L}$ with respect to p:

$$\frac{\partial \mathcal{L}}{\partial p} = -\left(\frac{y}{p} - \frac{1-y}{1-p}\right). \tag{4.12}$$

- Compute the derivative of p with respect to $\boldsymbol{d}$, and apply the chain rule:

$$\frac{\partial \mathcal{L}}{\partial d_i} = \frac{\partial \mathcal{L}}{\partial p} * \frac{\partial p}{\partial d_i} = -h_i \left(\frac{y}{p} - \frac{1-y}{1-p}\right) \sigma'(\boldsymbol{d} \cdot \boldsymbol{h}). \tag{4.13}$$

- Finally, compute the derivative of p with respect to $\boldsymbol{h}$ and of $\boldsymbol{h}$ with respect to $\boldsymbol{W}$ to obtain the last required derivative:

$$\frac{\partial \mathcal{L}}{\partial h_i} = \frac{\partial \mathcal{L}}{\partial p} * \frac{\partial p}{\partial h_i} = -d_i \left(\frac{y}{p} - \frac{1-y}{1-p}\right) \sigma'(\boldsymbol{d} \cdot \boldsymbol{h}). \tag{4.14}$$

$$\frac{\partial \mathcal{L}}{\partial W_{ij}} = \frac{\partial \mathcal{L}}{\partial d_i} * \frac{\partial d_i}{\partial W_{ij}} = -x_j H(\boldsymbol{W}\boldsymbol{x})|_i d_i \left(\frac{y}{p} - \frac{1-y}{1-p}\right) \sigma'(\boldsymbol{d} \cdot \boldsymbol{h}), \tag{4.15}$$

where H is the Heaviside function, equal to 1 if and only if its input is positive.

It is then possible to perform the Gradient Descent updates using the derivatives we computed:

$$\begin{cases} W_{ij}^{(t+1)} & = W_{ij}^{(t)} - \eta \frac{\partial \mathcal{L}}{\partial W_{ij}} \\ d_i^{(t+1)} & = d_i^{(t)} - \eta \frac{\partial \mathcal{L}}{\partial d_i} \end{cases} \tag{4.16}$$

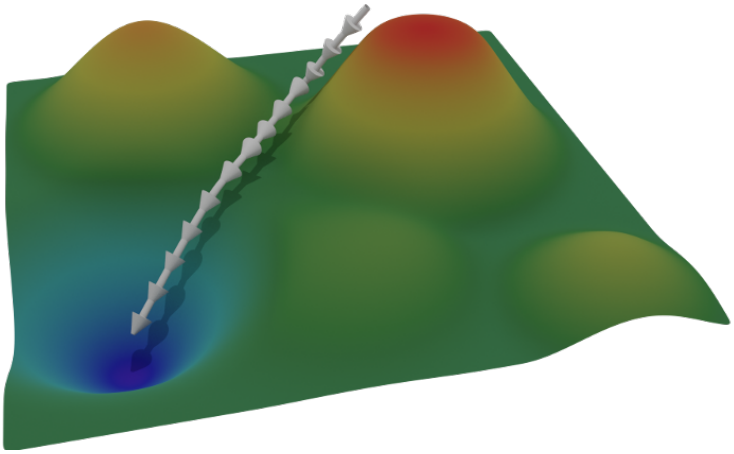


Figure 4.9: Visual illustration of the concept of Gradient Descent. The coordinates (x, y) in the 2–dimensional plane represent the value of the parameters θ that we try to optimize. The altitude z at a given position is given by the value of the cost function $\mathcal{C}$ that we are trying to minimize. The aim of any Gradient Descent algorithm is to change the value of θ in order to minimize $\mathcal{C}(\theta)$, *i.e.* to move downhill in the energy landscape.

While in this simple case the back-propagation could be performed by hand, the computation will become intractable when the number of layers becomes too large or when the computation graph becomes more complicated. In those cases, it becomes important to make use of an automatic differentiation framework, which will both compute the equations of the derivatives with respect to all parameters of the network, and make their computation during the backward pass efficient (*e.g.* by leveraging parallel computing, or keeping track of intermediate results of the forward pass to avoid redundant operations). Different approaches to this problem exist, and two main families can be distinguished:

- **compiled** frameworks, such as TensorFlow (Abadi et al., 2016), take as input a high-level description of the network's computation graph, and generate the machine code for forward and backward passes from it. These static functions are then used at runtime to perform the inference and training of the network.

- **tape-based differentiation** frameworks, such as the Autograd component of PyTorch (Paszke et al., 2019) and the newly introduced interface for TensorFlow, instead keep track of the operations applied during the forward pass (thus, filling the nominal "tape"), and compute the gradients by reading that tape in reverse order.

Both methods have their advantages: tape-based models are typically slower to execute than their compiled counterparts, but allow more flexibility and are easier to prototype and experiment with. All the experiments presented in this Thesis have been lead using the PyTorch framework.

Additionally, a fairly large improvement in performance (both from computer runtime and final results quality points of view) can be obtained by processing several inputs at once in what is referred to as a "batch" and applying Gradient Descent algorithms that exploit this simultaneous evaluation. These improvements have been made even more substantial in the last decade by the use of massively parallel hardware, in the form of Graphical Processing Units, or even more recently of Tensor Processing Units.

4.7 Some examples of Loss function

The freedom in neural network design resides in the choice of the surrogate function, which can be any differentiable mapping from the output space to real numbers. The guiding principle for choosing this function is that its absolute minima (or, much preferably, an overwhelming fraction of its local minima) must correspond to the desired behavior of the network.

Let us now give some examples:

- for the case of image color regression, an appropriate function is the mean-square error between the predicted RGB code and the actual one, since the minimum of this function is unique, and corresponds to perfect predictions. In practice however, if the objective is to obtain results which are meant to be presented to humans, it might be useful to compute the distance between colors in perceptual space, more closely related to the values of Hue, Saturation and Brightness.

- for binary image classification, the natural choice would be the fraction of correctly categorized samples. However, this is not a suitable surrogate as it is not differentiable[7]. Therefore, the negative log-likelihood surrogate is used:

$$\mathcal{L} = -\frac{1}{N}\sum_{k=1}^{N}[\overline{y}_k * log(p_k) + (1. - \overline{y}_k) * log(1 - p_k)]. \tag{4.17}$$

  The minimum of this function corresponds to a case where all examples have probability 1 assigned to the correct class. This Loss function is defined for binary classification, and can straightforwardly be extended to the multi-class case by considering the problem as a "one-versus-rest" classification, in which case, denoting the probabilities for sample k as $p_{k,j}$:

$$\mathcal{L} = -\frac{1}{N}\sum_{k=1}^{N}[\log(p_{k,\overline{y}_k}) + \sum_{j\neq\overline{y}_k}\log(1 - p_{k,j})]. \tag{4.18}$$

- if we wanted a single network, with two heads, to predict both the mean color of an image and the category of an object it represents, one could have one categorical head and one linear head, and use as a surrogate a positively weighted sum of the regression loss and multi-class log loss. Once again, the minimum of the surrogate corresponds to a network where both heads perform their tasks perfectly.

4.8 Transfer learning

As of now, we only considered the case where one had at its disposition a dataset and wished to train a neural network from scratch. However, training even a simple convolutional network requires a fairly large amount of data, which is sometimes not available. Let us take an example: one wants to train a network to recognize bees in an image, but only has, say, 100 images of bees. Obviously, this is far from enough to train a network

[7] More precisely, this loss function has a null derivative everywhere, except at certain points at which the derivative is infinite. Less pathological situations, such as the one of having a ReLU activation in the network (which is non-differentiable at certain points, but such that the derivatives have finite, non-vanishing limits from both directions) can however be accommodated.

with several dozen layers, as those used for state-of-the-art image classification, and a network trained from scratch on such a small dataset would overfit very badly.

This is where **transfer Learning** comes into play: the low-level features used for any image classification tasks are very similar (edges, color contrasts, uniform regions, etc...), so one could easily learn those on a very large, non-specific, dataset before refining them for the exact task at hand. The main advantage in doing so is that generic networks pretrained on very large image databases can be found very easily at almost no cost (whereas training such networks could take weeks, even on powerful computers).

Two main ways of using a pretrained network for a new task exist:

- fine-tuning: all the weights of the network are modified, with a very low learning rate.
- feature extraction: the first layers of the pretrained model are frozen, and one simply uses the network as a fixed transformation to map images to a more relevant set of features, before feeding the results to another Neural Network whose weights are then trained from scratch.

These two methods can obviously be combined easily, fixing the earliest layers and fine-tuning the last ones, for example. It should be noted that the feature extraction method is much more efficient, as in that case the large pretrained network is only used in forward mode and no gradients have to be computed, which reduces drastically the complexity of calculations. The efficiency of transfer learning relies heavily on a proper identification of the relevant features that should be encoded in the feature extractor: if the feature maps contain too much information, the network could overfit; if some critical low-level information is missing (*e.g.* we train a model on a color-insensitive task, then try to reuse it for color regression), the network placed after the extractor has no way of reconstructing it and therefore will perform very poorly.

Another way of looking at transfer learning, which is highly related to the one that we adopt in Chapter 8, is that it relies on the feature extractor to obtain a simpler, more structured **representation** of the input data, making training of the second network easier. For example, training a network to perform Path Integration could lead to the emergence of a **cognitive map**, which can then be used for more involved cognitive tasks such as goal-oriented navigation.

Chapter 5

Deep Learning II: beyond the fully-connected architecture

Abstract

In Chapter 4, we worked our way to a generic mathematical description of Deep Learning: the use of Gradient Descent on a well-chosen surrogate function, represented by a "Neural Network" graph of computations. For now, the graph of computations involved only "dense" connections, meaning that each neuron of a given layer receives input from all neurons of the previous layer. While this is the right way to go for a problem without any exploitable symmetries, it might make the number of tunable parameters so huge that the Gradient Descent procedure will either never converge or find an optimal solution that "overfits" the data: this is the (in)famous "Curse of Dimensionality" (Ganguli and Sompolinsky, 2012). The following section will be devoted to specific architectures that allow for much more efficient representation of specific input symmetries, therefore reducing the number of tunable parameters and necessary training examples. We will begin with the Convolutional Networks, and their extension to Residual Blocks, which are known to be particularly useful in Computer Vision and were inspired by the organization of the visual system observed by neurobiologists. We then consider the models of Recurrent Neural Networks, providing a link with the Ising models of Statistical Physics and a brief history of the major results that were obtained via this analogy, before presenting some refinements of these models (such as the Long Short Term Memory networks) and their applications. Finally, we present some more exotic architectures, such as Differentiable Neural Computers and Adversarial networks

5.1 Convolutional and pooling layers

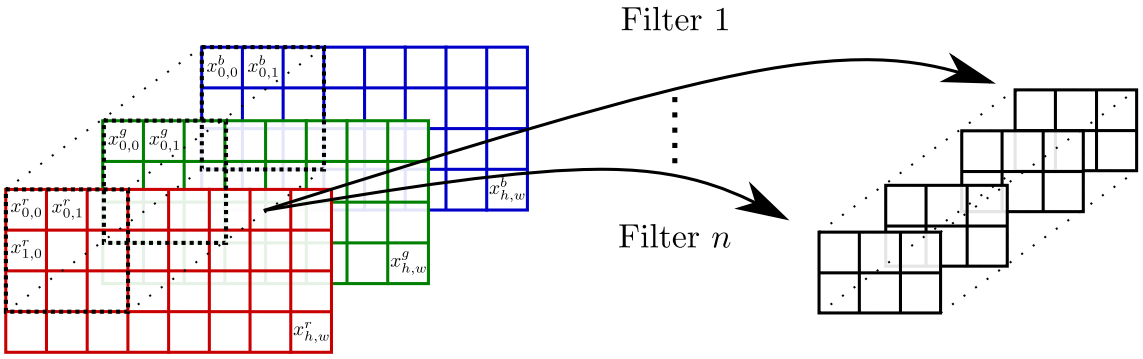
The need for invariances For Computer Vision tasks, using a Fully Connected network usually yields very poor results, because one could never hope to have enough data to correctly tune all parameters of the network, which usually results in the overfitting phenomenon described in Figure 1.5. Fortunately, the features that are of interest for Computer Vision tasks share two important properties: they are, approximately, translation and scale invariant. Using this information, it becomes possible to create networks that will more faithfully reproduce the underlying structure of the data in their internal representations (Goodfellow et al., 2009), and should therefore yield better results.

Extracting local features: filters In the case of an image input, it is clear that a two-pixels shift to the right or to the left does not meaningfully change the information content. A good way to encode this principle is to make use of convolutional layers (LeCun et al., 1989): the activation of the $(n+1)th$ layer is the convolution product of the $n-th$ one with a group of parametrized filters. Each filter acting on the previous layer input generates a so-called "feature map", so that the output of a convolutional layer is a collection of feature maps, that is fed to the next layer[1]. The filters are chosen as having the same number of dimensions as the input (1D for invariance along a sequence, 2D for invariance on an image, etc...) plus one to account for the fact that there are several feature maps for a given layer (and that convolution must be done on all of them at once). The size of the filter is usually chosen very small with respect to the one of the input, allowing the network to extract local features that are invariant under translations (for example, finding high-contrast or locally uniform regions), using only a small number of parameters. It should be noted that Convolutional Networks are strictly less expressive than their fully-connected counterparts, in the sense that the computation graph of any convolution layer can be represented by a dense one, but not the other way around: the improvements in performance come only from the reduction in number of parameters and increased regularity of the represented functions.

Integer vs. fractional stride After choosing the size of the filters to apply to our image, a second parameter has to be chosen: the **stride**, which describes by how many pixels the center of the filter gets translated before being applied again. If the stride is larger than 1, usually an integer, the resulting feature map will be smaller than the previous one; if the stride is smaller than one, the CNN is said to be fractional, and the output feature map will be bigger than the input one. In most applications, CNNs are expected to produce condensed representations (mapping an image to smaller and smaller representations), and the loss of spatial resolution due to integer-stride CNNs is actually desirable as it helps consider more global features of the image; however, in some specific applications (such as super-resolution, or autoencoders, see Section5.5), one is often interested in mapping a low-resolution feature-map to a higher-resolution image, in which case a fractional-stride CNN is desirable.

Changing scale: pooling After the convolution has been applied to the previous layer's feature maps, a pointwise non-linear activation function is applied. In order to progressively construct global features of an image from the result of local convolutions, one has to make use of downsampling operations, most commonly called pooling layers. The goal

[1]Each filter is referred to as a "channel" in analogy to the three "RGB" channels that an image is usually constituted of (which makes an image a 3D tensor for practical purposes)



Input: 3–channels (r, g, b) image

Output: n "feature maps" (n–channels image)

Figure 5.1: Principle of a Convolutional layer. Assuming we start from an RGB image, our input is a 3D tensor of shape (h, w, c_{in}) where $c_{in} = 3$ is the number of input "channels", here the color channels. A convolution is computed across the first two dimensions by computing the dot product of a sliding input volume with a 3D filter. The filter size is usually chosen small with respect to h and w but covers all channels at once, *e.g.* $(3, 3, c_{in})$. The convolution of the input with a given filter produces a single "feature map", and using several filters in parallel one may construct a collection of feature maps of size $(\tilde{h}, \tilde{w}, c_{out})$ which can then be used to feed another convolutional layer.

of these layers is to reduce the size of the feature maps, usually by a factor of two. The state of the "pixel" in the new downsampled feature map has to "condense" the state of the (usually 4) pixels of the initial feature map that correspond to it. This could be done either by taking the mean or the maximum in that region, and the latter is often preferred. As a practical trick, in some situations, downsampling can also be performed crudely by computing the result of the convolution filtering only on a subset of all possible positions, for example every other row and column.

Link with biology It is also of interest to emphasize the link between these networks and our understanding of the mammalian visual cortex. The analogy between an RGB image and the signal generated by the three types of cones in the outer-layer of the retina is self-explanatory, the only simplification between the two being that the image "cones" are uniformly spread, which has no major consequence on the following. The idea of having each neuron connected only to a spatially restricted set of neurons in the previous layer can be thought of as a way of enforcing limits on the spatial spread of the receptive fields of neurons. Then, through successive integration of the information from previous layers, more and more complex representations are generated, in compliance with the hypothesis of Hubel and Wiesel (1962), and McIntosh et al. (2016) have shown that these networks adequately capture the characteristics of neural response to natural scene images. This analogy is however limited because of the choice of using translation invariant filters can be thought of as imposing that different synapses keep exactly identical strengths, which is not biologically plausible. This "weight sharing" is essential to the practical relevance of CNNs, as it provides a drastic decrease in the number of trainable parameters and hence helps prevent overfitting; it is however not the only reason for improvement, as Saxe et al. (2011) note that CNNs with random weights perform surprisingly well, suggesting an intrinsic role of the connectivity structure independently of any training considerations.

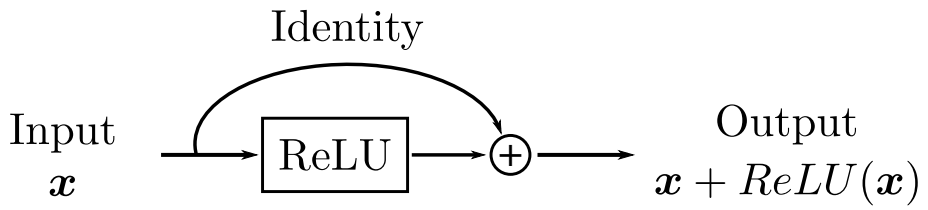


Figure 5.2: Simplest case of a residual block. The information pathway is divided in two, with one part going through a ReLU transformation while the other is left unchanged. The addition performed to reunite the two pathways makes it much easier for the network to represent mappings close to the identity. One could put several ResBlocks instead of a single ReLU, as long as input and output are correctly padded to be of the same size.

5.2 Towards deeper architectures: Residual blocks

A natural idea after introducing convolutional and pooling layers is to try and find ways to reasonably arrange these layers. The most straightforward idea is to construct functional blocks of the type $\{conv2D \rightarrow maxpool2D\}$, chaining (possibly several) convolutional layers into a downsampling layer. Stacking such blocks provides more and more complex representations of the input image, at the cost of lower and lower spatial resolution of the feature maps. The output of the series of blocks, a collection of features maps, is the flattened into a vector then fed to a Fully Connected Network with the correct head for the current task[2]

However, experience showed that networks constructed using this kind of blocks ran into overfitting issues: as the number of layers increased, the final performance decreased. This was mostly attributed to the vanishing gradient issue: when computing the gradients of the loss function, one has to apply the chain rule across all layers starting from the output. Each step of the chain multiplies the average amplitude of the gradients by a factor smaller than 1, making the gradient amplitude exponentially small as one moves to deeper and deeper layers.

A series of experiments on skip-connections (He et al., 2015, 2016) lead to a new type of layers, called Residual Blocks: by dividing the information path in two, one going through convolutional layers and the other through a simple linear activation, then combining this information through a simple addition [3], one effectively forces the factors accumulated across the backpropagation to be close to one, thus reducing the effect of Vanishing Gradient.

An intuitive explanation of this architecture is the following: some features (e.g. presence of a cat) need to be calculated by combining information in very complex manners, requiring a deep network; others in contrast can be calculated using very shallow networks (e.g. edges). The existence of skip connections allow for encoding of both shallow and deep features in the same network. That way, the addition of more layers simply allows the computation of more complex features without making it impossible to propagate the gradients of the simple ones.

[2]For some specific tasks, *e.g.* visual segmentation, the spatial information about the image cannot be lost; in that case, the output head will be a convolutional layer with a single filter, and the need for flattening disappears.

[3]This assumes that the feature maps are of the same size in the two information paths just before the sum, which is practically achieved by padding the images with zeros where the convolution filter cannot be applied.

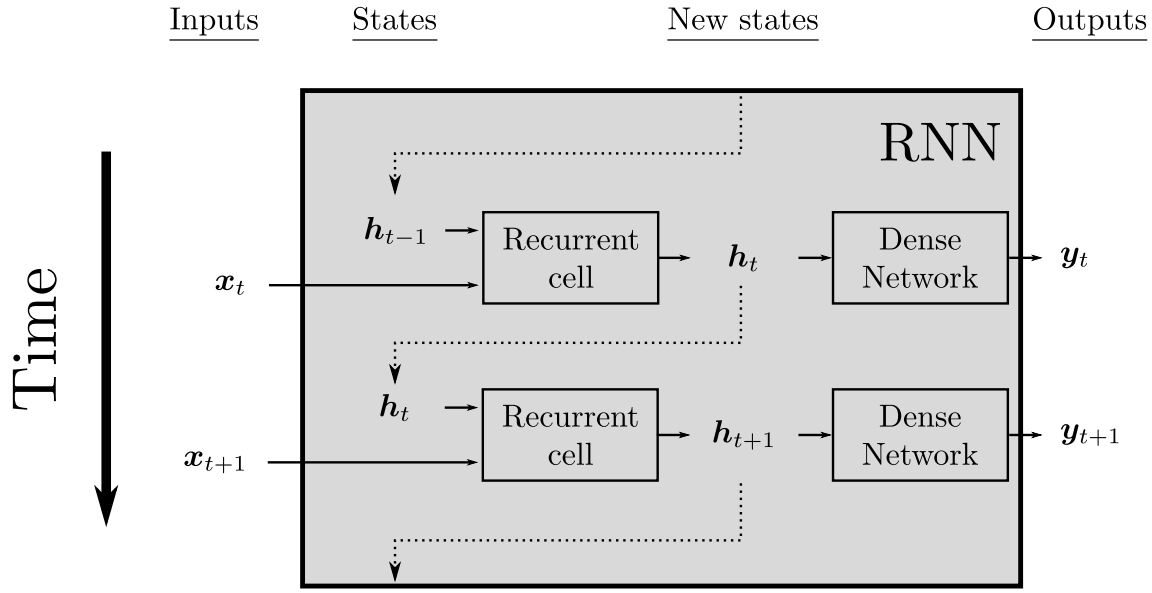


Figure 5.3: Schematic representation of a Recurrent Neural Network (RNN). The inputs and outputs to such networks are, in all generality, time-series of vectors $(\boldsymbol{x}_t)_{t=1,\dots,T_{max}}$ and $(\boldsymbol{y}_t)_{t=1,\dots,T_{max}}$. The network can *a priori* keep track of the previous inputs $(\boldsymbol{x}_0, \dots, \boldsymbol{x}_{t-1})$, through an **internal state** $\boldsymbol{h}_{t-1}$, which is used together with the current input $\boldsymbol{x}_t$ to generate the new internal state $\boldsymbol{h}_t$, from which the output $\boldsymbol{y}_t$ is obtained through a fully-connected network. It should be noted that in this diagram, the "recurrent cell" function that is applied to $(\boldsymbol{h}_t, \boldsymbol{x}_t)$ is the same at all time-steps, and similarly for the "dense network function". Finally, an initial internal state $\boldsymbol{h}_{-1}$ has to be provided, which can be set to the null vector if there is no better candidate.

5.3 Recurrent neural networks

Until now, the only kind of underlying structure that we considered for our data was translational invariance, particularly useful for images. This kind of models are obviously not adapted to modeling dynamics of a variable, where one of the data dimensions is a time index. In that case, one usually needs to properly integrate information along this dimension, sometimes with a more refined method than successive downscaling. The architecture of choice in that case is the one of a recurrent layer: the examples are fed successively to the network, and at time t the recurrent hidden layer receives as an input both the output of the previous hidden layer and its hidden state at time $t-1$.

5.3.1 The Ising model, link with Statistical Physics

A single *layer* of N independent, recurrently-connected, McCulloch-Pitts neurons can be considered as a spin glass system: a vector $\boldsymbol{s}$ represents the spin states[4], its product with the connectivity matrix $\boldsymbol{J}$ gives the local fields at all sites, and the sign of this field is used to update the state, so that it is possible to apply results on the dynamics of the Ising model (Ising (1925), Onsager (1944), etc...).

One example concerns the use of recurrent perceptrons as models of **associative memory**. The paradigm is the following: we define as a memory a configuration $\boldsymbol{s}$ of

[4]Formally, there is a factor two between the ± 1 states of MCP neurons and the spin 1/2 states, which we will forget in the following.

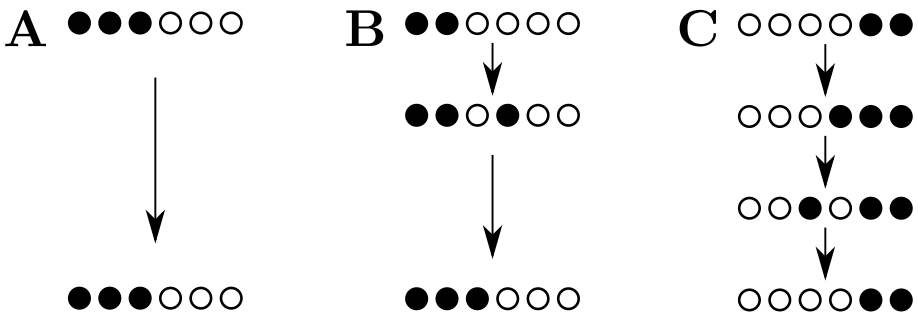


Figure 5.4: Schematic representation of the dynamics in the Ising model, considered as an example of Recurrent Neural Network. Some configurations of the network (**A**) are stable under the application of the state update procedure, while others (**B**) go through a transient regime before converging to a fixed point. Finally, if couplings are allowed to be asymmetric, stable periodic orbits (**C**) can also exist, in which the network will never converge to a stable state, but instead keeps oscillating between a finite number of states.

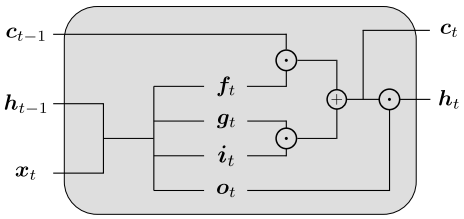
the system that is stable under the Ising dynamics $\boldsymbol{s}_{t+1} = Sign(\boldsymbol{J}\boldsymbol{s}_t)$. Such a definition has the benefit of allowing for a simple model of memory retrieval: each stable configuration of the system is *a priori* surrounded by a **basin of attraction**, in which all states will lead to said memory when repeatedly applying the Ising dynamics; it is then theoretically possible to retrieve that memory starting from a noisy or incomplete version of it. A schematic representation of possible trajectories under Ising dynamics, relevant for the memory analogy, is presented in Figure 5.4.

This approach was popularized by Little (1974) and Hopfield (1982), which proposed a simple prescription to tune the value of the couplings in order to make given patterns stable, based on the idea of "Hebbian Learning" that "neurons that fire together wire together". The expression of the weight-matrix $\boldsymbol{W}$ for a set $\{\boldsymbol{\xi}^{(1)}, \boldsymbol{\xi}^{(2)}, \dots, \boldsymbol{\xi}^{(p)}\}$ is:

$$W_{ij} = \frac{1}{p} \sum_{k=1}^{p} \xi_i^{(k)} \xi_j^{(k)}, \tag{5.1}$$

which is exactly the expression of the auto-correlation matrix of the stored patterns. A natural question that arises is the one of the **memory capacity** $\alpha = p/n$, *i.e.* the ratio between the maximum number of patterns p that can be stored and the number n of neurons in the network, in the limit $n \to \infty$. Amit et al. (1985) show that the capacity of the Hopfield model is equal to $\alpha^{hopfield} \simeq .14n$.

Gardner and Derrida (1988) provide a general estimation of the optimal storage capacity, independently of the choice of training procedure, and show that it grows as $p^{max} = 2N$. This result is obtained by considering directly the volume, in the space of coupling matrices, that satisfies the stability constraints of all patterns simultaneously. Gardner (1988) also proposes a learning rule that achieves this optimal capacity, obtained by applying the perceptron learning rule to all neurons in the population; despite its high storage capacity and guaranteed convergence, this algorithm has the disadvantage that no explicit expression of a coupling matrix $\boldsymbol{W}$ satisfying all constraints can be obtained. Additionally error signals are necessary at the level of each individual neuron, which is thought to be plausible in the cerebellum (Marr, 1969; Albus, 1971) but not necessarily in all cortical regions. Optimizing the learning rule to achieve near-optimal capacities while remaining biologically plausible remains a topic of research to this day, see for example Alemi et al. (2015).



$$\begin{cases} \boldsymbol{i}_t &= \sigma(\boldsymbol{W}_{ii}\boldsymbol{x}_t + \boldsymbol{b}_{ii} + \boldsymbol{W}_{hi}\boldsymbol{h}_{t-1} + \boldsymbol{b}_{hi}) \\ \boldsymbol{f}_t &= \sigma(\boldsymbol{W}_{if}\boldsymbol{x}_t + \boldsymbol{b}_{if} + \boldsymbol{W}_{hf}\boldsymbol{h}_{t-1} + \boldsymbol{b}_{hf}) \\ \boldsymbol{g}_t &= \tanh(\boldsymbol{W}_{ig}\boldsymbol{x}_t + \boldsymbol{b}_{ig} + \boldsymbol{W}_{hg}\boldsymbol{h}_{t-1} + \boldsymbol{b}_{hg}) \\ \boldsymbol{o}_t &= \sigma(\boldsymbol{W}_{io}\boldsymbol{x}_t + \boldsymbol{b}_{io} + \boldsymbol{W}_{ho}\boldsymbol{h}_{t-1} + \boldsymbol{b}_{ho}) \\ \boldsymbol{c}_t &= \boldsymbol{f}_t \bigodot \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \bigodot \boldsymbol{g}_t \\ \boldsymbol{h}_t &= \boldsymbol{o}_t \bigodot \tanh(\boldsymbol{c}_t) \end{cases} \tag{5.2}$$

Figure 5.5: Left: schematic representation of a LSTM layer. Right: computation graph for the corresponding LSTM model. This diagram does not correspond to the original LSTM model introduced in Hochreiter and Schmidhuber (1997), but instead to a more recent version implemented in PyTorch (Paszke et al., 2019) and used in many practical instances. The $\bigodot$ symbol represents the Hadamard operator, taking the product of two vectors component by component.

5.3.2 Subsequent developments

During the 1990s, the idea of Recurrent Neural Networks gained considerable attention, in thanks notably to the work of Elman (1990) and Jordan (1997). However, training of those networks requires performing back-propagation through time (Pearlmutter, 1995), a technique consisting in "unfolding" the recurrent network into a deep feedforward network with shared weights across layers, which is subject to the vanishing/exploding gradient problems, making it impossible to train networks on very long time-scales.

During the 2000s, another type of approach gained traction, broadly referred to as Reservoir Computing. These methods rely on the use of a chaotic system called a **reservoir** (for example, a non-linear recurrent network with random connection matrix) to generate complex dynamics, which are used to perform computation by tuning only the parameter of an input layer connecting the external inputs to the chaotic system, an output layer, used to read the expected output from the state of the chaotic system, and, optionally, a **feedback** layer that maps the outputs of the reservoir back to the inputs. Since the dynamics of the reservoir are not modified during training, methods that do not use feedback connections bypass the need for backpropagation through time, and are therefore much easier to train than other Recurrent Neural Networks. Among the different Reservoir Computer approaches, some such as the Echo State Network (Jaeger, 2001; Jaeger and Haas, 2004) and FORCE algorithm (Sussillo and Abbott, 2009) are based on continuous state neurons, while others such as Liquid State Machines (Maass et al., 2002, 2007) instead consider spiking neurons.

While the Reservoir Computing approaches remain relevant to this day, in particular since their mathematical properties are well established, another type of approach that has enjoyed major empirical success in the last decade is the one of gated recurrent networks, pioneered by the Long Short Term Memory (LSTM) of Hochreiter and Schmidhuber (1997), and more recently the Gated Recurrent Units of Cho et al. (2014). In those networks, the flow of information through the network from one time-step to the next is controlled by a gate, partially alleviating the issues of vanishing and exploding gradients. A major limitation of those networks is their mathematical complexity, which makes it almost impossible to derive any analytical result (see Figure 5.5).

We will now present some results that have been achieved using RNNs.

Dynamical system modeling One of the most generic applications of Recurrent Neural Networks, to which all others can at least formally be identified, consists in producing approximations of Dynamical Systems (Funahashi and Nakamura, 1993), either for purposes of forecasting the future evolution of a quantity given its past history (Connor et al., 1994) or for use in Dynamical Control Systems, see Narendra and Parthasarathy (1990) and Kosmatopoulos et al. (1995).

This problem is closely related to the idea of using a neural network to represent a Markov Decision process, a subclass of discrete-time Dynamical Systems in which the evolution of the system is conditioned by the choice at each time-step of an **action** by one component of the system, referred to as the *agent*. Such models are central in the field of Reinforcement Learning, presented in more details in Section 6, and the study of environment dynamics representation will be the topic of Chapter 8 in which we show how such a model can be used to achieve state-of-the-art performance on the Path Integration task.

Natural Language Processing A particularly tricky example of input space is the one of natural sentences that can be generated in a given language. As theorized by Chomsky (1959), those sequences of words are governed by strong formal structures, known as **grammars**, cannot be represented by *combinatorial circuits*, *i.e.* systems that process information without any memory of past activity the way a purely feedforward network does. Recurrent Neural Networks augment feedforward networks by allowing them to produce joint representations of symbols presented at different times, a necessary condition for Turing completeness (the ability to express any computational task), making them appear as prime candidates for grammar understanding as demonstrated by Cleeremans et al. (1989); Elman (1991). These methods have enjoyed unprecedented empirical success in the last decade, see Cho et al. (2014), and studies have been led to explain why these methods outperform the previous state-of-the-art (Karpathy et al., 2015) using notably Hidden Markov Models.

An interesting subtopic of Natural Language processing is the one of word embeddings: in order to be able to use Neural Networks on sentences, one must first transform them into time-series by finding an appropriate mapping from words to vectors. A very naïve solution consists in building a "one-hot" encoding of words by using vectors of length equal to the number of possible words, each word being represented as a vector of zeros and a single one. Even if such an encoding was not impossible to put in practice due to the sheer size of the resulting vectors, the space of possible words would be represented without any consideration for the semantic relationships between words. A much more elegant solution consists in trying to map words to vectors in such a way that related words (such as "apple" and "orange") are more "similar" than unrelated ones (*e.g.* "apple" and "neuron"); some approaches, such as GloVE embeddings claim to represent even stronger relationships (*e.g.* asking that in embedding space, "king-man+woman=queen"). Those embeddings are usually constructed from statistical analysis of large text corpora, such as co-occurence of words, and techniques have recently been proposed to improve them further by pretraining Deep Recurrent Networks for specific Natural Language Processing tasks (for example, prediction of each word in a sentence given both its predecessor and successor, see Devlin et al. (2019)).

Sound analysis One notable example in which RNNs do not perform nearly as well as one could expect is the one of direct sound analysis. Physically, sounds are transmitted through acoustic waves, which can be fully characterized by the value of the local pressure

as a function of time; this **wave-form** can be recorded through a microphone, yielding a 1-dimensional time-series $(p_t)_{t=1,\dots,T}$ which can be used as direct input to an RNN to try and perform *e.g.* instrument classification or phoneme detection. One way to understand why these time-series might not be adapted to RNNs is that the time-scales on which information has to propagate are much longer than those that can be achieved by standard RNNs: typical wave-form recordings are generated at 8kHz, and the duration of an individual phoneme is around 100ms, meaning that the number of time-steps in the waveform is around 1000.

A much more common and successful approach to Machine Learning on sound relies on prior knowledge about the human auditory system, which was already mentioned in Section 3.4.2: the wave-form is first mapped to **MEL Frequency Cepstrum Coefficients**, a time-series where each time-step is an energy density histogram, across the audible frequency domain, computed in a sliding window across the waveform and sometimes referred to as a **cochleogram** due to its similarity with the way sound is perceived at the level of the human cochlea. An example of such mapping was presented in Figure 3.6. The time-series obtained in this way are much denser in information, and the time-scales on which information has to be integrated becomes short enough to allow for meaningful tasks to be performed (Graves and Schmidhuber, 2005; Amodei et al., 2016). It should however be noted that Convolutional Networks are also often used to analyze cochleograms, see Abdel-Hamid et al. (2014), with convolutions being applied both in the time and in the energy dimensions, the latter being much less intuitively relevant as invariance of audio patterns across the energy spectrum is not well established.

5.4 Differentiable Neural Computers

In specific contexts, one might want to consider is called the "Differentiable Neural Computer" (Graves et al., 2016), constructed as an extension of the Neural Turing Machines (Graves et al., 2014). As the name suggests, this particular architecture was not developed in analogy with neuroscience, but with computer hardware design.

In a traditional computer, the Core Process Unit which executes the computations stores data on an external static memory that can be accessed without interfering with the computations. In a feedforward neural network, the only form of memory is the value of the weights at the end of the training procedure, which can be seen as a memory of the history of training examples, and no information can be stored at the time of inference when the network is provided with new examples: from a computation theory point of view, these are "combinatorial circuits" and not fully-capable Turing Machines. In a recurrent network, the situation is different as a persistent internal state exists, allowing to propagate information between time-steps. However, this "memory" is *a priori* affecting and affected by the computation currently performed, so that keeping information stable for a large number of updates might be difficult, requiring fine-tuning of the weights. This type of memory could be compared to the Cache of a CPU: a small, very efficient memory that is not made to hold information for a long time.

A Differentiable Neural Computer aims at introducing a much more robust form of information storage: a neural network, the **Controller** is used to manage an external memory array, in a very similar way as a CPU accessing its RAM. The controller is used to manage read and write heads that can affect the memory matrix, and these heads benefit from three forms of attention mechanisms:

- content lookup: a "key" vector is used to find memory addresses with similar content, assigning a weight to each address. This can be used for associative recall, where

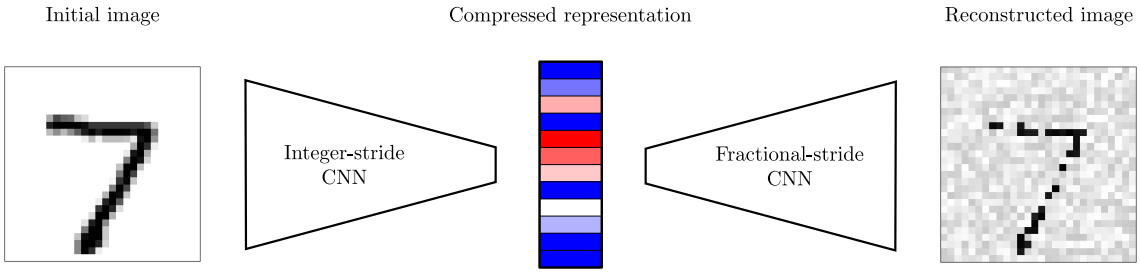


Figure 5.6: General structure of an auto-encoder network for images. The initial image is mapped to a low-dimensional vector, through an **encoder** CNN whose parameters (in particular, an integer stride) are chosen to reduce the size of the feature map (representation) at each layer. This vector can then be used as the input to a **decoder**, whose connectivity pattern can be thought of as "symmetric" to the one of the encoder in the sense that it uses "fractional" stride to increase the size of the representation at each layer. The final output of the encoder-decoder system has the same shape as the original image, and the parameters of both networks are tuned simultaneously to minimize the distance between the input and output.

the key is a noisy or incomplete version of the desired information.

- temporal link mapping: by recording the matrix of sites that were consecutively written to, one can define an operator that transforms an attention vector w into a time-shifted version (for example, the sites that were written to just before the name "dog" was put in the memory)

- usage evaluation: the controller can access the state of usage of all addresses, in order to know which ones are important and which ones can be overwritten.

As such, a DNC can be seen as a differentiable analog to a fully Turing-Complete computer. Experiments suggest that Gradient Descent can be used to accommodate both supervised and unsupervised learning tasks that are otherwise considered impossible for LSTM networks.

5.5 Auto-encoders

In Chapter 1, we briefly introduced the concept of dimensionality reduction through Principal Component Analysis, a technique in which information about a set of n–dimensional vectors is condensed by projecting each vector on a low-dimensional manifold, chosen to maximize the variance of the projected points. Similarly to Multi Layer Perceptrons, this method makes the assumption that no particular structure exist on the space of input data; therefore, when trying to compress structured data such as images, it might be interesting to use specialized architectures, usually referred to as **auto-encoders**.

While many architectures are possible depending on the type of input data and the goal of the auto-encoder, they usually rely on an hourglass shape in the computation graph, see Figure 5.6, and we will focus on the example of image data: an input image with N pixels, is mapped to a n–dimensional vector ($n \ll N$) by a Convolutional Neural Network; then, a "fractional" Convolutional Network is used to map this condensed representation back to the original image shape. The parameters of both networks are initialized randomly, and trained jointly in a supervised way on the distance between initial and reconstructed

images[5]. Depending on the context, it might be useful to impose additional constraints on the compressed representation obtained after the encoder: in the case of MNIST digit images, one could try to impose that this encoding retains information on the category of the image by simultaneously training a classifier taking the encoding as input; conversely, one could be interested in creating a model which does not retain that information, so that the intermediate representation will be as unstructured as possible, making it more suitable for image generation (see next section).

It should be noted that these techniques are not yet a viable replacement for wavelet transforms (Mallat, 1996) in the context of image compression, but rather provide a more anatomically plausible alternative for explaining this type of compression in biological information processing systems.

5.6 Generative Adversarial Networks

A natural question that arises from the definition of auto-encoders is whether those networks could be used as generative models: for example, one could train a Boltzmann Machine to reproduce the statistics of the compressed representations, and use the decoder as a **generative model** to obtain a distribution on images. While this approach is of course possible, an empirically much more successful approach consists in introducing a second network, called the **discriminator**, whose role is to predict if the images it receives are real or were artificially generated, see Figure 5.7. The training procedure for these two networks is said to be adversarial, as one network is trained to maximize the error of the other: two gradient descent optimizations have to be performed in parallel, each working on the parameters of one of the subnetworks.

If we denote as $\mathcal{G}$ the parameters of the generator network and $\mathcal{D}$ the ones of the discriminator, the optimization problem has the form:

$$\max_{\mathcal{D}} \min_{\mathcal{G}} \quad \Big\langle \log(\mathcal{D}(\boldsymbol{x})) \Big\rangle_{\boldsymbol{x} \sim p_{data}} + \Big\langle \log(1 - \mathcal{D}(\boldsymbol{x})) \Big\rangle_{\boldsymbol{x} \sim p_{gen}}. \tag{5.3}$$

It can easily be shown that the optimal discriminator satisfies

$$\mathcal{D}^*(\boldsymbol{x}) \equiv \frac{p_{data}(x)}{p_{data}(x) + p_{gen}(x)}, \tag{5.4}$$

and that the optimal generator satisfies $p_{gen}(x) \equiv p_{data}(x)$, so that when the generator is optimal the discriminator is maximally confused and answers $1/2$ for every sample.

The question of how to improve the performance of these networks, usually through the use of meaningful regularization terms in the loss function (an approach similar to the one we explore in Chapters 2 and 8) or more involved optimization schemes, remains mostly open and has given rise to a wide variety of different models, see Creswell et al. (2018) for a review.

[5]This type of dimensionality reduction method is still generally considered unsupervised, as no label is attached to the input images.

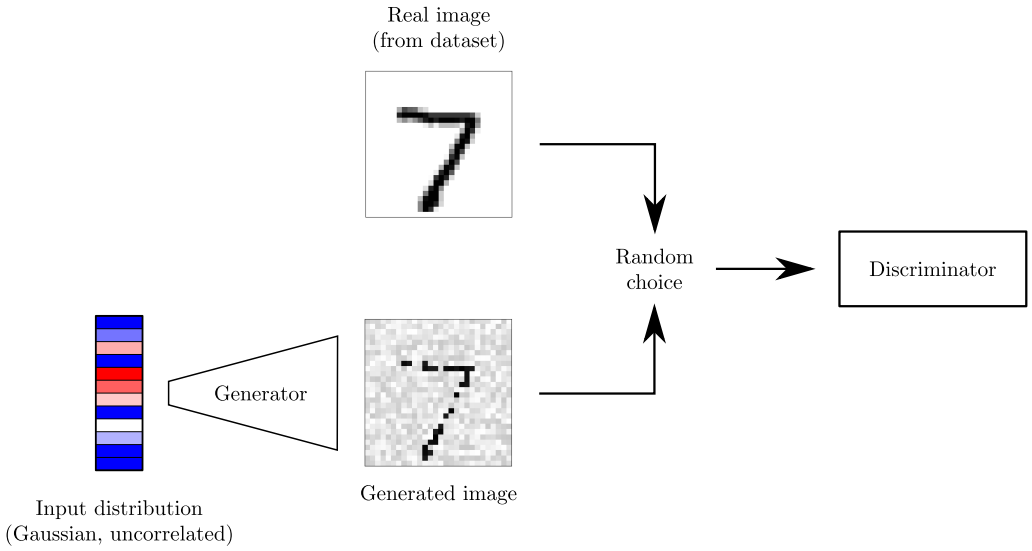


Figure 5.7: Architecture for a Generative Adversarial Network. A **generator** is used to map an initial probability distribution, usually a n–dimensional Gaussian with no correlation between variables, to a distribution of images; a **discriminator** is used to determine if the images it receives come from the initial dataset, or from the generator. Those networks are trained in an adversarial manner: the network is optimized to fool the discriminator, and the discriminator is optimized for the exact opposite.

Chapter 6

Deep Learning III: the Reinforcement Learning paradigm

Abstract

In this Chapter, we present some history and recent advances in Reinforcement Learning. This paradigm completes the one of supervised learning algorithms, which require the data scientist to establish a set of "ground-truth" labeled examples, by allowing the agent to learn from trial-and-error. Such methods, initially developed in the field of Statistics to find optimal exploration strategies in Multi-Armed Bandits, are now formulated in the framework of Markov Decision Processes, and have been applied with huge practical success in the field of video game AI. In those problems, it is often impossible to write optimal policies "by hand", but the Deep Reinforcement Learning algorithms (combining the ideas of Markov Decision Processes with neural network approximation techniques) allow agents to learn such policies. From the point of view of behavioral neuroscience, these experiments can bring valuable insights by allowing comparison between the policies of biological agents and of artificial ones.

6.1 Link with Optimal Control Theory

Reinforcement Learning is closely related to the field of Optimal Control theory (Kirk, 2004), in which an exact model of a physical dynamical system is explicitly given to a "planner", for example as a set of differential equations, along with an objective (*e.g.* maximize the yield of a chemical reaction, stabilize the generated power in a nuclear plant to its nominal value, etc...); the planner then has to determine how to interact with the system (resp. increase or decrease the temperature of the solution and the concentration of products, increase or decrease the interaction rate) to obtain the expected result. It has been found by Lillicrap et al. (2019) that model-free Reinforcement Learning procedures can achieve similar results to those procedures, despite lacking explicit access to the information about the system dynamics. This is very similar to the deduction of an optimal **policy** from the inference of a Q-function through Bellman equations and Monte-Carlo exploration of an environment that we explain in this Chapter: the Q-function is a model of the action-reward relationship; its evaluation relies on the computation of the transition matrix as an intermediate step, which is a model of the state evolution under the influence of the "agent"; the extraction of the optimal policy from the Q-function, albeit trivial, constitutes the planning procedure.

While it is rarely incorporated directly in training procedures, it is often implicitly expected from Deep Reinforcement Learning algorithms to construct meaningful representations of their environments, and perform some form of planning on those. It has been shown that learning of representations is a limiting step in many Machine Learning contexts (see Bengio et al. (2013a) for a review), and we argue that one way to facilitate this emergence is to explicitly train the representation layers, usually the first ones in the information processing pathway, as part of an environment model. In Chapter 8, we will consider the application of this idea to a spatial navigation task. The internal model, which will feature both a **direct** and **inverse** component, will be implemented through a Neural Network, trained from random interactions with the environment, and we show that these models can be used for efficient learning of spatially organized tasks.

6.2 The Multi-Armed Bandit problem

As a way to fix intuition, let us consider for a moment a bee, who every day has to leave its hive in order to collect pollen; at the beginning, the bee has no choice but to **explore** the surroundings, more or less randomly; after some time, it has learned the path to a nearby flower patch, to which it could return every day to **exploit** its current knowledge of the environment; however, if it were to continue to explore a bit longer, it might find a bigger, much more rewarding field just a few miles further, which would be more beneficial in the long run. This situation, often referred to as the exploration-exploitation dilemma, is a major point of interest in Reinforcement Learning. One of the first practical applications of this dilemma comes from the field of medical trials (Stoyan, 1987): how should a researcher choose which subjects do and do not receive a candidate treatment in order to maximize the information collected about the treatment's efficiency, while also helping as many patients as possible to recover? in recent years, with the apparition of the World Wide Web, similar questions of "regret minimization" arose for recommendation systems, such as the ones used for news article (Li et al., 2010), advertisement, and audio-visual content on platforms such as Youtube, Spotify or Netflix.

The resulting model, called the Multi-Armed Bandit, is described as follows: an **agent** is placed in front of a slot machine, which has N different arms that it can pull; at each

time-step, the agent chooses one arm to pull, and receives a certain amount of money, positive or negative, which we call a **reward**; the distribution of possible rewards at a given step is determined by which arm the agent pulls; the objective of the agent is to obtain a sum of rewards as high as possible on sessions of $T \simeq \infty$ pulls, averaged over the randomness of the rewards at each time-step.

Just to fix the ideas, consider a very simple case in which there are two arms, and the distributions of rewards for arm i is a Bernoulli with probability p_i of outputting 0, with $p_1 > p_2$. If we consider an omniscient agent that knows those two distributions, in particular the values of p_1 and p_2, the optimal strategy π^* to maximize the rewards is to always pull the first arm, wince it has the highest chance of actually outputting a reward, and the expected cumulated reward in that case is given by:

$$\Big\langle \sum_{t=1}^{T} r^{\pi^*} \Big\rangle = p_1 \, T. \tag{6.1}$$

A more realistic scenario has the agent begin its session without knowing the values of $\boldsymbol{p} = (p_1, p_2)$, which it has to infer at the same time as it is collecting rewards. A possible strategy π^{IE} when faced with this situation is the one of Initial Exploration: at the beginning, choose each arm k times, then always pull the arm which yielded the highest average reward during this initial exploration phase. While this strategy can be useful in practice, it is far from being optimal in the case where $T = \infty$: in all non-degenerate cases (*i.e.* both p_i are different from 0, 1 and each other), there is probability ϵ that after k draws of both arms, the arm with the lowest p_i gave a larger reward than the other. Therefore, the expected cumulated reward in that case is given by:

$$\Big\langle \sum_{t=1}^{T} r^{\pi^{IE}} \Big\rangle = (1-\epsilon)\, p_1 \, T + \epsilon \, p_2 \, T. \tag{6.2}$$

It is then possible to define the **regret** of a policy π as the difference between the average expected reward when choosing the arms according to π, and the average expected reward under the optimal policy π^*. In our case, the regret of the Initial Exploration strategy grows linearly with time:

$$R(\pi^{IE}) = \epsilon \, (p_2 - p_1) \, T. \tag{6.3}$$

A more involved strategy called Upper Confidence Bound π^{UCB} was defined in Lai and Robbins (1985) and achives a logarithmic scaling of regret, which the authors show is the optimal possible scaling in the general case. This algorithm proceeds by keeping an estimation of the average reward $\hat{r}_i$ of all arms, as well as the uncertainty σ_i on this average; the arm that is chosen is not the one that currently has the highest average, but the highest "upper confidence bound"

$$\hat{p}_i^{opt}(t) = \underbrace{\frac{1}{N_{i,t}} \sum_{t, a_t = i} r_t}_{\text{MAP estimate: mean } \overline{p}} + \underbrace{c \sqrt{\frac{\ln(t)}{N_{i,t}}}}_{\text{Uncertainty penalty } \sigma_{p_i}}, \tag{6.4}$$

where $N_{i,t}$ is the number of times the arm i has been chosen up to time t, and c controls how much emphasis the algorithm puts on exploration. Intuitively, this algorithm can be formulated as optimism in the face of uncertainty: the agent will sometimes choose an arm that has a suboptimal average reward in order to improve its knowledge about it, and make sure that the one it uses is truly optimal. It should be noted that the second

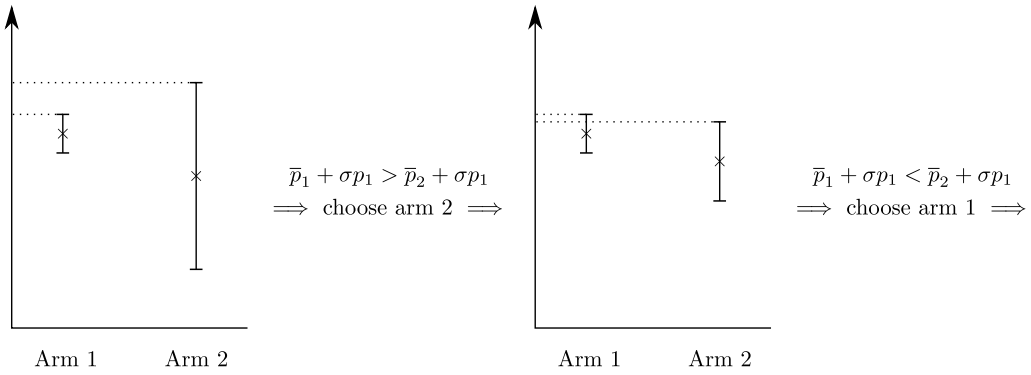


Figure 6.1: Example of Upper Confidence Bound for the Multi-Armed Bandit problem. After a certain number of draws, the agent has constructed confidence intervals on the values of the parameters of the two Bernoulli arms, p_1 and p_2. The Maximum A Posteriori estimates, *i.e.* the centers of those intervals, seem to indicate that arm 1 has the higher reward probability of the two; however, the uncertainty on p_2 is larger since that arm has been drawn less often, so that the "optimistic" estimation (top of the confidence interval) is larger for arm 2. The agent therefore chooses to draw from arm 2, in order to decrease the uncertainty in its estimation of p_2. At the next step, the uncertainty on arm 2 has decreased enough that the Upper Bound on p_2 is now below the one of arm 1, and the agent will now choose it. To achieve log-scaling regret, the confidence level is increased as a function of time, encouraging the agent to always keep exploring all states, albeit less frequently.

term in 6.4 will diverge if t increases, ensuring that exploration steps are still taken even at large times (hence the logarithmic regret).

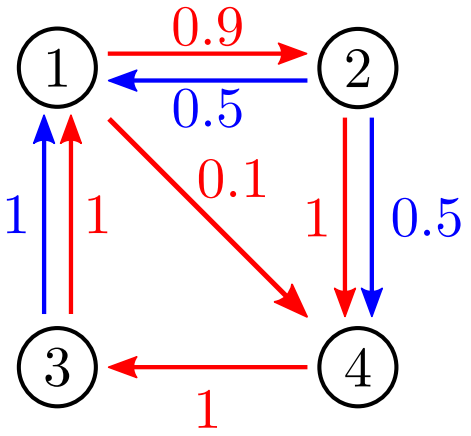
It should however be noted that the theoretical asymptotics of the UCB algorithm are only valid after a very large number of trials, often much more than the typical use case, and simpler algorithms might significantly outperform it in realistic settings (Kuleshov and Precup, 2014). Finite-time results are however much harder to derive, and remain an active topic of research to this day (Auer et al., 2002).

6.3 Markov Decision Processes

One of the major limitations of the classical Multi-Armed Bandit framework is that it assumes that the distribution of rewards attached to each arm is independent of the arms chosen by the agent at previous time-steps. Going back to our bee analogy of the previous section, we would want to be able to take into account the fact that if our bee collects all the pollen in a given area, then the next time it will visit it there will be no pollen left, and the whole trip will yield no reward. The way we incorporate this subtlety is by introducing the concept of environment **states**, *e.g.* the quantity of pollen remaining in each patch of flower. When the agent performs a given **action** (the generalization of the arms in the bandit case), the state undergoes a **transition**, which might be stochastic or deterministic, and a reward is given to the agent, conditioned on the previous state and the action.

The resulting mathematical object is called a Markov Decision Process (MDP) (Sutton and Barto, 1998), and is characterized by the following elements:

- a state space $\mathcal{S}$



Blue action transition matrix

$$\begin{pmatrix} - & 0.5 & 1 & - \\ - & 0 & 0 & - \\ - & 0 & 0 & - \\ - & 0.5 & 0 & - \end{pmatrix}$$

Red action transition matrix

$$\begin{pmatrix} 0 & 0 & 1 & 0 \\ 0.9 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0.1 & 1 & 0 & 0 \end{pmatrix}$$

Figure 6.2: Example of an MDP with two actions (blue or red) and four states. The transition probability matrices $\boldsymbol{P}$ are such that $P_{i,j} = p(s_{t+1} = i | s_t = j)$ when choosing the corresponding action. The rewards are not included for clarity, as each of the arrows could be associated with its own reward distribution. The blue action is not available in all states, which is not a problem for later algorithms. A situation not represented here would be transition probabilities from a state to itself, which once again requires only minor adjustments.

- an observation space $\mathcal{O}$
- an action space $\mathcal{A}$
- For each state $s \in \mathcal{S}$ and action $a \in \mathcal{A}$:
  - a distribution of rewards $r(s, a)$.
  - a transition probability distribution $p(\tilde{s}|s, a)$.

Those two distributions, which we will mostly consider to be deterministic in the following, are used so that every time the action a is taken while in state s, a reward is drawn from $r(s, a)$ and a new state is sampled from $p(\tilde{s}|s, a)$. In the bee analogy, the state transition could be that once the bee has collected pollen 3 times from the same location, subsequent attempts to collect at that same location will yield no reward; we provide in Figure 6.2 an example of abstract MDP for reference.

6.4 Exact solution of an MDP: Bellman equations and Dynamic Programming

Similarly to the case of Multi-Armed Bandits, one might be interested in deriving policies $\pi : \mathcal{S} \to \mathcal{A}$ that map environment states to agent actions, in such a way that following that policy yields, averaged on transitions and reward randomness, the highest possible cumulated reward. Such policies can *a priori* be chosen deterministic or stochastic, but it can be shown that at least one deterministic policy exists that maximizes the expected reward (see Sutton and Barto (1998) for a full mathematical proof).

To derive such an optimal policy, it is useful to introduce the **State-Action Value Function**, often referred to as the **Q–function**, especially in the context of Deep Rein-

forcement Learning. It is formally defined for any policy π as:

$$Q(s,a) = \Big\langle r(s,a) + \sum_{t=1}^{\infty} \gamma^t \, r(s_t, \pi(s_t)) \Big\rangle_{\text{rewards, transitions}}, \tag{6.5}$$

where $(s_t)_{t=1..\infty}$ is the trajectory followed by the agent when starting from state s, taking initial action a, and subsequently following the policy π. The randomness of the MDP can be found both in the value of the reward and the exact trajectory followed. The **discount factor** γ is a real number between 0 and 1, quantifying how much the future rewards are weighted with respect to immediate ones[1]. From the Q–function of a given policy, one can also define its **value function** $V(s) = Q(s, \pi(s))$, which is simply the expected discounted sum of rewards following the policy π at every step starting in state s.

Let us now introduce Richard Bellman's **Principle of Optimality** (Bellman, 1954):

An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision.

Mathematically, the Value Function of the optimal policy, V^*, must be maximal at any state, from which one can deduce the unicity of the optimal value function as well as the **Optimal Bellman Equation**:

$$\forall s, V^*(s) = \max_a \Big\{ r(s,a) + \gamma \sum_{\tilde{s}} V^*(\tilde{s}) \, p(\tilde{s}|s,a) \Big\}. \tag{6.6}$$

Since this is a non-linear system of equations, one for each possible state, analytical resolution is not possible. However, one may introduce the **Optimal Bellman Operator** $\mathcal{B}$ acting on the space of Value Functions (since we only consider discrete state-spaces, this space is formally identifiable to $\mathbb{R}^N$ where N is the number of states):

$$\forall u \in R^N, \quad \mathcal{B}u = \max_a \Big\{ r(s,a) + \gamma \sum_{\tilde{s}} u_{\tilde{s}} \, p(\tilde{s}|s,a) \Big\}, \tag{6.7}$$

which can easily be shown to be L_∞-contractive[2]. From the Banach Fixed Point Theorem, $\mathcal{B}$ admits a single fixed point, which satisfies the Bellman Optimality Principle: the fixed point is the optimal value function V^*.

Using the aforementioned properties, one can derive the so-called Value Iteration algorithm, which allows for iterative numerical solving of the non-linear system 6.6: starting from any value function V_0 and applying $\mathcal{B}$ an infinite number of times, one recovers the optimal value function V^*, from which the optimal policy is computed greedily as

$$\pi^*(s) = \arg\max_a \Big\{ r(s,a) + \gamma \sum_{\tilde{s}} V^*(\tilde{s}) \, p(\tilde{s}|s,a) \Big\}. \tag{6.8}$$

6.5 The Q-learning algorithm

Many variants of the Bellman algorithm exist, depending on the particular situation at hand, but they rely both on access to all the information about the MDP, and the ability to exactly represent this information. These are very restrictive hypotheses, in particular the second one, and we will show in a later section that recent empirical successes of Reinforcement Learning can at least partially be attributed to the emergence of new

[1] It is not obvious how exactly to fix γ, but a good rule of thumb is to chose is so that γ^T vanishes in a time comparable to the time on which the agent should try to plan its moves.

[2] For any couple (u, v), $|\mathcal{B}u - \mathcal{B}v|_\infty < |u - v|_\infty$.

Figure 6.3: Generic Reinforcement Learning scheme. The agent interacts with its environment by "observing" the environment state s_t; using its policy π, it acts on the environment through an action $a_t = \pi(s_t)$; given this action, the environment updates its state, and the agent observes both the new environment state s_{t+1} and the reward r_t it obtained when performing its action. The resulting **transition tuple** $\tau = (s_t, a_t, s_{t+1}, r_t)$ is then transmitted to the Reinforcement Learning algorithm, which updates the policy π accordingly in order to maximize the obtained rewards.

approximation techniques for high-dimensional functions, namely Deep Neural Networks. However, if the spaces of actions $\mathcal{A}$ and states $\mathcal{S}$ are discrete, all the probability distributions involved are finite-sized matrices, whose components can *a priori* be estimated to an arbitrary level of precision in finite-time by using a suitable exploration strategy (for example, at every step choose the action which was used the least when in the current state). This approach is often described as a particular case of Monte-Carlo method, and can be seen as an analogue of the "Initial Exploration" strategy of Multi-Armed Bandits. While this approach is far from optimal to minimize the regret during learning, it allows the agent to construct a perfect **model** of the environment (*i.e.* the MDP) which it can then exploit to compute a policy that will yield optimal regret at test time.

We will now switch to an episodic framework: the agent has access to a simulator of the MDP, that we will call the **environment**. The agent can interact with the environment in order to generate trajectories along which it gathers information and, at the same time, optimizes its policy. This "trial-and-error", or "unsupervised learning", is much more closely related to what living organisms experience than the standard classification or regression tasks of supervised learning.

Let us now present the Q-Learning algorithm (Watkins and Dayan, 1992), used to solve this problem. It relies on a bootstrap estimation of the Q–function from many trajectories. At each step of each trajectory, we apply the following procedure:

- Choose an action a according to a suitable exploration policy
- Observe the new state and reward $(\tilde{s}, r)$
- Update the current estimate Q_t using this new information

We will not enter the details of the different policy updates and their convergence guarantees, as they are fairly mathematically involved and will not be important in the following. We once again refer to (Sutton and Barto, 1998) for more details, and will focus on the particular case of Approximate Reinforcement Learning, which will allow use to make a link towards Deep Learning.

An important detail however is the following: what is a suitable exploration policy? This is an example of the famous **Exploration / Exploitation dilemma**: if the agent only acts according to its current estimation of the best policy, it will fail to find more hidden but better performing strategies; on the other hand, if it tries all actions at random, it will explore all possible trajectories but with such a low probability that it will never get enough information to find the optimal solution. A good compromise between the

two is the so-called ϵ–greedy exploration scheme: at each time-step, the agent chooses his current optimal action with probability $1-\epsilon$, and a random action with probability ϵ. In particular, one may start with ϵ close to one and slowly decrease it towards 0 so that the agent begins by pure trial-and-error and acts with more confidence over time, while still collecting information about small deviations from its optimal policy.

6.6 Approximate Reinforcement Learning

In the following, we will stop considering the case of discrete observation spaces (*i.e.* where the input to the network could be represented simply by an integer in $[0, N_{states}]$)), but almost always remain in the case of discrete action space.

The fact that both states and actions are discrete was a major advantage: the Q–function could then be represented by a matrix with dimensions corresponding to state and action. When the states are no-longer discrete[3], this is not possible anymore, and one has to use approximation techniques to allow efficient representation of the policy. The approximation techniques that we will consider, and which proved extremely successful for Reinforcement Learning applied to video games, is simple: use a neural network to represent the policy. The advantage is that, as was the case for classical Deep Learning, the right choice of architecture allows to drastically reduce the number of parameters to tune in the approximator by exploiting intrinsic symmetries of the data. For example, when the observation is an image, it seems reasonable to expect that a Residual Network should be able to correctly encode the optimal policy[4].

Two main families of algorithms exist: Policy Gradient and Deep Q Learning. As we shall see, they are very similar in their implementation but correspond to complementary points of view on the problem.

6.6.1 Policy Gradient methods

For these methods, the output of the Neural Network is the parameter vector of a pre-defined probability distribution. In the following, we only consider the case where the distribution is the multinomial over a discrete set of actions $[1, N]$, so that the output layer will simply be a softmax over N neurons.

A policy π represented in that way is parametrized by the set θ of its network weights, which makes it possible to tune by the following gradient descent procedure, known as the Reinforce algorithm (Williams, 1992):

1. Perform a certain number of trajectories (epochs) by drawing the actions from the probability distribution outputted by the network for the current state. During this step, conserve a history of the log-probabilities of each action and of the obtained rewards r. This step can be thought of as a form of Monte-Carlo evaluation of the average rewards.

2. Along each trajectory, compute the discounted rewards:

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}. \tag{6.9}$$

[3]More precisely, when the cardinality of the input space is too large: the space of $(256, 256)$ images is indeed discrete, however constructing a matrix of 256^3 entries is unreasonable.

[4]While reasonable, it is not completely obvious that a network used for image classification and one used for image-based reinforcement learning could share their weights.

For stability reasons, these rewards are often normalized within each batch of trajectories to mean zero and variance one.

3. Minimize the loss
$$\mathcal{L} = \sum_{trajs} \sum_{t} -R_t \log p_t \tag{6.10}$$
using any kind of Gradient Descent update. This encourages the network to increase the probability of actions that yield positive rewards. An additional entropy term ϵH_t can be added, where $H_t = \sum_{i=1}^{N} p_{i,t} \log p_{i,t}$, in order to penalize convergence to deterministic solutions and encourage exploration.

4. Repeat steps 1 to 3 until convergence.

Such methods are often referred to as being **on-policy**, meaning that the transitions used to compute the loss must have been collected with the policy currently represented by the network. This limitation makes them less sample efficient than their Value Function counterparts, which can reuse multiple times the same transitions, and even transitions obtained using a different strategy than the one represented by the network, for example random actions: these methods are called **off-policy**, and we will study them in more details in the next Section. However, Policy Gradient methods are not necessarily much harder computationally as a single step of update is much simpler and quicker, making them comparable in terms of wall clock time to convergence.

6.6.2 Value Function methods

Policy Gradient methods are designed to increase the likelihood of actions that yield high-rewards, but do not actually build any model of the Markov Decision Process. Value function methods such as Deep Q Learning, on the other hand, do one step in that direction by trying to approximate the Q–function of this policy via Monte Carlo, and using that Q–function to deduce the optimal action in any given state, a very similar procedure to the one described in Section 6.5.

Since the functions are now represented by Neural Networks instead of tables, updates can no longer be done exactly, but instead correspond to one step of Gradient Descent; for additional stability, which is known to be a limiting factor in Deep Reinforcement Learning, the Gradients are not computed on a single transition tuple but rather on a large number of them. To do so, transitions that are observed from the environment are stored in a **replay buffer** (Mnih et al., 2013, 2015), from which a batch of transitions is randomly selected at each step, as illustrated in Figure 6.4; alternatively, some methods instead rely on maintaining several copies of the agents, and letting them interact with the environment and update their policies independently (Mnih et al., 2016). Importantly, and contrary to Policy Gradient methods, the transitions that are used to fill the replay buffer do not need to be obtained "on-policy": Value Function methods can learn from observing other agents, for example humans, interacting with their environment, a field known as imitation learning (Hussein et al., 2017). This also makes them *a priori* more sample efficient, by allowing samples to be reused multiple times, which can be a major benefit in cases where the environment is expensive to run (either because of long computation time, or because actual data has to be collected).

Another influential algorithm which models the value function adopts the Actor-Critic idea (Konda and Tsitsiklis, 2000) of having two different networks, one that is used to predict the Value Function at a given state (the critic), while another (the actor) actually represents the policy. This architecture can be seen as an intermediate situation between

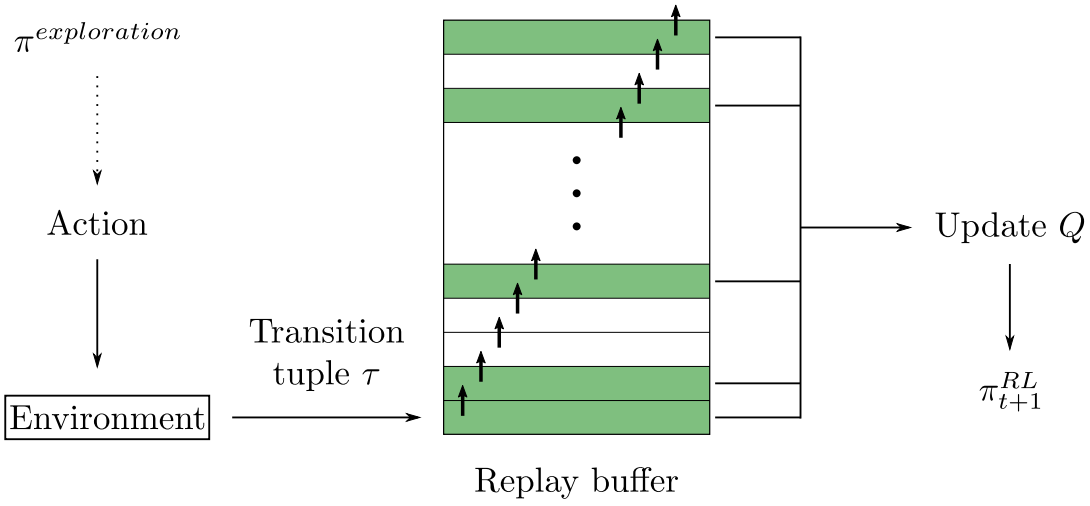


Figure 6.4: Deep Q Network learning scheme. At each time-step, the agent interacts with its environment using an action a_t drawn from an exploration policy $\pi^{exploration}$, which can be arbitrary (for example, with random exploration moves, or incorporating expert supervision); this is in contrast with Policy Gradient methods which requires action be drawn from the policy being currently optimized π^{RL}_t. From this action, a transition tuple is obtained, which is stored in a "first-in, first-out" (meaning that oldest entries get discarded first) *replay buffer* (Mnih et al., 2015). Then, a subset of the currently buffered transitions are randomly selected and used to perform one step of optimization of the Q function, from which the new policy π^{RL}_{t+1} is obtained.

on and off-policy algorithms, and aims at bringing the added stability of Deep Q Learning to the basic Reinforce algorithm.

6.7 State-of-the-art methods

Despite their major empirical successes, all aforementioned methods can be significantly improved. We will now present some recent developments in the field, providing reference to the relevant literature without entering details.

One major limitation of all algorithms described until now is that they do not allow the action space to be continuous. This is an important feature for real-world applications, most notably robotics in which the different motors can be moved to continuous angles; while it is a priori always possible to discretize the action space (*e.g.* by the minimum angular displacement of the motor), each of those actions will be considered as "categorically" different from the others, and the relation between the different actions (in particular the fact that one action can often be reconstructed as the sum of two other actions) will *a priori* not be exploited. Many methods have been developed specifically for continuous action tasks, both on-policy (Trust-Region Policy Optimization (Kakade and Langford, 2002; Schulman et al., 2017a, 2018) and Proximal Policy Optimization (Schulman et al., 2017b; Heess et al., 2017)) and off-policy (Deep Deterministic Policy Gradient and its more stable variant TD3 (Lillicrap et al., 2019; Fujimoto et al., 2018), Soft Actor-Critic (Haarnoja et al., 2018, 2019)).

Another topic of research is the one of accommodating multiple tasks in a single Neural Network, creating what Schaul et al. (2015) refer to as "Universal Value Function Approximators". This is often reformulated as stating that a given environment can have different "rules" or "goals", which the agent is made aware of as part of the observation space, which impact the policy that the agent must adopt; one example that we will consider in Chapter 8 is a spatial navigation task in which the goal is a position within the environment that the agent must reach. In that case, similarly to the problem of learning continuous actions, one could learn one policy for each goal, without taking into consideration the structure relating the different goals. Andrychowicz et al. (2018) propose a new type of experience replay, called the Hindsight Experience Replay, which can be used in combination with any off-policy method to exploit the fact that the agent might be achieving a goal, which is not the one it currently pursues. We used the implementations of TD3 and HER found in Raffin et al. (2021) to train an agent to reach an arbitrary position in a continuous visual environment in Chapter 8 with great success, despite lack of extensive hyperparameter optimization.

Finally, it should be noted that there is no "be all, end all" argument in favor of any particular algorithms in any given situation; instead, it is more often than not a choice based on heuristics and personal preference, or if computing budget allows, another hyperparameter to optimize.

6.8 Examples of environment

Now that we have discussed the different methods used for learning, we will describe some environments on which this can be applied.

First, we introduce a very important toy model, whose simplicity and modularity make it a perfect test bench for Reinforcement Learning: the Grid World. The base of a Grid World is, as suggested by the name, a discrete set of possible states, which can be thought of as tiles on a chess board. An agent is placed in this grid world, and is allowed to interact with it by moving from one tile to another, sometimes with restrictions or special properties. The typical task in this type of environment is that the agent is spawned randomly somewhere on the grid, and has to navigate until it finds an exit.

This model can easily be extended to accommodate a wide variety of tasks, for example:

- optimal path in a cluttered environment: not all movements are allowed, and the agent receives a reward only when exiting.
- planning: the agent is allowed to exit and receive its reward only if it performed a specific action, like going through a specific tile, forcing the agent to adopt a "two-parts" strategy.
- continuous environment simulation: by assigning a "continuous" property to each tile, for example a random image taken in a given room, one can use Grid Worlds to simulate coarse-grain behavior on a non-discrete environment.

The main advantage of these environments is that they are extremely fast to simulate, and require little computational power. Example of such environments are the Mini-Grid (Chevalier-Boisvert et al., 2018) and Gym-Maze projects.

More involved environments usually rely on the use of existing video game engines, such as the one of Atari (Bellemare et al., 2013), Minecraft (Johnson et al., 2016), Doom (Wydmuch et al., 2018) or Starcraft (Vinyals et al., 2017). More general-purpose engines

have also been made accessible for Machine Learning research, such as the Unity framework (Ward et al., 2020) which has recently been used by OpenAI to create the XLand environment (Team et al., 2021), which combines procedural world and rules generation to generate a wide variety of different "games" on which a single network is trained to try and generate reusable behaviors. Others instead rely on actual physical simulations, for example using the MuJoCo control suite (Todorov et al., 2012). It should be noted that most environments published now follow the specifications laid out by OpenAI's Gym software (Brockman et al., 2016), allowing for the development of projects such as Stable-Baselines (Raffin et al., 2021) that regroup implementations for many state-of-the-art algorithms, allowing for fast and easy prototyping on new problems. The environment we developed for Chapter 8 follows the Gym specification.

Finally, some real-world applications of Reinforcement Learning, not relying on software environments but rather on a set of actuators and sensors also exist. One of the earliest examples came from the field of robotics in the 1950s, with Marvin Minsky's Stochastic Neural Analog Reinforcement Calculator, a physical implementation of Hebbian Learning with 40 synapses, and more recently similar projects have been focusing on autonomous driving of miniature cars (Balaji et al., 2019).

Chapter 7

Low-Dimensional manifolds in Recurrent Neural Networks

Abstract

We study the learning dynamics and the representations emerging in Recurrent Neural Networks trained to integrate one or multiple temporal signals. Combining analytical and numerical investigations, we characterize the conditions under which an RNN with n neurons learns to integrate $D(\ll n)$ scalar signals of arbitrary duration. We show, for linear, ReLU and sigmoidal neurons, that the internal state lives close to a D-dimensional manifold, whose shape is related to the activation function. Each neuron therefore carries, to various degrees, information about the value of all integrals. We discuss the deep analogy between our results and the concept of *mixed selectivity* forged by computational neuroscientists to interpret cortical recordings. Finally, we introduce a general loss function, adaptable to any non-linearity in the neural activation, that can be used to train networks without evaluating the network on any input data.

This Chapter is a reproduction of the original manuscript of the article "Low-Dimensional Manifolds Support Multiplexed Integrations in Recurrent Neural Networks" by Arnaud Fanthomme and Rémi Monasson, published in Neural Computation and available here.

7.1 Introduction

Recurrent neural networks (RNNs) have emerged over the past years as a versatile and powerful architecture for supervised learning of complex tasks from examples, involving in particular dynamical processing of temporal signals (Chung et al., 2014). Applications of RNNs or of their variants designed to capture very long-term dependencies in input sequences through gating mechanisms, such as GRU or LSTM, are numerous and range from state-of-the-art speech recognition networks (Amodei et al., 2016) to protein sequence analysis (Almagro Armenteros et al., 2017).

How these tasks are actually learned and performed has been extensively studied in the Reservoir Computing setup where the recurrent part of the dynamics is fixed, see (Tanaka et al., 2019) for a review, while the general case of RNNs remains mostly an open question. Understanding of those networks would bring valuable advantages to both neuroscience and machine learning, as suggested in (Barak, 2017; Richards et al., 2019). Some results have been recently obtained, when the representations and the dynamics are low dimensional (Sussillo and Barak, 2012; Mastrogiuseppe and Ostojic, 2018; Schuessler et al., 2020a,b), a prominent feature of the Neural Integrators that are the focus of the present study. Neural integrators, whose function is to perform integration of time-series, have been studied for several decades in neuroscience, both experimentally (Robinson, 1989, 1968; Wong and Wang, 2006; Aksay et al., 2007) and theoretically (Elman, 1990, 1991; Seung, 1996; Lee et al., 1997), and more recently, numerically, in the context of machine learning (Song et al., 2016).

The goal of our study is three-fold. First, we want to study how exactly the task of integration is learned from examples by RNNs. We derive rigorous results for linear RNNs and approximate ones for non-linear RNNs, which can be compared to numerical simulations. This approach is similar to the one adopted by (Saxe et al., 2014) for the case of deep networks, and more recently by (Schuessler et al., 2020b) in the case of recurrent networks. Second, we seek to understand the nature of the internal representations built during learning, an issue of general interest in neural network-based learning (Li et al., 2017; Zhang and Zhu, 2018; Montavon et al., 2018; Olah et al., 2018). It is, in particular, tempting to compare representations emerging in artificial neural networks to their natural counterparts in computational neuroscience. Third, we do not limit ourselves to a single integration, but consider the issue of learning multiple integrators within a unique network. While we do not expect an increase in performance for each individual task, as was observed in the case of Natural Language Processing by (Luong et al., 2016), we are interested in finding representations adequate for parallel computations within a single network, allowing for considerations on the topic of "mixed selectivity" developed in computational neuroscience and studied by (Rigotti et al., 2013). The issue of network capacity, the maximum number of tasks that can be performed in parallel, has been previously studied numerically by (Collins et al., 2017), but remains out of the scope of this study, which will focus on a number D of integrals small compared to the network size.

Our paper is organized as follows. We define the RNNs we consider in this work, the integration task and the training procedure in Section 7.2. The case of linear activation function is studied in detail in Section 7.3. RNNs with non-linear activation functions are studied in Section 7.4 in the case of a single channel ($D = 1$), while our results for the general situation of multiple channels ($D \geq 2$) are presented in Section 7.5. Conclusions and perspectives can be found in Section 7.6. The paper is complemented by a series of Appendices containing details about the calculations, simulations and further figures. The source code for the simulations can be found on GitHub.

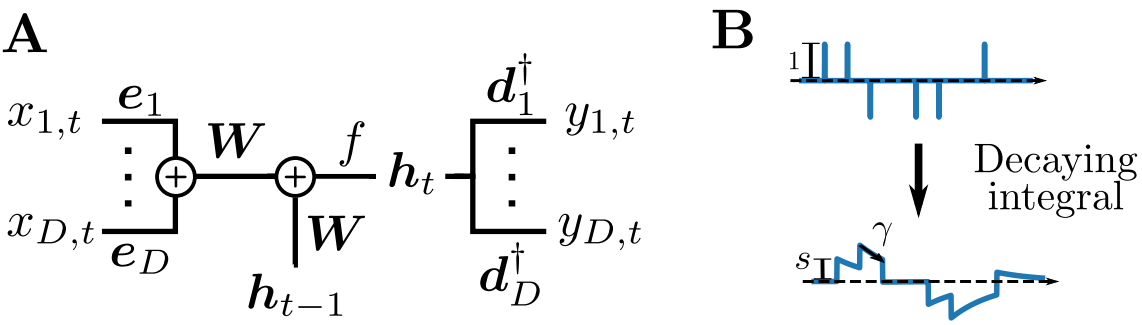


Figure 7.1: **A**: Multiplexed Recurrent Neural Network, with D input channels (left) and the same number of output channels (right). The internal state of the RNN, $\boldsymbol{h}_t$, is a vector of dimension n. The inputs are encoded by the vectors $\boldsymbol{e}_c$ and decoded from the internal state through the decoder weights $\boldsymbol{d}_c$. **B**: Illustration of the decaying integral mapping that we want networks to approximate, on a sparse input sequence. At each time-step t, if the input time-series x_t is non-zero, the integral is increased by $s\,x_t$; then, it is multiplied by $\gamma < 1$, which produces the exponential decay in absence of inputs. In practice, sequences used for experiments were Gaussian noise, and these sparse sequences are used only for visualization.

7.2 Definitions and model

Description of the network. We consider a single-layer RNN of size n, without any gating mechanism; while such refinements are found to improve performance, as reviewed by (Lipton et al., 2015), we omit them as they are not necessary for such a simple task. The computation diagram presented in Figure 7.1A can be summed up as follows: at time t, the scalar inputs along all channels $c = 1..D$, denoted $x_{c,t}$, are multiplied by their respective **encoder** vectors $\boldsymbol{e}_c$; these vectors are summed to the previous internal state $\boldsymbol{h}_{t-1}$[1], and multiplied by the **weight matrix** $\boldsymbol{W}$ before a componentwise **activation** f is applied to get the new internal state $\boldsymbol{h}_t$. The update equation for $\boldsymbol{h}$ is therefore

$$\boldsymbol{h}_t = f\left(\boldsymbol{\nu}_t\right) \ , \tag{7.1}$$

where the current[2] $\boldsymbol{\nu}_t$ is defined through

$$\boldsymbol{\nu}_t = \boldsymbol{W} \cdot \left[\boldsymbol{h}_{t-1} + \sum_{c=1}^{D} x_{c,t}\,\boldsymbol{e}_c\right] \ . \tag{7.2}$$

The output units are linear: their values $y_{c,t}$ are simply obtained by taking the scalar product of $\boldsymbol{h}_t$ and the **decoder** vectors $\boldsymbol{d}_c$,

$$y_{c,t} = \boldsymbol{d}_c \cdot \boldsymbol{h}_t \ . \tag{7.3}$$

Most of this study will be focused on two different activation functions f: the "linear" activation, which is simply the identity, and the ReLU non-linearity, which takes component-wise maximum of a vector and zero. Linear activation allows for exact results to be derived on both the learning dynamics and the structure of solutions, while the choice

[1] We initialize the internal state before any input to $\boldsymbol{h}_{-1} = \boldsymbol{0}$.

[2] This name is chosen in analogy with computational neuroscience, where W_{ij} is a synaptic weight from neuron j to neuron i, and the image of the activity vector, $\nu_i = \sum_j W_{ij} h_j$, represents the total current from the recurrent population coming onto neuron i.

of ReLU will serve as an example of non-linear activation that can be used to create perfectly generalizing integrators (at least in the $D = 1$ case), and show that the conclusions of the linear network study remain relevant. Finally, we propose a generic procedure to train an RNN with arbitrary non-linearity f to perform multiplexed integrations, which we illustrate with success in the case of sigmoidal activation.

Description of the task. The networks will be trained to map D input time-series $(x_{c,t})_{t\in\mathbb{N}}$ to D output ones $(y_{c,t})_{t\in\mathbb{N}}$: for all channels $c = 1, ..., D$, the c-th output should match the γ_c-discounted sum of the c-th channel inputs, times the scale factor s_c:

$$\overline{y}_{c,t} = s_c \sum_{k=0}^{t} \gamma_c^{k+1} \, x_{c,t-k} \ , \tag{7.4}$$

see Figure 7.1B. The values of the decay constants γ_c are chosen in $[0, 1]$ to restrict memory to recent events and avoid instabilities[3].

We quantify the performance of the network through the mean square error between the actual and target outputs, across the D channels, computed on training input sequences of length (duration) T:

$$\mathcal{L} = \left\langle \sum_{c=1}^{D} \sum_{t=0}^{T-1} \left(y_{c,t} - \overline{y}_{c,t}\right)^2 \right\rangle_X . \tag{7.5}$$

Description of the learning procedure. Except when otherwise specified, the encoder $\mathbf{e}$ and decoder $\mathbf{d}$ will be considered as randomly fixed at network initialization, and forced to be of unit norm. The reason for this hypothesis is two-fold. First, our focus of interest is how the network of connections between neurons evolves during training and the nature of the solutions and representations obtained. The simplified setup allows for deeper mathematical analysis of the dynamics of the $\mathbf{W}$ than the general case, where all parameters of the network evolve simultaneously during training. Second, while the speed of convergence is positively impacted by relaxing the constraint of fixing the decoder, numerical experiments indicate that the nature of the $\mathbf{W}$ network is qualitatively unchanged if $\mathbf{e}$ and $\mathbf{d}$ are also trained, in particular when it comes to the way the integrals are represented.

For theoretical analysis, we train the recurrent weights $\boldsymbol{W}$ using Gradient Descent (GD) updates at learning rate η:

$$W_{ij}^{(\tau+1)} = W_{ij}^{(\tau)} - \eta \, \frac{\partial \mathcal{L}}{\partial W_{ij}}(\boldsymbol{W}^{(\tau)}) \ , \tag{7.6}$$

where τ is the discrete learning time. We also performed experiments using the non-linear Adam optimizer (Kingma and Ba, 2017) to ensure robustness of our results with respect to the specific choice of optimization procedure. Numerical implementations were performed in Python, making extensive use of the SciPy (Virtanen et al., 2020) and PyTorch (Paszke et al., 2019) packages respectively for scientific computing and implementation of Automatic Differentiation and Gradient Descent optimization.

[3] If γ is chosen too close to 1, the network might during training have an effective "decay" larger than 1; in that case, the values of the outputs and the associated gradients become large (in particular when training on long input sequences), which can then be overcompensated and make the training divergent.

7.3 Case of linear activation

Throughout this section, we assume that the activation function f is linear. We start with the simplest case of a single channel ($D = 1$), and will omit the subscript $c = 1$ below for simplicity; the case of multiple channels $D \geq 2$ will be studied in Section 7.3.4.

As the network dynamics $\boldsymbol{h}_t \to \boldsymbol{h}_{t+1}$ is linear, the loss (7.5) can be analytically averaged over the input data distribution. The computation is presented in Appendix A.1, and yields:

$$\mathcal{L}(\boldsymbol{W}) = \sum_{q,p=1}^{T} \boldsymbol{\chi}_{qp}(\mu_q - s\gamma^q)(\mu_p - s\gamma^p), \tag{7.7}$$

where

$$\mu_q = \boldsymbol{d}^\dagger \boldsymbol{W}^q \boldsymbol{e} \tag{7.8}$$

will be hereafter referred to as the q-th moment of $\boldsymbol{W}$, and $\boldsymbol{\chi}$ is a positive-definite matrix, related to the covariance matrix of the inputs x_t.

The average loss implicitly depends on γ, T, s, $\boldsymbol{e}$, $\boldsymbol{d}$ and input correlations $\boldsymbol{\chi}$, which do not evolve during training and are therefore omitted from the argument. Since $\boldsymbol{\chi}$ is positive-definite, the global minimum of the loss is reached when the moments of $\boldsymbol{W}$ fulfill

$$\mu_q = s\gamma^q \ , \tag{7.9}$$

for all $q = 1, \ldots, T$. The same conditions are obtained for uncorrelated inputs, so we will restrict to this case for numerical investigations in the following.

The gradient of the averaged loss with respect to the weight matrix $\boldsymbol{W}$ can be computed (see Appendix A.2), with the result

$$\frac{\partial \mathcal{L}}{\partial W_{ij}} = 2 \sum_{q,p=1}^{T} \boldsymbol{\chi}_{qp} \, (\mu_q - s\gamma^q) \sum_{m=0}^{p-1} \sum_{\alpha=1}^{n} d_\alpha (W^m)_{\alpha i} \sum_{\beta=1}^{n} (W^{p-1-m})_{j\beta} \, e_\beta \ . \tag{7.10}$$

We emphasize that, while the network update dynamics is linear, the training dynamics over $\boldsymbol{W}$ defined by (7.6) and (7.10) is highly non-linear.

7.3.1 Conditions for generalizing integrators

Conditions (7.9) over the moments μ_q, with $q = 1, ..., T$, ensure that the RNN will perfectly integrate input sequences of length up to the epoch duration T. We call **generalizing integrator (GI)** an RNN such that these conditions are satisfied for **all** integer-valued q, ensuring perfect integration of input sequences of arbitrary length.

We will now derive a set of sufficient and necessary conditions for a diagonalizable matrix $\boldsymbol{W}$ to be a GI[4]. Let us assume $\boldsymbol{W}$ is diagonalized as $\boldsymbol{P}\boldsymbol{\Lambda}\boldsymbol{P}^{-1}$, where the spectral matrix $\boldsymbol{\Lambda} = \mathrm{diag}(\boldsymbol{\lambda})$ is diagonal and $\boldsymbol{P}$ is invertible, of inverse $\boldsymbol{P}^{-1}$. The moments of $\boldsymbol{W}$ can be expressed from the eigenvalues as follows:

$$\mu_q = \boldsymbol{d}^\dagger \boldsymbol{P} \boldsymbol{\Lambda}^q \boldsymbol{P}^{-1} \boldsymbol{e} = \sum_{i=1}^{n} g_i \lambda_i^q \quad \text{with} \quad g_i = (\boldsymbol{P}^\dagger \boldsymbol{d})_i (\boldsymbol{P}^{-1} \boldsymbol{e})_i \ . \tag{7.11}$$

[4] As the set of diagonalizable matrices is dense in the space of matrices, any non-diagonalizable matrix $\boldsymbol{W}$ can be made diagonalizable through the addition of an infinitesimal matrix; the moments of the resulting matrix are arbitrarily close to the ones of $\boldsymbol{W}$, which makes our results for diagonalizable matrices directly applicable to $\boldsymbol{W}$, see Section 7.3.3.

Obviously, a null eigenvalue does not contribute to the above sum, hence the conditions that we obtain in the following will only apply to non-zero eigenvalues. Our condition for null loss is that all the aforementioned moments μ_q are equal to $s\,\gamma^q$.

The above set of conditions can be rewritten as follows. For any real-valued polynomial $Q(z)$ of degree less than, or equal to T in z, such that $Q(0) = 0$, we have

$$\sum_i g_i\, Q(\lambda_i) = s\, Q(\gamma)\ . \tag{7.12}$$

We can evaluate the previous equality for well-chosen polynomials. Let us consider one eigenvalue, say, λ_κ assumed to be different from γ, and the Lagrange Polynomial $Q(z)$ equal to one for $z = \lambda_\kappa$ and to 0 for $z = 0$, $z = \lambda_i \neq \lambda_\kappa$ and $z = \gamma$. Such a polynomial exists as soon as $T \geq n+1$ in the general case where all eigenvalues are distinct from each other, 0, γ, and as soon as $T \geq r+1$ if $n-r$ eigenvalues are equal *e.g.* to 0. Equality (7.12) gives:

$$\sum_i g_i\, \delta_{\lambda_i,\lambda_\kappa} = 0\ , \tag{7.13}$$

where $\delta_{\cdot,\cdot}$ denotes the Kronecker delta. Therefore, any eigenvalue different from γ must satisfy an exact cancellation condition for the associated g coefficients, ensuring that it does not contribute to the network output. Similarly, a condition for the γ eigenvalue can be written, to ensure that an input of magnitude 1 entails a change of magnitude s in the output.

The necessary and sufficient conditions for a diagonalizable matrix $\boldsymbol{W}$ to be a global minimum of the loss defined with $T \geq n+1$ therefore read

$$\begin{cases} & \sum_i g_i\, \delta_{\lambda_i,\gamma} = s \\ \forall \kappa \text{ s.t. } \lambda_\kappa \notin \{\gamma, 0\}, & \sum_i g_i\, \delta_{\lambda_i,\lambda_\kappa} = 0 \end{cases} \tag{7.14}$$

These conditions are in turn enough to guarantee that $\boldsymbol{W}$ is a global minimum of the loss for any value of T, hence the Generalizing Integrators and the minima of the losses defined with $T \geq n+1$ are equal.

Clearly, any global minimum of the averaged loss $\mathcal{L}$ experimentally obtained when using training sequences of length $T \geq n+1$ is a GI. Networks trained with much shorter epochs can also be GIs if the rank of $\boldsymbol{W}$ remains small enough throughout the training dynamics. More precisely, if we assume we have found a minimum of the loss of rank $r \leq n$, it will be a GI as soon as $T \geq r+1$. An important illustration is provided by the null initialization of the weights ($\boldsymbol{W}^{(\tau=0)} = \boldsymbol{0}$), which ensures that $\boldsymbol{W}$ remains of rank $r = 2$ at all times τ, see (7.10) and next subsection.

7.3.2 Special case of null-weight initialization

We now assume that the weight matrix $\boldsymbol{W}$ is initially set to zero, and characterize all the GIs accessible through GD, as well as the local convergence to those solutions. A study of the full training dynamics for two special cases ($T = 1$ and $\boldsymbol{e} = \boldsymbol{d}$) can be found in Appendix A.3.

Low-rank parametrization. From the expression of the gradients (7.10) and the linearity of the weight updates (7.6), it is clear that starting from $\boldsymbol{W} = \boldsymbol{0}$, the weight matrix will remain at all times τ in the subspace generated by the four rank-1 matrices $\boldsymbol{d}\boldsymbol{d}^\dagger, \boldsymbol{d}\boldsymbol{e}^\dagger, \boldsymbol{e}\boldsymbol{d}^\dagger, \boldsymbol{e}\boldsymbol{e}^\dagger$. We introduce an orthonormal basis for the $\boldsymbol{v}_1 \equiv \boldsymbol{e}, \boldsymbol{v}_2 \equiv \boldsymbol{d}$ space,

$$\overline{\boldsymbol{v}_a} = \sum_{b=1}^{2} (\boldsymbol{\Sigma}^{-1/2})_{ab}\, \boldsymbol{v}_b, \quad \text{with} \quad \boldsymbol{\Sigma}_{ab} = \boldsymbol{v}_a^\dagger \boldsymbol{v}_b\ , \tag{7.15}$$

and the corresponding parametrization of the subspace spanned by $\boldsymbol{W}$:

$$\boldsymbol{W}^{(\tau)} = \sum_{a,b=1}^{2} \omega_{ab}^{(\tau)} \, \overline{\boldsymbol{v}_a} \, \overline{\boldsymbol{v}_b}^{\dagger} \ , \tag{7.16}$$

where $\omega^{(\tau)}$ is a 2×2-matrix.

Generalizing integrators. Conditions (7.14) for $\boldsymbol{W}$ to be a GI can be turned into conditions over ω, see Appendix A.4. Let us assume that ω is diagonalized through $\omega = \boldsymbol{P}_\omega \boldsymbol{\Lambda}_\omega \boldsymbol{P}_\omega^{-1}$ with $\boldsymbol{\Lambda}_\omega = \mathrm{diag}(\lambda_1, \lambda_2)$, and define

$$g_i = (\boldsymbol{P}_\omega^{\dagger} \sqrt{\boldsymbol{\Sigma}})_{i,1} (\boldsymbol{P}_\omega^{-1} \sqrt{\boldsymbol{\Sigma}})_{i,2}.$$

The conditions for ω to define a GI through (7.16) are:

- $\lambda_i = \gamma$ for $i = 1$ or 2,
- $\sum_i \delta_{\gamma,\lambda_i} g_i = s$,
- $g_i = 0$ if $\lambda_i \notin \{0, \gamma\}$.

Taking into account the constraint $g_1 + g_2 = \boldsymbol{\Sigma}_{1,2} = \boldsymbol{d}^{\dagger}\boldsymbol{e}$, we find that the set of GIs is spanned by the following three manifolds in the 4-dimensional space of ω matrices, see Appendix A.5 for details:

- The first manifold is of dimension 2, and contains rank-1 integrators $\boldsymbol{W}$ at all scales. These weight matrices have one eigenvalue equal to γ, and the other to 0 so that one of the g coefficients remains unconstrained:

$$\omega = \frac{\gamma}{\beta - \alpha} \begin{pmatrix} \beta & -1 \\ \alpha\beta & -\alpha \end{pmatrix} \ , \tag{7.17}$$

  where $(\alpha, \beta) \in \mathbb{R}^2$. Fixing the scale s to any value different from $\boldsymbol{d}^{\dagger}\boldsymbol{e}$ introduces exactly one relation between α and β, making the set of rank-1 perfect integrators at scale s a 1–dimensional manifold, see Appendix A.5.

- The other two manifolds contain rank-2 integrators, operating at the scale $s^* = \boldsymbol{d}^{\dagger}\boldsymbol{e}$ only. For generic independent encoder and decoder vectors, the scale $s^* = \boldsymbol{d}^{\dagger}\boldsymbol{e}$ is of the order of $n^{-1/2}$ and vanishes in the large size limit. We will discard these solutions, and focus on rank-1 solutions given by (7.17) at finite scale s ($\neq \boldsymbol{d}^{\dagger}\boldsymbol{e}$).

The structure of the GI manifolds is sketched in Figure 7.2.

In Appendix A.6, we compute the gradient and Hessian of the loss in the subspace of weight matrices reachable from null initialization. In the case of fixed encoder and decoder, the convergence towards a GI is generically exponentially fast; the corresponding decay time can be minimized by appropriately choosing the value s of the scale s, see Appendix A.7. For some specific choices of the scale s, convergence can be much slower and exhibit an algebraic behavior, see Appendix A.8.

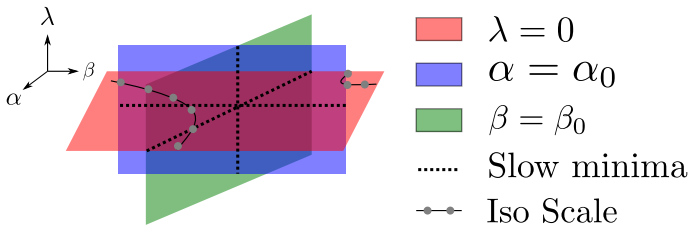


Figure 7.2: Illustration of the three GI manifolds in the space of 2×2–matrices with one eigenvalue equal to γ, the second to λ, and the remaining two degrees of freedom being labeled α and β. In one manifold (red), the second eigenvalue is zero, so that all of those matrices are GIs with decay γ, and any scale s. The other two manifolds contain integrators at the particular scale $s^* = \boldsymbol{d}^\dagger \boldsymbol{e}$ only, and are of rank 2. The values of α_0 and β_0 are computed in Appendix A.5, where details on the parametrization used here can also be found.

7.3.3 Initialization with full rank connection matrices

The results above assumed that training started from a null weight matrix, in order to constrain the dynamics of $\boldsymbol{W}$ to a very low-dimensional space. Training RNNs on very short epochs ($T = 3$) was then sufficient to obtain rank-1 GIs capable of integrating arbitrary long input sequences.

In practice, we observe that initializing the network with a matrix $\boldsymbol{W}$ of small spectral norm (instead of being strictly equal to zero) does not change the fact that only one of the eigenvalues of $\boldsymbol{W}$ is significantly altered during training, and a GI is obtained as soon as $T \geq 3$. The use of a non-linear optimization scheme such as Adam rather than GD does not change this observation.

To gain insights about this empirical result, we consider a perturbation $\boldsymbol{\epsilon} = \sum_i \epsilon_i \boldsymbol{u}_i \boldsymbol{v}_i$, with singular values bounded by 1, around a generalizing integrator of rank 1, $\boldsymbol{W} = \sigma \boldsymbol{l} \boldsymbol{r}^\dagger$. Under the assumption that the $\boldsymbol{u}$ and $\boldsymbol{v}$ vectors are drawn randomly on the unit sphere of dimension n, their dot products with $\boldsymbol{e}$, $\boldsymbol{d}$ and each other are realizations of a centered Gaussian distribution of variance $1/n$. We can then consider the image of $\boldsymbol{e}$ by our perturbed matrix:

$$(\boldsymbol{W} + \boldsymbol{\epsilon})\boldsymbol{e} = (\sigma \boldsymbol{r}^\dagger \boldsymbol{e})\boldsymbol{l} + \sum_{i=1}^{n} \epsilon_i (\boldsymbol{v}_i^\dagger \boldsymbol{e}) \boldsymbol{u}_i. \tag{7.18}$$

The second term, originating from the perturbation, is a vector whose components are sums of n terms of unfixed signs and magnitudes $1/n$, and is, hence, of the order of $1/\sqrt{n}$. Accordingly, the dot product of this perturbation vector with $\boldsymbol{d}$, which is exactly the perturbation to the first moment μ_1, will be of the order of $1/\sqrt{n}$ too. Under similar hypotheses of independence of Gaussian vectors, all moments μ_q will be perturbed by terms of that same order.

Since unstructured eigenvectors do not contribute to the network output at first order, the gradients with respect to those parameters will also be subleading and this perturbation will remain mostly unchanged during training, in agreement with numerical simulations.

7.3.4 Case of multiple channels

We have seen that GD is generally able to train a linear RNN exponentially fast towards a rank–1 single-channel GI with associated eigenvalue γ and singular vectors tuned to ensure the correct scale of integration. The state of the corresponding network is easily interpretable: it is, at all times, proportional to the output integral. Due to the linearity of

the network, this result can be straightforwardly extended to the case of $D > 1$ integration channels, as we show below.

Interpretation of rank–1 solutions in the single channel case. We write the rank–1 GI as $\boldsymbol{W} = \sigma \boldsymbol{l}\boldsymbol{r}^\dagger$, where $\boldsymbol{l}$ and $\boldsymbol{r}$ are normalized to 1, and σ is positive. Since $\boldsymbol{W}$ must have γ as its eigenvalue, we need $\sigma \boldsymbol{r}^\dagger \boldsymbol{l} = \gamma$. Additionally, to ensure that the first non-zero input gives the correct output, we require that $\sigma(\boldsymbol{d}^\dagger \boldsymbol{l})(\boldsymbol{r}^\dagger \boldsymbol{e}) = s\gamma$. It is easy to check that these conditions are sufficient to ensure that the state of the network is

$$\boldsymbol{h}_t = a\, \overline{y}_t\, \boldsymbol{l} \qquad \text{with} \qquad a = \frac{1}{\boldsymbol{d}^\dagger \boldsymbol{l}}\ , \tag{7.19}$$

at all times t, which in turn ensures perfect integration ($y_t = \overline{y}_t$). In other words, rank–1 GIs rely on a linear, one-dimensional representation of the target integral: the internal state is, at all times, proportional to $\overline{y}_t$.

Representation of integrals with multiple channels. The above discussion of the single-channel case generalizes to multiple channels. Through training a weight matrix $\boldsymbol{W}$ of rank D is constructed, which has $(\gamma_1, ..., \gamma_D)$ as its eigenvalues, and singular vectors compatible with the (fixed) encoder and decoder weight vectors. The GI conditions are as follows:

$$\begin{cases} \forall c \in [\![1, D]\!], & \sigma_c \boldsymbol{r}_c^\dagger \boldsymbol{l}_c = \gamma_c \\ \forall c \in [\![1, D]\!], & \sigma_c\, (\boldsymbol{d}_c^\dagger \boldsymbol{l}_c)\, (\boldsymbol{r}_c^\dagger \boldsymbol{e}_c) = s_c\, \gamma_c \\ \forall (c, c') \in [\![1, D]\!]^2, c \neq c', & \boldsymbol{r}_c^\dagger \boldsymbol{e}_{c'} = 0 \\ \forall (c, c') \in [\![1, D]\!]^2, c \neq c', & \boldsymbol{d}_c^\dagger \boldsymbol{l}_{c'} = 0 \\ \forall (c, c') \in [\![1, D]\!]^2, c \neq c', & \boldsymbol{r}_c^\dagger \boldsymbol{l}_{c'} = 0 \end{cases} \tag{7.20}$$

The first two conditions are exactly the same as in the single channel case, while the last three ensure that the modes of the weight matrix coding for the different integration channels c do not interfere, and can independently update the values of their outputs to match the targets $\overline{y}_{c,t}$.

Assuming these conditions are satisfied, the network state is at time t equal to

$$\boldsymbol{h}_t = \sum_{c=1}^{D} a_c\, \overline{y}_{c,t}\, \boldsymbol{l}_c\ , \tag{7.21}$$

where the a_c's are structural coefficients, which generalizes expression (7.19) to the case $D \geq 2$. The state of any neuron i is therefore a linear combination of the D integrals across the multiple channels. Multiplexing is here possible as long as $D \leq n$, and encoders and decoders each form free families of $\mathbb{R}^n$.

7.4 Non-linear activation: case of a single channel

We now turn to the case of non-linear activation. The computation of the averaged loss is not analytically feasible any longer. However, by investigating RNNs trained with Gradient Descent on the mean square error (7.5) computed on batches of inputs, hereafter referred to as batch–SGD, we have identified structural and dynamical properties, from which sufficient conditions for generalization can be constructed.

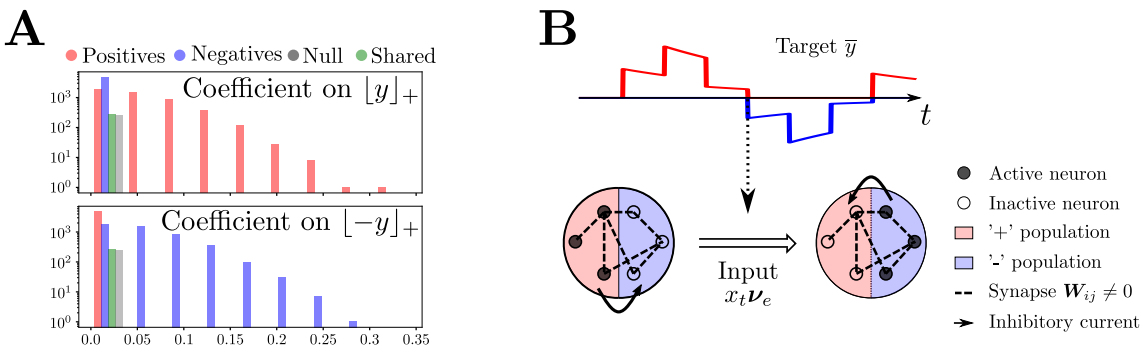


Figure 7.3: Internal encoding of the integral y_t by a single-channel ReLU network using two populations. **A**: Experimentally observed distributions of the components of $\boldsymbol{L}_\pm$, determined by fitting the activity of each neuron with (7.22). Results are aggregated across 10 realizations of batch–SGD training $n = 1000$, $s = 1$. **B**: Illustration of the activity shift from the $+$ to the $-$ population at arrival of an input that changes the sign of the target. Mutual inhibition between the two sub-networks guarantees only one can be active at a given time, and an external input is required to perform the shift.

7.4.1 Empirical study of neural representations in a ReLU network

We start by considering the case of the ReLU activation, where $f = \lfloor \cdot \rfloor_+ = \max(\cdot, 0)$ is a non-linear component-wise operator. The simple encoding (7.19) adopted by linear-activation networks relied on the fact that the activity of each neuron could change sign with $\overline{y}_t$. This is not possible with ReLU activation anymore since activities are forced to remain non-negative, and a novel encoding is obtained after training of the RNNs that we expose below.

Behavior of neuron activities. Based on numerical simulations reported in Figure 7.3A, we argue that the population activity in ReLU networks depends on two vectors, referred to as $\boldsymbol{L}_+$ and $\boldsymbol{L}_-$, with non-negative components and dot products with $\boldsymbol{d}$ equal to, respectively, $+1$ and -1. More precisely, these vectors determine how the neural activities vary with the integral $\overline{y}_t$, depending on its sign:

$$\boldsymbol{h}_t = \lfloor \overline{y}_t \rfloor_+ \boldsymbol{L}_+ + \lfloor -\overline{y}_t \rfloor_+ \boldsymbol{L}_- \ . \tag{7.22}$$

Hence, in the space of possible internal states $\mathbb{R}^n_+$, the state $\boldsymbol{h}$ of the RNN lies in the union of the two half lines along $\boldsymbol{L}_+$ and $\boldsymbol{L}_-$, a 1-dimensional piecewise linear manifold whose geometry is imposed by the non-linear activation $\lfloor \cdot \rfloor_+$.

The n components of $\boldsymbol{L}_+, \boldsymbol{L}_-$ define *a priori* four sub–populations: if $(L_+)_i > 0$ and $(L_-)_i > 0$ neuron i is active at all times t ("shared" population); if $(L_+)_i > 0$ and $(L_-)_i = 0$ (respectively, $(L_+)_i = 0$ and $(L_-)_i > 0$), the neuron is only active when the integral is positive (resp. negative), defining the "+" (resp. "-") population; if $(L_+)_i = (L_-)_i = 0$, the neuron is never active and belongs to the "null" population. In numerical experiments, the shared and null populations account for a small fraction of the neurons (around 5%, see Figure 7.3A) when training is performed using batch–SGD; in addition, shared neurons never have strong activities and their contributions to the output integral seem irrelevant. We introduce in equation (7.31) a new loss function, which allows for training of perfect integrators that do not exhibit any shared or null neurons.

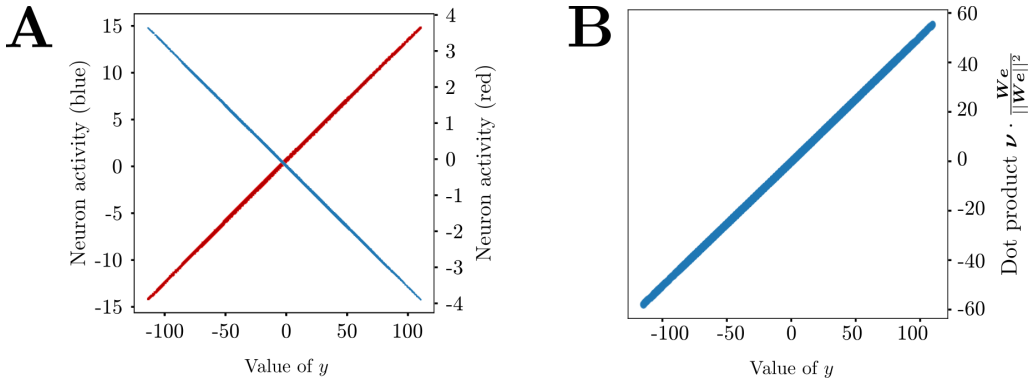


Figure 7.4: Behavior of currents in a ReLU network trained using the batch–SGD loss, $s = 2$, $\gamma = 0.995$. **A**: Parametric plot of the currents $(\boldsymbol{\nu}_t)_i$ incoming on two representative neurons i (red, blue) vs. target integral y_t across time t. We observe a linear relation, with a slope that varies both in sign and magnitude from neuron to neuron. **B**: Normalized dot product between the vector of currents $\boldsymbol{\nu}$ and the image of the encoder $\boldsymbol{We}$ vs. value of the integral, illustrating eqns. (7.23) and (7.26).

Behavior of neuron currents. Numerical experiments furthermore indicate that the dependence of the current $\boldsymbol{\nu}_t$ (7.2) on the integral y_t is simpler than the one shown by the activity $\boldsymbol{h}_t$. We observe that the current vector is proportional to the integral,

$$\boldsymbol{\nu}_t = y_t \, \boldsymbol{L}, \tag{7.23}$$

where the components L_i of the vector $\boldsymbol{L}$ vary from neuron to neuron, both in amplitude and in sign, see Figure 7.4A.

The representation of the integral based on two non-overlapping populations reported above may be seen as a straightforward consequence of the linear encoding at the level of pre-activation currents expressed by (7.23):

$$\boldsymbol{h}_t = \lfloor \boldsymbol{\nu}_t \rfloor_+ = \lfloor y_t \, \boldsymbol{L} \rfloor_+ = \lfloor y_t \rfloor_+ \, \lfloor \boldsymbol{L} \rfloor_+ + \lfloor -y_t \rfloor_+ \, \lfloor -\boldsymbol{L} \rfloor_+, \tag{7.24}$$

from which we deduce that the population vectors $\boldsymbol{L}_+$ and $\boldsymbol{L}_-$ defined in (7.22) are equal to, respectively, $\lfloor \boldsymbol{L} \rfloor_+$ and $\lfloor -\boldsymbol{L} \rfloor_+$. In other words, neurons i encode positive or negative values of the integral depending on the signs of the components L_i.

Hence, while the neural state $\boldsymbol{h}_t = f(\boldsymbol{\nu}_t)$ of a ReLU RNN is not proportional to the integral value, see (7.22), as was the case for linear RNNs in Section 7.3.3, proportionality is recovered at the level of the pre-activation currents $\boldsymbol{\nu}_t$. We will see below that the linearity of the currents with respect to the integrals extends to the case of multi-channel integrators.

7.4.2 Theoretical analysis of the ReLU integrators

We now explain the origin of the linear relationship between current and integral values (7.23), and how the vector $\boldsymbol{L}$ defining the current direction is related to the connection matrix $\boldsymbol{W}$, the encoder $\boldsymbol{e}$, and the parameters s, γ.

Sufficient conditions for integration. Let us first consider the network at time $t = 0$, with all activities set to zero ($\boldsymbol{h}_0 = \boldsymbol{0}$). As the first input x_1 is read by the encoder, the current vector at time $t = 1$ takes value

$$\boldsymbol{\nu}_1 = \boldsymbol{W}(\boldsymbol{0} + x_1 \, \boldsymbol{e}) = x_1 \, \boldsymbol{We} = \frac{\bar{y}_1}{s\gamma} \, \boldsymbol{We} \; . \tag{7.25}$$

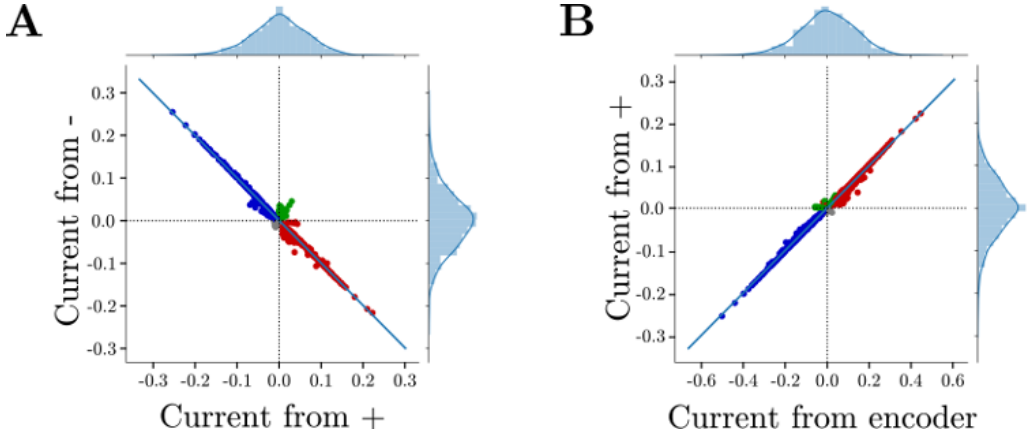


Figure 7.5: Contributions to the currents in a ReLU integrator trained with batch–SGD. Left: scatter plot of $\boldsymbol{W}\boldsymbol{L}_-$ vs. $\boldsymbol{W}\boldsymbol{L}_+$. Right: $\boldsymbol{W}\boldsymbol{L}_+$ vs. $\boldsymbol{W}\boldsymbol{e}$. Colors refer to the neural populations, see Figure 7.3A. For both panels, we show on the sides the histograms of current components. Results were obtained with $T = 10$, $\gamma = 0.995$, $s = 2$, $n = 1000$. Numerical findings confirm that $\boldsymbol{W}\boldsymbol{L}_+ = -\boldsymbol{W}\boldsymbol{L}_-$ and $\boldsymbol{W}\boldsymbol{e} = s\,\boldsymbol{W}\boldsymbol{L}_+$.

The above equality agrees with the linear relationship (7.23) provided we have

$$\boldsymbol{L} = \frac{1}{s\gamma}\,\boldsymbol{W}\boldsymbol{e}\ . \tag{7.26}$$

This identity is in excellent agreement with numerical findings, as shown in Figure 7.4B.

We now assume that the current linearly expresses the target integral $\bar{y}_t$ at time t, and look for sufficient conditions for relationship (7.23) to hold at time $t+1$ after the new input x_{t+1} is received by the network. The current at time $t+1$ reads

$$\begin{aligned}\boldsymbol{\nu}_{t+1} &= \boldsymbol{W}(\boldsymbol{h}_t + x_{t+1}\,\boldsymbol{e}) = \boldsymbol{W}(\lfloor\boldsymbol{\nu}_t\rfloor_+ + x_{t+1}\,\boldsymbol{e})\\ &= \boldsymbol{W}\,(\lfloor\bar{y}_t\,\boldsymbol{L}\rfloor_+ + x_{t+1}\,\boldsymbol{e}) = \boldsymbol{W}(\lfloor\bar{y}_t\rfloor_+\,\boldsymbol{L}_+ + \lfloor-\bar{y}_t\rfloor_+\,\boldsymbol{L}_-) + x_{t+1}\,\boldsymbol{W}\boldsymbol{e}\\ &= \lfloor\bar{y}_t\rfloor_+\,\boldsymbol{W}\boldsymbol{L}_+ + \lfloor-\bar{y}_t\rfloor_+\,\boldsymbol{W}\boldsymbol{L}_- + x_{t+1}\,\boldsymbol{W}\boldsymbol{e}\ ,\end{aligned} \tag{7.27}$$

and should match

$$\boldsymbol{\nu}_{t+1} = \frac{\bar{y}_{t+1}}{s\gamma}\,\boldsymbol{W}\boldsymbol{e} = (\frac{\bar{y}_t}{s} + x_{t+1})\,\boldsymbol{W}\boldsymbol{e} \tag{7.28}$$

according to (7.23) and (7.26). We deduce that $\boldsymbol{W}\boldsymbol{L}_+$ and $\boldsymbol{W}\boldsymbol{L}_-$ have to be aligned along $\boldsymbol{W}\boldsymbol{e}$, see (7.26). Furthermore, based on the identity $y = \lfloor y\rfloor_+ - \lfloor -y\rfloor_+$, we readily obtain that

$$\boldsymbol{W}\boldsymbol{L}_+ = -\boldsymbol{W}\boldsymbol{L}_- = s^{-1}\,\boldsymbol{W}\boldsymbol{e}\ . \tag{7.29}$$

These relations are in very good agreement with numerics, see Figure 7.5.

Proxy loss for integration by a network of ReLU units. Conditions (7.26) and (7.29), as well as the relations between $\boldsymbol{L}_+, \boldsymbol{L}_-, \boldsymbol{L}$ ensure perfect generalization. They can be summarized into the set of four equalities

$$\begin{cases}\boldsymbol{d}^\dagger\lfloor\pm\boldsymbol{W}\boldsymbol{e}\rfloor_+ &= \pm s\gamma\\ \boldsymbol{W}\lfloor\pm\boldsymbol{W}\boldsymbol{e}\rfloor_+ &= \pm\gamma\boldsymbol{W}\boldsymbol{e}\end{cases} \tag{7.30}$$

linking the matrix of connections, the encoder and decoder vectors, as well as the scale and decay parameters.

We now introduce a proxy loss for $\boldsymbol{W}$, whose global minimum is achieved when conditions (7.30) above are fulfilled,

$$\mathcal{L}^{proxy} = \sum_{z=\pm 1} (\boldsymbol{d}^\dagger \lfloor z\boldsymbol{W}\boldsymbol{e} \rfloor_+ - \, zs\gamma)^2 + \sum_{z=\pm 1} |\boldsymbol{W} \lfloor z\boldsymbol{W}\boldsymbol{e} \rfloor_+ - \, z\gamma \boldsymbol{W}\boldsymbol{e}|^2 \, . \tag{7.31}$$

Experimentally, training on this proxy loss is extremely effective and as expected leads to perfect integrators satisfying the relations between currents shown in Figure 7.5. Similarly to the linear case, if the encoder and decoder are fixed during training, the convergence time of GD is strongly dependent on s with a preferred scale around $|\boldsymbol{e}||\boldsymbol{d}|$, see study of dynamics of learning with $\mathcal{L}^{proxy}$ in Appendix A.9,

While the batch-SGD loss is by definition based on actual computation of the network output for sample input sequences, the proxy loss imposes strict conditions on the dynamical behavior of the network that, in turn, ensure that the batch-SGD loss will be zero. While there is no *a priori* reason to believe that all global minima of (7.5) are global minima of (7.31), we empirically observed that the solutions $\boldsymbol{W}$ found by minimizing the batch–SGD seemed to also be approximate minima of the proxy loss (see Figure 7.5 for the ReLU case).

Properties of the connection matrix. Training integrators with either batch–SGD or the proxy loss yields solutions with one dominant singular value, of the form

$$\boldsymbol{W} \simeq \sigma \, \boldsymbol{l} \, \boldsymbol{r}^\dagger.$$

We report some properties of these solutions in Appendix A.10. In particular, the singular value σ is, in the case of fixed encoder and decoder with unit norms, bounded from below by $2\max(1, s)$, where s is the scale. In practice, except for scales close to 1, this lower bound seems to be tight, *i.e.* $\sigma = 2\max(1, s)$, see Appendix A.10, Figure A.5. We interpret this saturation as a manifestation of the conjecture by (Arora et al., 2019) that gradient descent implicitly favors solutions with small matrix norm, as rank–1 matrices have a Frobenius norm equal to their singular value.

7.4.3 Case of generic non-linear activation.

We now turn to the generic case of non-linear activation function f, showing how the idea of proxy loss developed in the ReLU case can be naturally extended to any f.

Generic proxy loss. We start by writing, for an arbitrary activation function f, the dynamical equation for the current, rather than for the activity state,

$$\boldsymbol{\nu}_{t+1} = \boldsymbol{W}(f(\boldsymbol{\nu}_t) + x_{t+1}\boldsymbol{e}). \tag{7.32}$$

At the first time-step, since $\boldsymbol{h}_{-1} = \boldsymbol{0}$ the current will be equal to $\boldsymbol{\nu}_0 = x_0 \boldsymbol{W}\boldsymbol{e} = y_0/(s\gamma)\boldsymbol{W}\boldsymbol{e}$. The error will thus vanish if and only if, for all y in the range of values of the target integral,

$$\boldsymbol{d}^\dagger f\left(\frac{y}{s\gamma}\,\boldsymbol{W}\boldsymbol{e}\right) = y. \tag{7.33}$$

These relations generalize the first two conditions in (7.30) for ReLU activation. Furthermore, imposing that $\boldsymbol{W}\boldsymbol{e}$ is an 'eigenvector' of the non-linear operator $\boldsymbol{W}f(\cdot)$ with eigenvalue γ, *i.e.*

$$\boldsymbol{W} \cdot f\left(\frac{y}{s\gamma}\,\boldsymbol{W}\boldsymbol{e}\right) = \frac{y}{s}\,\boldsymbol{W}\boldsymbol{e}, \tag{7.34}$$

for any y, will force the current to remain at any time aligned along $\boldsymbol{W}\boldsymbol{e}$. A simple inductive proof similar to (7.27) shows that in these idealized conditions the coordinate along that line will evolve proportionally to the output, similarly to eqn. (7.23). Combined with the condition derived for the first input, this is enough to guarantee perfectly generalizing integration.

For arbitrary f, conditions (7.33) and (7.34) can generally not be exactly satisfied for y varying over a continuous domain, *i.e.* for an infinite number of values of y. However, these conditions can be fulfilled for a discrete and finite subset, which will provide sufficient accuracy for good integration in practice, and we observe that the error on the integral of a time series of T inputs to scale as $\epsilon \sim n^{-1/2}$, irrespectively of T (as long as the integral values remains below y_{max}).

Based on these considerations, we propose a proxy loss for integration of a single scalar signal using an RNN with arbitrary non-linearity:

$$\mathcal{L}^{proxy,f,D=1}(\boldsymbol{W}) = \int_{z\in Z} \left[\boldsymbol{d}^\dagger f(z\boldsymbol{W}\boldsymbol{e}) - s\,\gamma\,z\right]^2 + \left[\boldsymbol{W}\cdot f(z\,\boldsymbol{W}\boldsymbol{e}) - z\,\gamma\,\boldsymbol{W}\boldsymbol{e}\right]^2. \tag{7.35}$$

This integral can be estimated via Monte-Carlo, and the choice of integration domain $Z = [-z_{max}, z_{max}]$ will restrict the maximum value $y_{max} = s\,\gamma\,z_{max}$ of y that can be represented through our network. It is still possible to obtain generalization to infinite number of integration steps, but the choice of γ has to be tuned so that the integral never exceeds the range the network was trained for.

Application to sigmoidal activation. We tested this new loss with a sigmoidal activation function[5]

$$f : x \to \frac{1}{1+e^{-50(x-0.1)}}.$$

Trained with a decay $\gamma = 0.8$, scale $s = 1$, $Z = [-5, 5]$ [6], those networks converge to a solution with a single dominant singular value and manage to integrate signals of arbitrary length, despite their inability to generalize to larger values of the integral. We observe that some neurons in the network exhibit a saturated behavior when the integral is above (resp. below) a neuron-specific threshold θ_i, while other neurons never reach that saturation. This results in a behavior where, during the monotonous evolution of the integral starting from 0, an increasing number of neurons get activated to support the integral, see Figure 7.6. While these networks have a very different phenomenology from the ReLU ones in state space, the integration is still performed through linear currents. We also confirmed that sigmoidal networks could be trained on the batch-SGD loss, yielding integrators with a single dominant singular value; training with γ too close to 1 results in poor performance, suggesting that the issues of generalization to large values of y is not entirely due to the choice of proxy loss, but could hint at intrinsic limitations of the network, related to the activation function.

The proxy loss (7.35) will be extended below to the general case $D > 1$. It should be noted that all non-linear integrators need not be absolute minima of the proxy loss

[5] The choice of the slope and bias, here 50 and 0.1 respectively, is not critical to the results. We chose the slope so that the transition from 0 to 1 of the sigmoid happens on a scale of $1/50$, close to the expected magnitude of the currents $n^{-1/2} \simeq 1/30$ for $n = 1000$. The bias was then chosen so that $x = 0$ is not in the linear portion of the sigmoid, nor in a fully saturated portion to avoid the null weight-matrix $\boldsymbol{W}^{(0)} = \boldsymbol{0}$ to be a fixed point of the learning dynamics.

[6] For $\gamma = 0.8$, $s = 1$, and inputs of magnitude bounded by 1, the integral evolves in $[-4, 4]$ as y_{max} is solution of $y_{max} = \gamma(y_{max} + s)$, hence $z_{max} = y_{max}/(s\gamma) = 5$. In practice, to observe the regimes $|y| \simeq y_{max}$ more easily, we test the network using sequences alternating between bursts of ± 1 inputs and long periods with no external input, see Figure 7.9.

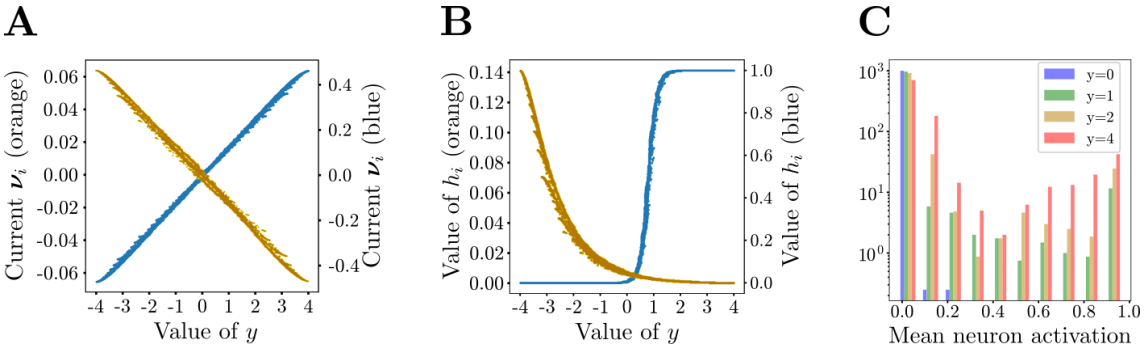


Figure 7.6: **A**: Value of the pre-activation current ν_i as a function of the integral for two representative neurons. **B**: Activity-integral characteristic curve for the neurons of panel **A**. One of them (blue, right scale) saturates for low enough values of y_t, while the other (orange, left scale) never saturates. **C**: Histogram of the mean activity of neurons for different values of the integrals, aggregated across 8 realizations of the training on the proxy loss (7.35). The range of integral values $[-4, 4]$ was divided in 100 bins to select the time-steps in the test sequences that corresponded to the values of y indicated in the legend. As the value of the integral increases, more neurons get strongly activated, and eventually saturate. The same evolution could be observed for integrals y_t decreasing below the zero value. Those networks were trained using the batch–SGD loss, $\gamma = 0.8$, $s = 1$, $n = 1000$, and the same results are found using the proxy loss.

and follow the linear current representation. We only show here that it is one possible representation scheme, which can be adapted to any non-linearity and could therefore help bridge the gap between idealized ReLU activation and more complex examples, e.g. inspired from real neurons.

7.5 Non-linear activation: case of multiple channels

We now consider the case of a multiplexed integrator with D input-output channels, performing D integrals in parallel. In practice, numerical experiments were carried out for $D = 2, 3, 4$.

Batch and proxy losses for multiple integrators. To train our RNN to carry out multiple integrations, we followed two different strategies. First, we used the batch loss defined in (7.5) from a set of input data, combined with a learning algorithm, *e.g.* SGD.

Second, drawing our inspiration from the detailed analysis of the single-channel case studied in the previous section, we introduced an extension of the proxy loss (7.35) to an arbitrary number $D > 1$ of input signals,

$$\mathcal{L}^{proxy,f,D}(\boldsymbol{W}) = \int_{z_1 \in Z_1} \cdots \int_{z_D \in Z_D} \Big\{ \sum_c \Big[\boldsymbol{d}_c^\dagger f\big(\sum_c z_c \boldsymbol{W} \boldsymbol{e}_c\big) - s_c\, \gamma_c\, z_c \Big]^2 + \Big[\boldsymbol{W} \cdot f\big(\sum_c z_c \boldsymbol{W} \boldsymbol{e}_c\big) - \sum_c \gamma_c\, z_c \boldsymbol{W} \boldsymbol{e}_c \Big]^2 \Big\} , \tag{7.36}$$

where the integral runs overs the D-dimensional range of values of the integrals, $Z_1 \times Z_2 \times \ldots \times Z_D$. As we shall see below, training with this loss allowed us to obtain networks with

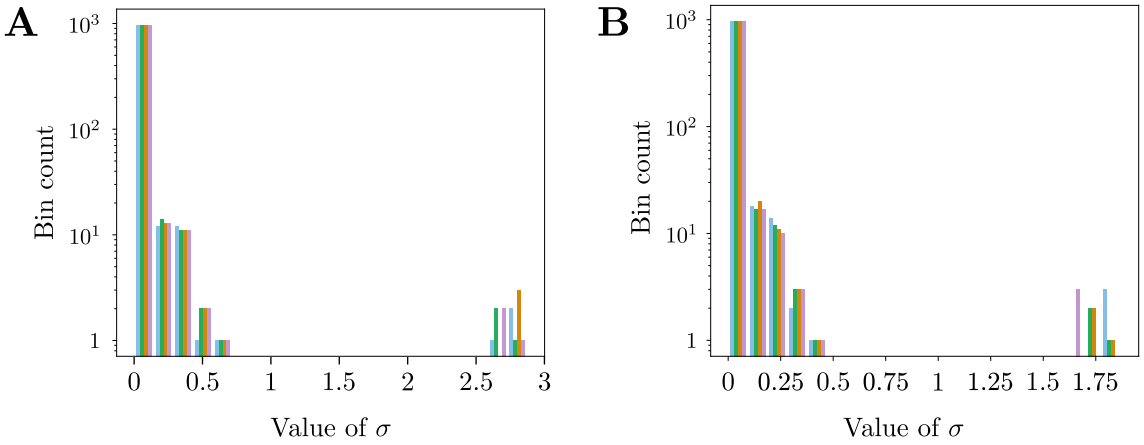


Figure 7.7: Histograms of the singular values of $\boldsymbol{W}$ in a ReLU (**A**) and sigmoidal (**B**) network across 4 realizations (one color each) of batch–SGD with $D = 3$, $T = 10$, $n = 1000$. The ReLU networks were trained with $\gamma_1 = \gamma_2 = .995$, $\gamma_3 = .992$, while the sigmoidal ones were trained with $\gamma_1 = \gamma_2 = .8$, $\gamma_3 = .75$. In both cases, a *bulk* of eigenvalues are found close to 0, while exactly 3 of them become substantially larger. A fair amount of variability can be observed in the exact value of those large eigenvalues, even using the same values of the decays.

arbitrary non-linearity that represent the integral values linearly in the space of currents, as we shall see below. Note that different activation functions, varying from neuron to neuron, could be also considered, *e.g.* through the introduction of a distribution of thresholds for the sigmoidal function.

Characterization of currents for ReLU networks. We start with the ReLU case. As in the linear case, training ReLU networks with Stochastic Gradient Descent of the batch loss yields networks that perform multiple integrations with excellent accuracy. Inspection of the connection matrices $\boldsymbol{W}$ reveals that they have D dominant singular values, as illustrated in Figure 7.7A for $D = 3$ channels. Such a spectral structure, consisting of a large number of "bulk" values and a few "outliers" that perform a computational task, is reminiscent of the setting investigated in (Schuessler et al., 2020a,b).

The D corresponding left eigenvectors $\boldsymbol{l}_c$ of the $\boldsymbol{W}$ matrix define a D–dimensional linear manifold for the current vector $\boldsymbol{\nu}_t$,

$$\boldsymbol{\nu}_t \simeq \sum_{c=1}^{D} \alpha_{c,t}\, \boldsymbol{l}_c, \;\; (\alpha_{1,t}, .., \alpha_{D,t}) \in \mathbb{R}^D \;\; , \tag{7.37}$$

while the activity state $\boldsymbol{h}_t$ of the network lives on a non-linear version of this manifold, shaped by the ReLU activation function:

$$\boldsymbol{h}_t = \lfloor \boldsymbol{\nu}_t \rfloor_+ \;\; . \tag{7.38}$$

Investigating the relation between the α coordinates in the current manifold and the values of the different integrals $\overline{\boldsymbol{y}}$, we empirically find that they are related by a linear mapping. More precisely, there exists a $D \times D$–matrix $\boldsymbol{R}$ such that the coordinates $\boldsymbol{\alpha}_t$ along the current-manifold can be written at all times as:

$$\alpha_{c,t} = \sum_{c'=1}^{D} R_{c,c'}\; \overline{y}_{c',t} \;\; . \tag{7.39}$$

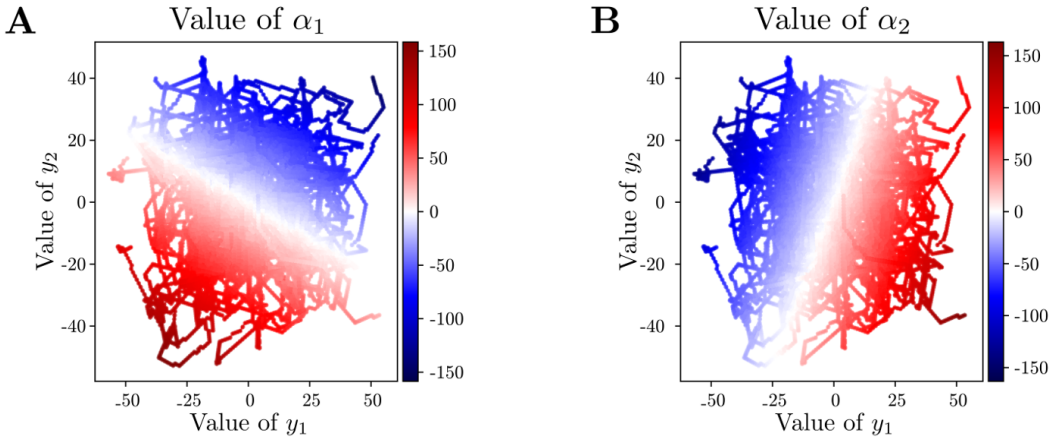


Figure 7.8: Value of the coordinates α_1 and α_2 in the current manifold as a function of the value of the target outputs $\bar{\boldsymbol{y}}$. Both coordinates depend linearly on the value of the two integrals (y_1, y_2), so that the position in the current manifolds is a linear representation of the integrals. The points were aggregated across 256 trajectories of length $T^{test} = 200$, for networks trained using batch–SGD on the mean square error (7.5) with training epochs duration $T^{train} = 10$, $\gamma_1 = 0.995$, $\gamma_2 = 0.992$.

In Figure 7.8, we illustrate this mapping in the $D = 2$ case. The methodology adopted is the following. While the network is performing integration, at each time-step t, we infer the $\alpha_{c,t}$ coordinates from the values of the currents through (7.37). The panels of Figure 7.8 show the coordinate α_c (left: $c = 1$; right: $c = 2$), see color code in the figure, as a function of the two integrals $\overline{y}_1, \overline{y}_2$. Aggregating those results across many long trajectories, we find that the value of the currents as a function of the targets is independent of the exact input sequence and linearly depends on the value of the integrals. Hence, the linear dependence of the current on the integrals, empirically found for $D = 1$ in (7.23), also holds in the multi-channel case.

We emphasize that the presence of a bulk of small, but not negligible, singular values of $\boldsymbol{W}$ (in addition to the D dominant ones) is not in contradiction with the fact that the current lives in a D–dimensional manifold. The corresponding singular vectors may be orthogonal to the encoders, and therefore never contribute to the internal state. To illustrate this point, we provide a quantitative evaluation of the distance between the currents $\boldsymbol{\nu}_t$ and the D–dimensional vector space $\mathcal{D}$ spanned by the D largest singular vectors $\boldsymbol{l}_c$ on the right-hand side of eqn. (7.37) as follows. After collecting the currents $\boldsymbol{\nu}$ at all time-steps during 128 trajectories of duration $T = 200$, we compute the projection $\boldsymbol{\nu}_t^{\parallel}$ of those currents on $\mathcal{D}$ using least-squares, and the orthogonal projection, $\boldsymbol{\nu}_t^{\perp}$. The ratio of their norms

$$r = \frac{\left\langle |\boldsymbol{\nu}_t^{\perp}| \right\rangle_t}{\left\langle |\boldsymbol{\nu}_t^{\parallel}| \right\rangle_t} \,, \tag{7.40}$$

where $\langle \cdot \rangle_t$ denotes the average over time, estimates how much of the current lies out of the D-dimensional manifold. Results for the ratios are reported in the first line of Table 7.1 for networks obtained from the batch and the proxy losses, and are very small, $r < 0.5$. These values are significantly smaller than what would be expected by chance in a null model in which all directions in the n-dimensional space of currents would be equally significant,

$$r_{null} = \sqrt{\frac{n}{D} - 1} \,, \tag{7.41}$$

	Batch		Proxy			
$\times 10^{-2}$	$D=1$	$D=2$	$D=1$	$D=2$	$D=3$	$D=5$
ReLU	1.3 ± 0.2	1.3 ± 0.1	1.63 ± 0.1	4.9 ± 0.5	3.3 ± 0.5	2.5 ± 0.1
Sigmoid	5.6 ± 1.5	33.4 ± 1.9	3.22 ± 0.2	11.3 ± 1.4	10.0 ± 0.4	5.9 ± 0.3

Table 7.1: Average ratios r of the projections of the current outside and inside the best D–dimensional subspace, see eqn. (7.40), for different activation functions and values of D, and $n = 1000$. Errors were estimated from 8 realizations of the training in the same conditions, and all values reported in the table are $10^2 \times r$ for readability.

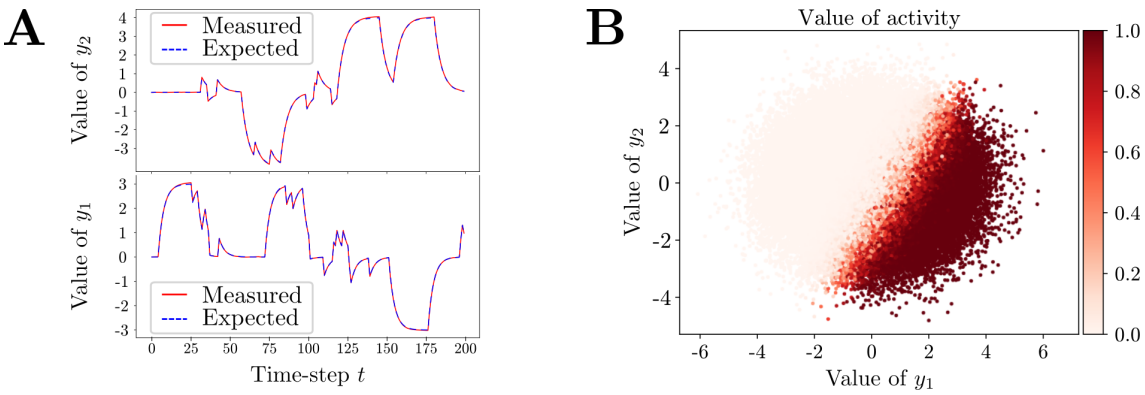


Figure 7.9: Learning of D-dimensional integrators with sigmoidal networks. **A**: Comparison between expected and measured output on structured test sequences, designed to alternate between bursts of ± 1 inputs and long periods with no external input to allow for visual discrimination of the origin of errors between scale and decay. **B**: Activity of a representative neuron in the (y_1, y_2) plane, measured on white-noise inputs. The decays are equal to 0.8 and 0.75, $n = 1000$, and the sigmoidal networks were trained using the proxy loss (7.36).

whose value is larger than 20 for $n = 1000$ and $D = 1, 2$.

Case of sigmoidal units. We have repeated the above analysis on networks with sigmoidal units, trained both from the batch and proxy losses. Results for a representative networks trained with the proxy loss to integrate $D = 2$ channels are shown in Figure 7.9A. We observe an excellent match between the output integrals and their target values. Similar results, albeit less accurate, are obtained with the batch loss.

As in the ReLU case, the connectivity matrix $\boldsymbol{W}$ is characterized by D large singular values, and a bulk of smaller ones. This bulk is influenced by several factors, including the initial condition over the matrix W and the choice of the learning algorithm. Despite the presence of these small singular values, the D-dimensional nature of the current can be assessed, see ratios r reported in Table 7.1. The values of r are much smaller than what would be expected from a null model, and confirm the low-dimensionality of the current manifold. Unsurprisingly, the values of the ratios for sigmoidal networks are 2 to 10 times larger than for their ReLU counterparts (for the same size n), as expected from the higher difficulty to solve conditions (7.33,7.34), see discussion in Section 7.4.3.

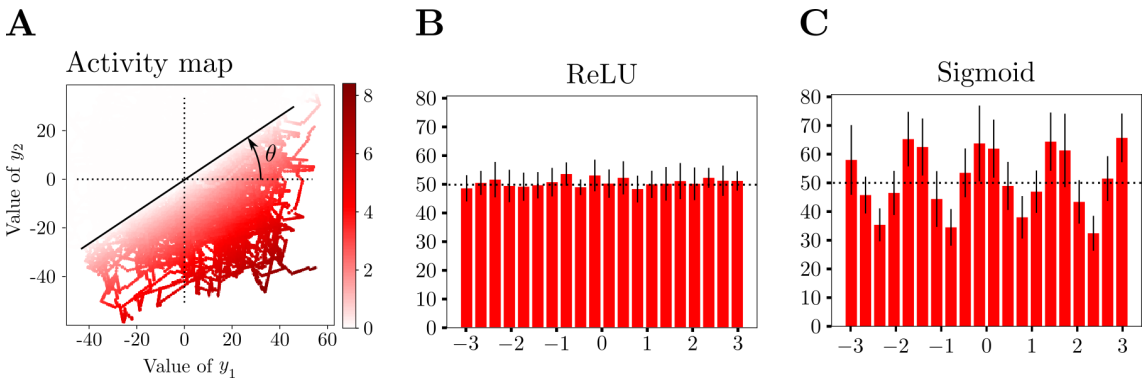


Figure 7.10: Mixed selectivity in bichannel integrators. **A**: Activity h_i of a representative neuron i in a ReLU network as a function of the two integrals, aggregated across 512 epochs of $T^{test} = 200$ time-steps. This activity is of the form $\max(\boldsymbol{s}_i^\dagger \boldsymbol{y}, 0)$, meaning that the neuron will only ever be active in half of the (y_1, y_2) plane. **B**: Distribution of the angle of the boundary plane between zero and non-zero activity across the $n = 1000$ neurons of a ReLU network. **C**: Distribution of the angle in the case of a Sigmoidal network. Horizontal dotted lines represent the uniform distribution. Same parameters as in Figure 7.9; histograms were aggregated across 16 repetitions of the training.

Nature of single neuron activity and mixed selectivity. The above findings allow us to determine how the state h_i of a neuron depends on the integrals $\bar{\boldsymbol{y}} = (\bar{y}_1, \bar{y}_2, ..., \bar{y}_D)$ in a ReLU network:

$$h_i = \lfloor \boldsymbol{s}_i^\dagger \bar{\boldsymbol{y}} \rfloor_+ \ , \ \text{with} \ \ s_{i,c} = \sum_{c'} R_{c',c} \boldsymbol{l}_{c',i} \ . \tag{7.42}$$

From a geometrical point of view, as illustrated in Figure 7.10A in the $D = 2$ case, each neuron activity h_i is the image through the ReLU non-linearity of the dot product between an associated direction $\boldsymbol{s}_i$ and the set of integrals $\bar{\boldsymbol{y}}$. The same feature is encountered for sigmoidal units, as shown in Figure 7.9B. We have then characterized the distribution of the angular direction of $\boldsymbol{s}_i$ across the n neurons, and find that it is equally distributed on $[0, 2\pi]$ when the network activation is ReLU, while it shows clear peaks for multiples of $\pi/2$ in the case of sigmoidal activation, see Figure 7.10B&C.

In the solutions empirically obtained through Gradient Descent, either on the batch loss or the proxy loss and for any number D of channels, we found that the network jointly encodes information about all integrals in the state of all neurons, a phenomenon similar to the one of "mixed selectivity" used to interpret cortical recordings in the field of computational neuroscience (Rigotti et al., 2013), and closely related to the issue of class selectivity in computer vision, see (Leavitt and Morcos, 2020a).

Mixed selectivity can be seen here as being deeply connected to the choice of the input and output layers of the network: in our experiments, all encoders and decoders have non-zero components on all neurons of the internal state. Therefore, during training, the connectivity matrix will be optimized in such a way that each of those neurons will extract and represent information about all integrals. If we instead constrain the encoder and decoder for each channel to have the same support, spanning only n/D neurons and non-overlapping with the support for any other channel, we find that the obtained solutions do not exhibit mixed selectivity anymore: the connection matrices $\boldsymbol{W}$ are block-diagonal, indicating that the network subdivided into D independent populations, each responsible for the coding of one integral. Relaxing the support constraint on either the encoders

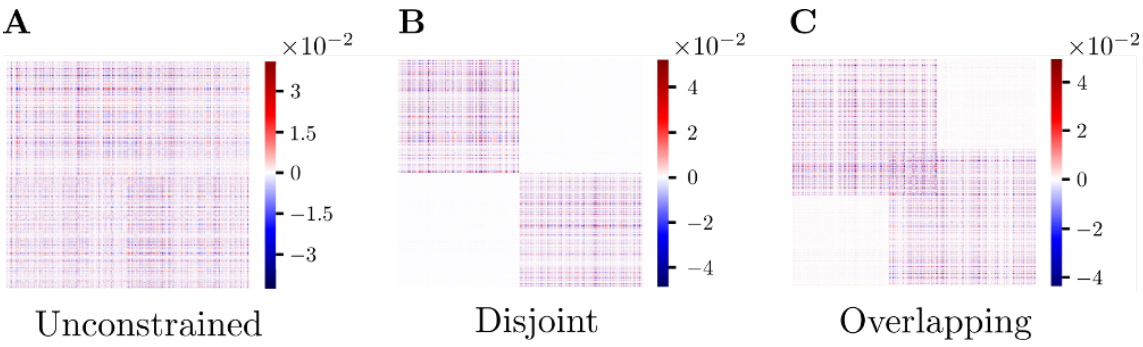


Figure 7.11: Visualization of the elements of the weight matrix $\boldsymbol{W}$ after training a ReLU network to integrate $D = 2$ signals through batch–SGD in three different cases of initialization: (left) the encoders and decoders are independent Gaussian vectors without any restriction; (middle) the population is divided in two: the first half of the neurons have non-zero encoder and decoder only on channel 1, and similarly the other half on channel 2; (right) starting from the non-overlapping case, we allow a small fraction of the neurons (middle portion) to have non-zero components on all $\boldsymbol{e}, \boldsymbol{d}$ vectors. We find that the use of disjoint supports produces block-diagonal solutions where one population is in charge of one integral and isolated from the others, thus exhibiting single selectivity.

or the decoders causes mixed specificity to reappear. Last of all, allowing the support of the channels to overlap causes the corresponding neurons to exhibit mixed selectivity, while the rest of the network remains simply selective. Those findings are illustrated in Figure 7.11.

We interpret this difference in behavior by the fact that the heavy constraints imposed between the encoders and decoders through their supports are enough to modify the energy landscape in such a way that the entropically favored connectivity matrices do not exhibit mixed selectivity anymore. None of these support constraints significantly impacts the final performance, nor the learning dynamics, and only the topology of the connectivity matrix is affected. Finally, we note that the choice of activation function also influences the distribution of selectivity angle, a fact that can not easily be understood from entropic considerations and could potentially be related to learning.

Learning with sign-constrained connections. So far, the only biological constraint we have considered regarded the states of neurons, which were forced to remain positive through the use of the ReLU activation function in order to represent firing rates. We now introduce a constraint on the weight matrix $\boldsymbol{W}$ itself, corresponding to the observed division between excitatory and inhibitory neurons known as Dale's Law (Dayan and Abbott, 2001): at initialization, we fix a certain fraction of the columns of $\boldsymbol{W}$, corresponding to the outgoing connections from a subpopulation of neurons, to have only negative entries, while the rest of the columns will have only positive entries. In order to maintain these constraints satisfied during training, after each step of optimization, we fix to 0 all the elements of $\boldsymbol{W}$ that changed sign.

At the end of the training the weight matrices exhibit one additional relevant singular value compared to their unconstrained counterparts:

$$\boldsymbol{W} \simeq \sigma_0 \boldsymbol{l}_0 \boldsymbol{r}_0^\dagger + \sum_{c=1}^{D} \sigma_c \boldsymbol{l}_c \boldsymbol{r}_c^\dagger \ .$$

The rank-1 contribution coming from this additional mode has the correct signs to

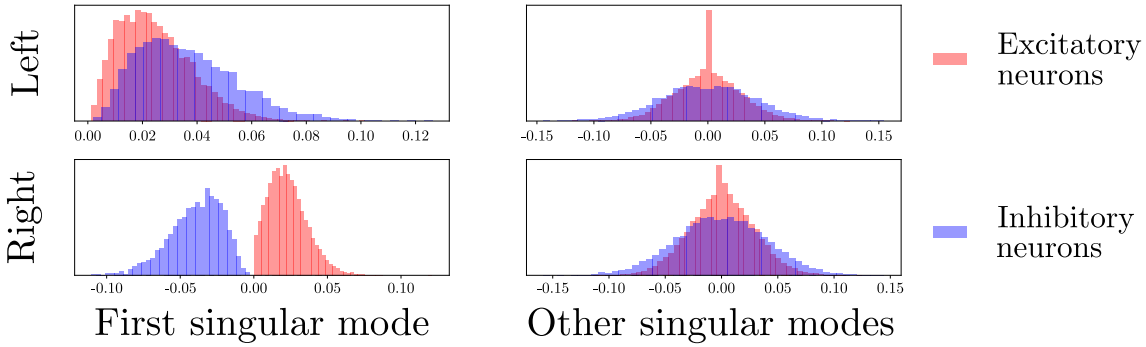


Figure 7.12: Distribution of the components of the left and right singular vectors for the largest singular value (left) and the following D ones (right). These histograms were obtained with 16 realizations of the batch–SGD training, using $n = 1000$, $D = 2$, and 25% of inhibitory neurons. While the signs of the components of the 2nd and 3rd singular vectors appear random, they have a particular structure in the first singular vector : the left singular vector is always positive, while the right is positive (resp. negative) if the neuron is in the excitatory (resp. inhibitory) population; the corresponding rank-1 matrix has columns of fixed signs corresponding to the ones of Dale's constraints.

satisfy Dale's constraint, as illustrated in Figure 7.12. Additionally, the left singular vector $\boldsymbol{l}_0$ is almost orthogonal to all decoding vectors $\boldsymbol{d}_c$, suggesting that this mode is not used for the computation of the integrals, but only as a way to satisfy the sign constraints over $\boldsymbol{W}$. It should be noted that our empirical result does not rule out the existence of networks of rank D performing D multiplexed integrals while satisfying Dale's Law. However, such solutions, if they exist, are not obtained through a simple Gradient Descent procedure from a zero or small $\boldsymbol{W}$.

7.6 Conclusion and perspectives

Summary of results and open questions. We have studied in this work how an RNN with n neurons learns to perform one or more integrations of temporal inputs; each integration was characterized by the target values of the scale factor s and of the decay coefficient γ (generally, slightly below 1).

In the case of an RNN with linear activation performing a single integral, we have precisely characterized the length of the temporal input necessary for perfect generalization (integration of any temporal signal), the optimal learning rate and the convergence time of the training procedure when the weight matrix is initially set to zero (or is small enough in norm). The coding of the integral is realized in a simple way: the activity vector of the entire neural population varies along a 1-dimensional direction in the n-dimensional space, with a proportionality factor equal to the integral.

In the case of ReLU activation, very accurate integration was obtained at the end of the training too. While a full mathematical analysis seemed much harder than for linear activation, we showed empirical evidence for the fact that the activity vector belongs to a piecewise 1-dimensional manifold. Coding of the positive and negative values of the integrals is done by two essentially non-overlapping populations of neurons, switching on and off when the integral value crosses zero. Remarkably, the pre-activation current of the ReLU units shows a simple behavior: it is proportional to the integral. We have derived sufficient conditions over the weight matrix for such a coding to take place, and

characterized the nature (directions of left and right eigenvectors, amplitude of singular value as a function of s, γ) of the corresponding rank-1 integrator.

In the case of a multiplexed network with D input/output channels, we have found that the weight matrix is of rank D; this statement is exact for linear activation and approximately true for ReLU activation RNNs, whose weight matrix has D large singular values compared to the $n - D$ remaining ones. Consequently, the network activity is restricted to a D–dimensional manifold in $\mathbb{R}^n$, whose geometry is imposed by the activation function of the neurons. For ReLU activation, as in the single-integral case, strong empirical evidence suggests that the pre-activation currents are linear combinations of the D integrals and span a D-dimensional linear subspace.

It is important to stress that the above results are not mere consequences of the threshold-linear nature of ReLU units. We have repeated our analysis with saturating units, obeying a sigmoidal activation function, with essentially the same results. Interestingly, some units never saturate for all possible values of the integral(s), other do, and all participate to produce the right outputs. To elucidate the reason for the D-dimensional nature of the coding of integrals by the currents, we have introduced a proxy loss reflecting sufficient conditions for such a coding. The networks trained from data (and the batch loss) behave similarly to the networks minimizing this proxy loss, both from the point of view of performance and representations.

From a purely machine-learning point of view, our work shows the versatility of RNNs to achieve simultaneously several computational tasks. The variety of representations supporting these computations could then be harnessed for transfer learning, see (Pan and Yang, 2010) for a review, by using our trained RNN as a (possibly fixed) feature extractor. One example of such a task is the one of context-dependent integration, studied in the prefrontal cortex of monkeys by (Mante et al., 2013), and which we adapt to our setup in Appendix A.11. The proxy loss we derived could also *a priori* be used as part of a full-task loss, following a similar reasoning to (Haviv et al., 2019), where one term in the loss is used to encourage internal dynamics that are known to be relevant for the task at hand and facilitate training.

Empirical analysis shows that very accurate multi-integrators with non-trivial activation functions can be obtained through Gradient Descent, and the representation scheme they adapt is linear in the space of currents. Three main limitations in this observation have to be noted. First, we do not show that this is the only representation scheme possible, and different solutions could possibly be found from pure mathematical reasoning. Second, rigorous analysis of the proxy loss remains necessary to understand in which conditions these representations are achievable, and to which accuracy. Finally, our study has focused on the case where the number of integrals D is small and the number of neurons n is large, and the question of how the optimal computational capacity (maximal sustainable value of D) precisely increases with n remains to be understood in the case of RNNs with non-linear activation.

Nature of representations and connection with computational neuroscience. While scalar integration using a single-layer recurrent network is far from state-of-the-art Machine Learning, the abundance of studies in the field of neuroscience (often motivated by the oculomotor system in fish) and the absence of a comprehensive theory of representation in such networks make it a worthwhile case study. Our theoretical analysis provides new evidence for the relevance of low-dimensional representations, and this result is robust to changes in the training method, the initial conditions of the weight matrix, as well as the choice of activation function. Our work therefore provides additional motivation for the

theoretical study of the properties of RNNs with low-rank coupling matrix initiated in the contexts of statistical physics (Mastrogiuseppe and Ostojic, 2018) and computational neuroscience, see (Barak, 2017) for a review.

As far as neuroscience is concerned, we believe that our result about the encoding of multiple integrals by each neuron, expressed by (7.42), is of particular interest. There is, indeed, a very striking analogy between our findings and the concept of mixed selectivity used to interpret cortical recordings in the field of computational neuroscience (Rigotti et al., 2013). For a long time, neuroscientists have focused on neurons whose activities depended on a single sensory relevant variable, such as the orientation of a bar in the visual cortex area V1 or the animal's head direction in the subiculum (in our case, the value of one particular integral y_c). Such neurons are, obviously, easier to identify from activity recordings. However, there is growing recognition that most cells display mixed sensitivity, that is, have activities varying non-linearly with several relevant variables, and that the relative degree of importance of each variable in determining the activity may considerably vary from neuron to neuron (as we find in Figure 7.10). Such mixed representations could be useful for decision-making based on multisensorial streams of information, a possibility sometimes put forward to explain their relevance in neuroscience. It is, from this point of view, remarkable that mixed representations spontaneously emerge in our study, where the RNN lacks any explicit incentive to exploit them, simply because they are much more likely than pure representations when the encoders and decoders have no intrinsic structure (Figure 7.10). The computational advantages of such mixed representations have been studied in (Leavitt and Morcos, 2020a,b), and suggest that they could improve both generalization and robustness of the performed computations. Other studies have focused on the emergence of disentangled representations, which have been shown to be relevant in both Natural Language Processing (Radford et al., 2017) and Computer Vision (Denton and Birodkar, 2017; Lee et al., 2018), suggesting that the optimal type of representation might depend on the specific task it supports. Studying the representations of computational tasks in artificial neural networks could therefore be a valuable tool to understand their biological counterparts, an approach already proposed in the domain of spatial navigation (Banino et al., 2018).

Chapter 8

Stable cognitive maps for Path Integration from bimodal inputs

Abstract

Spatial navigation in biological agents relies on the interplay between allothetic (visual, olfactory, auditory, ...) and idiothetic (proprioception, linear and angular velocity, ...) signals. How to combine and exploit these two streams of information, which vastly differ in terms of availability and reliability, is a crucial issue. In the context of a new two–dimensional continuous environment we developed, we propose a direct-inverse model of environment dynamics to fuse image and action related signals, allowing reconstruction of the action relating the two successive images, as well as prediction of the new image from its current value and the action. The definition of those models naturally leads to the proposal of a minimalistic recurrent architecture, called Resetting Path Integrator (RPI), that can easily and reliably be trained to keep track of its position relative to its starting point during a sequence of movements. RPI updates its internal state using the (possibly noisy) self-motion signal, and occasionally resets it when the image signal is present. Notably, the internal state of this minimal model exhibits strong correlation with position in the environment due to the direct-inverse models, is stable across long trajectories through resetting, and allows for disambiguation of visually confusing positions in the environment through integration of past movement, making it a prime candidate for a **cognitive map**. Our architecture is compared to off-the-shelf LSTM networks on identical tasks, and consistently shows better performance while also offering more interpretable internal dynamics and higher-quality representations.

8.1 Context

The Path Integration task. Path Integration (PI), a task in which an agent has to integrate information about a sequence of movements to keep track of the distance between its current and initial positions, has been extensively studied both in rodents (Etienne and Jeffery, 2004; McNaughton et al., 2006), and in artificial systems (Arleo et al., 2000; Banino et al., 2018; Zhao et al., 2020), and is thought by many to be an essential ingredient in the elaboration of **cognitive maps** (Tolman, 1948; Redish and Touretzky, 1997), that is, internal representations of the spatial structure of an environment capable of supporting navigation tasks (Golledge, 2003). Path Integration is particularly relevant from the point of view of representation learning as it relies on the interplay between qualitatively different inputs, a subject known as multi-modal learning and recently reviewed by Summaira et al. (2021). Those inputs can be broadly categorized into two groups. On the one hand, **idiothetic** signals, such as velocity (Kropff et al., 2015), head direction (Taube et al., 1990), memory of past trajectories (Cooper et al., 2001) or reafferent copies of motor signals (Iacoboni et al., 2001), which are generated by the agent itself. On the other hand, **allothetic** cues, *e.g.* provided by vision (Etienne et al., 1996), olfaction or "whisking" in mice (Deschênes et al., 2012) are intrinsically related to the external environment.

The simplest solution to implement PI would be an integrator network that takes as inputs proprioception signals, or, equivalently, the agent's time-dependent velocity. While the theory of integrator networks and the representations that emerge have been well studied (Seung, 1996; Fanthomme and Monasson, 2021), such a solution suffers from two major limitations. First, the accumulation of errors across the trajectory, either coming from imperfect sensor information or from imperfect integration, would make it unsuitable to represent Path Integration on arbitrarily long trajectories. Second, even if integration could be done without any error, representations constructed from proprioceptive information only would depend on the sequence of relative movements, and would be inadequate to the establishment of allocentric cognitive maps.

It is therefore crucial to understand how allothetic cues can be fused with self-motion information to achieve accurate PI, and to provide appropriate support for representations informative about the absolute position of the animal in its environment. This question has already been addressed in several works. Uria et al. (2020) introduced multiple recurrent neural networks (RNN) to build sophisticated 3D cognitive maps, with a variety of neurons displaying sensory-correlate features analogous to the ones encountered in cortical and hippocampal cell populations. Bicanski and Burgess (2018) proposed a model for the interactions between multiple brain areas concurring to the production of high-level spatial representations. In the field of robotics, Simultaneous Localization And Mapping systems based on Deep Learning are an active topic of research and show promising performance in key benchmarks of 3D navigation (Gupta et al., 2019; Zhang et al., 2020; Chaplot et al., 2020).

The objective of the present work is to address the issue of PI in the simplest possible setup, from the environment and network points of view, allowing for both good performance and interpretability. While our goal is not to provide a state-of-the-art method, *e.g.* directly applicable to robotics, we believe that such conceptual and (over)simplified approaches are valuable to help answer open questions about PI, such as its supervised or unsupervised nature, and its relevance to RL. In addition, the discrepancies between the representations built by our network and its natural biological counterparts may shed light on the additional functional and structural constraints acting on the latter.

The environment, associated sensors, and PI loss. In order to study PI in a simple and controlled setting, we developed a continuous 2–dimensional environment, which follows the basis of the OpenAI Gym specification (Brockman et al., 2016) to allow other researchers to reuse it in their experiments. This environment, detailed in Appendix B.1 and Figure 8.1 includes a certain number of colored markers, which will act as landmarks to allow the agent to determine its absolute position. It also includes walls, which will impede some movements and restrict visibility. Movement and perception in this environment corresponds to a top-down perspective, similar to what could be found in a Pac-Man game, centered on the current position of the agent. This setup is limited compared to real three-dimensional environments, such as the ones based on Minecraft (Johnson et al., 2016) or Doom (Wydmuch et al., 2018). However, the resulting simulations are much faster and easier to run, and our framework is convenient for the study of sensor fusion. In addition, primates could be trained and monitored while performing a similar task, with eyes fixated on a screen displaying the environment and moving via a joystick; this would provide a direct comparison between artificial agents and biological ones and allow for a better understanding of both (Yang and Wang, 2020).

The two sensors that we want our agent to combine are: 1) a noisy copy of the action $\boldsymbol{a}^{re} \equiv \boldsymbol{a}^{tr} + \epsilon\, \mathbf{u}$, where $\mathbf{u}$ is a unit Gaussian vector, and ϵ the level of noise. This **reafferent action** represents the proprioceptive signal, in that it does not depend on the state of the environment; 2) a retinal signal $\boldsymbol{s}$, which represents the allocentric information, and depends on the position of the agent (Figure 1, Right).

This retinal state mimics, to some degree, the activity of a biological retina such as the one of our hypothetical monkey: an array[1] of Difference of Gaussians retinal cells is centered on the position of the agent; the activity of each cell is computed by summing over the currently visible landmarks the activity they each elicit, which depends on their distances to the center of the cell **receptive field**. More details on the retina, notably on the optimal linear decoding of position from the activity can be found in Appendix B.2. For this sensor, we consider two possible types of errors: (1) the retina receives no information, similar to what would happen if the screen flickers or the animal closes its eyes; (2) the retinal state is correct, but at some point along the visual processing pipeline the obtained representation is "shuffled" between neurons. This second type of errors is meant to represent a form of temporal multiplexing (Akam and Kullmann, 2014) in the corresponding population. Depending on time steps, cells participate in the visual processing pipeline, or in other cognitive tasks; in the latter case, we would expect the population activity to have similar distribution across neurons, but no correlation with the visual representation, which is here achieved through reshuffling.

Based on these sensory signals, the agent has to estimate the displacements $\boldsymbol{\Delta r}_t$ from its starting point at all times t, see output of the PI network in Figure 8.2. We quantify the PI error along a trajectory of T steps through the loss

$$\mathcal{L}^{PI} = \sum_{t=1}^{T} \left\langle \left(\boldsymbol{\Delta r}_t - \sum_{k=1}^{t} \boldsymbol{a}_k^{tr} \right)^2 \right\rangle, \tag{8.1}$$

where $\langle \cdot \rangle$ represents the average over trajectories.

Network architectures. Since our PI task requires propagation of information from one time step to the next, it is not suitable for Multi-Layer Perceptron types of networks, which hold no memory of the previously received inputs, but can be handled with a

[1] In practice, we use three superimposed arrays, one for each RGB color channel, see Appendix B.2.

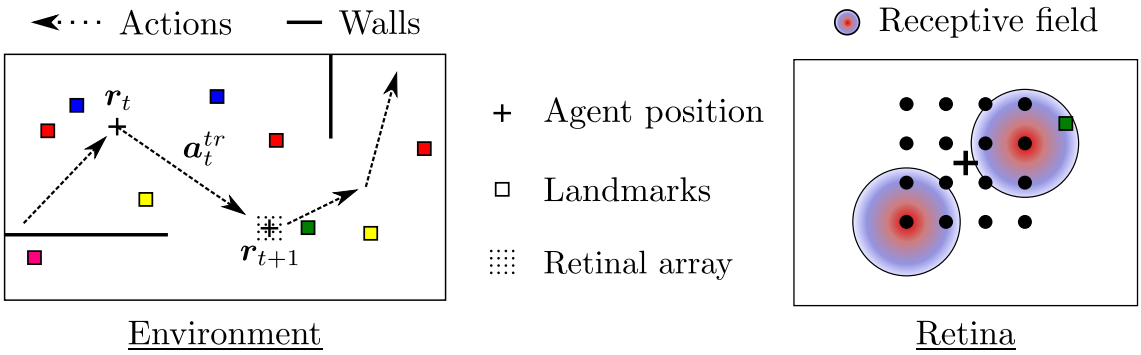


Figure 8.1: Presentation of our top-down perspective, two–dimensional continuous environment. Left: At each time step, the agent moves between positions $\boldsymbol{r}_t$ and $\boldsymbol{r}_{t+1}$ by performing an action $\boldsymbol{a}_t^{tr}$. The image it perceives through its retina, now centered on the new position $\boldsymbol{r}_{t+1}$, is modified accordingly, as the "landmarks" now occupy different positions with respect to the center of the retinal array. Right: Each neuron in the retinal array has an associated receptive field of the "Difference of Gaussians" type (for clarity, we represent only two); depending on the position of the landmark with respect to its receptive field, each neuron will be more or less activated, generating the "retinal state" that we will consider in the following as the "observation" received from the environment

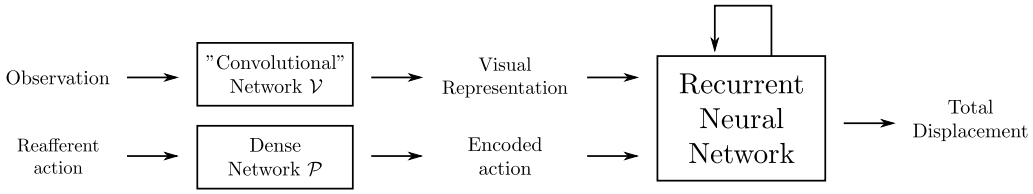


Figure 8.2: Shared structure of the models of Path Integration. The signals coming from the allocentric and proprioceptive sensors (respectively, the retinal activity and the reafferent action) are encoded through a first set of Neural Networks, before being used as inputs to a Recurrent Neural Network, whose output will be the predicted total displacement.

Recurrent Neural Network (RNN). In the following we will consider two broad categories of RNNs, with variable architectures and training procedures: (1) off-the-shelf Long Short-Term Memory modules (Hochreiter and Schmidhuber, 1997); (2) a minimal RNN based on the idea of direct-inverse environment models defined in Section 2, which we call the Resetting Path Integrator (RPI) and introduce in Section 8.3. Precise architectures are presented in Appendix B.3, and a sketch of the common structure shared by all our networks is presented in Figure 8.2.

A natural approach to multimodal PI consists in simply concatenating non-recurrent encodings of action and visual inputs, and feeding the resulting joint representation to a Recurrent Neural Network trained on the PI loss (1). However, as reported in the following, these initial attempts yielded unsatisfying solutions that 1) failed to perform "resetting" (see Section 3) when an image was available, and 2) had internal states in the RNN that were correlated only with displacement from the start of the trajectory, and not with the absolute position in the environment. In order to foster the emergence of allocentric representations of the environment, we now introduce the concept of direct–inverse models, and their associated losses. Direct–inverse models impose strong relationships between proprioceptive and visual signals, and as we shall see, lead to a natural approach to performing PI using those two qualitatively distinct streams of information.

8.2 Direct-inverse environment models

Evidence for **internal models** of environments has been found both in mammals (Ito, 2018) and in humans (Wolpert et al., 1998), notably within the Purkinje Cells of the Cerebellum, and have been hypothesized to be relevant for a wide range of motor (Wolpert and Miall, 1996; Kawato, 1999) and reasoning (Merfeld et al., 1999) behaviors. They have also been studied in the field of Reinforcement Learning, notably by Anderson et al. (2015), who showed that prediction of environment dynamics is an efficient pretraining step, by Pathak et al. (2017), who used the error in this model as a form of "curiosity" signal to encourage exploration, Corneil et al. (2018), who built a tabular model of an environment for use in an explicitly model-based planning algorithm, and Ha and Schmidhuber (2018), who used an internal model to allow the agent to learn from trajectories it "dreams" rather than from direct interaction with the environment, a formalism that could explain the observed coordinated replays of place and grid cells in rats (Ólafsdóttir et al., 2016).

These models can be formalized using the vocabulary of Partially Observable Markov Decision Processes (Sutton and Barto, 1998) used in Reinforcement Learning: the "hidden" state of the environment is the agent's absolute position; the observation is the retinal signal $\boldsymbol{s}$ (see Section 8.1 for details), the "action" $\boldsymbol{a}_t^{tr}$ at time t is the displacement in the environment from time t to $t+1$[2]. Models of the environment are defined on **transition tuples** $\boldsymbol{\tau} = (\boldsymbol{s}_t, \boldsymbol{a}_t^{tr}, \boldsymbol{s}_{t+1})$[3] and aim at predicting one of its components from the other two:

- the **direct** model $\mathcal{D}$ estimates the next state from the current one and the action:

$$\mathcal{D} : (\boldsymbol{s}_t, \boldsymbol{a}_t) \mapsto \overline{\boldsymbol{s}_{t+1}}. \tag{8.2}$$

- the **inverse** model $\mathcal{I}$ estimates the action that relates two states:

$$\mathcal{I} : (\boldsymbol{s}_t, \boldsymbol{s}_{t+1}) \mapsto \overline{\boldsymbol{a}_t}, \tag{8.3}$$

where $\langle\cdot\rangle$ represents the average over transition tuples. In practice, this approach would be highly inefficient and noise-sensitive in the case where the observed states are of high dimension but contain little relevant information (*e.g.* images). It is often preferable to construct these models on **representations**, obtained for example via a Convolutional Network $\mathcal{V}$; similarly, we introduce a Multi-Layer Perceptron (MLP) $\mathcal{P}$ that will map the two-dimensional action $\boldsymbol{a}_t$ to a vector of the same dimension as the representation $\mathcal{V}(\boldsymbol{s}_t)$; the resulting computation graph is presented in Figure 8.3, and the detailed architecture of the individual modules can be found in Appendix B.3.

To train these models we introduce two loss functions, computed from transition tuples:

$$\mathcal{L}^D(\mathcal{V}, \mathcal{P}, \mathcal{D}) = \left\langle \left[\mathcal{V}(\boldsymbol{s}_{t+1}) - \mathcal{D}(\mathcal{V}(\boldsymbol{s}_t), \mathcal{P}(\boldsymbol{a}_t))\right]^2 \right\rangle, \tag{8.4}$$

$$\mathcal{L}^I(\mathcal{V}, \mathcal{I}) = \left\langle \left[\boldsymbol{a}_t - \mathcal{I}(\mathcal{V}(\boldsymbol{s}_{t+1}), \mathcal{V}(\boldsymbol{s}_t))\right]^2 \right\rangle. \tag{8.5}$$

It should be noted that training the direct model $\mathcal{D}$ alone using the loss $\mathcal{L}^D$ of eqn (8.4) results in a trivial representation scheme in which all observations are mapped to the null

[2]The action could be considered a part of the observation, and the "partial observability" comes from the noise on these two as described in Section 8.1

[3]We do not include a reward signal as these experiments aim at mimicking "free foraging", in which the agent randomly explores an environment without explicit incentive to do so. The influence of a reward on representations will be the subject of a follow-up study.

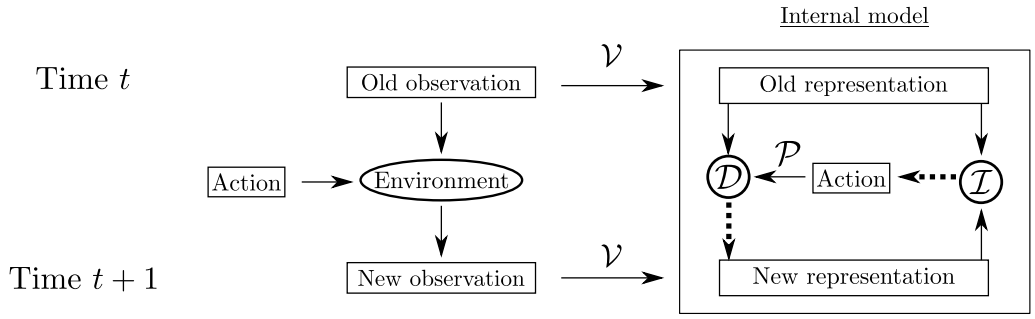


Figure 8.3: Overview of the direct-inverse model architecture, in which operators acting on internal representations aim at reproducing the dynamics of an environment. Dotted arrows indicate that the module they come from is trained to output the quantity they point towards (eqs. 8.4, 8.5).

vector $\mathbf{0}$. The inverse model $\mathcal{I}$ can be independently trained, but will generate irregular representations. When training $\mathcal{D}$ and $\mathcal{I}$ all together, the direct loss acts as a regularization, while the inverse loss breaks the symmetry required to converge to the trivial direct model. This yields a spatially structured representation of states, on which the direct operator acts non-trivially. More details on these representations can be found in Appendix B.4.

8.3 Resetting Path Integrator from direct-inverse models

The direct and inverse models can straightforwardly be combined to create a RNN capable of Path Integration, which we will call Resetting Path Integrator (RPI) in the following, and which is summarized in Figure 8.4. The internal state $\boldsymbol{H}$ is initialized with two concatenated copies of the initial observation [4]:

$$\boldsymbol{H}_0 = \Big(\mathcal{V}(\boldsymbol{s}_0)\,;\,\mathcal{V}(\boldsymbol{s}_0)\Big). \tag{8.6}$$

Then, at each time step, the internal state is updated using the direct model $\mathcal{D}$

$$\boldsymbol{H}_{t+1} \equiv \Big(\boldsymbol{h}_{t+1}\,;\,\mathcal{V}(\boldsymbol{s}_0)\Big) = \Big(\mathcal{D}(\boldsymbol{h}_t, \mathcal{P}(\boldsymbol{a}_t))\,;\,\mathcal{V}(\boldsymbol{s}_0)\Big), \tag{8.7}$$

and the displacement $\boldsymbol{\Delta r}_t$ is computed by applying the inverse model $\mathcal{I}$ between the updated and non-updated versions of the initial observation:

$$\boldsymbol{\Delta r}_t = \mathcal{I}(\boldsymbol{h}_t, \mathcal{V}(\boldsymbol{s}_0)). \tag{8.8}$$

While this approach suffers *a priori* from the same accumulation of errors as the direct movement integration, it also allows for an additional **resetting** mechanism, which was hypothesized by Prescott (1996) as a sufficient mechanism for spatial navigation: at any time-step, the agent could use the current visual observation to "correct" its internal state by disregarding the result of the direct model $\mathcal{D}$. To allow for this, as well as partial resetting[5], we introduce a **gating network** $\mathcal{G}$ that maps the current visual observation

[4] The second copy, which will not be modified by the network dynamics, is used as explicit "memory" of the starting point and is essential to observe resetting (see Section 8.4 and Appendix B.5 for more details).

[5] Another possible approach could have been to enforce total resetting by using $\mathcal{G}$ as a probability to choose the visual state, and to train this gate with a Reinforce-like algorithm. This setting seems less biologically relevant than ours and we did not investigate it further.

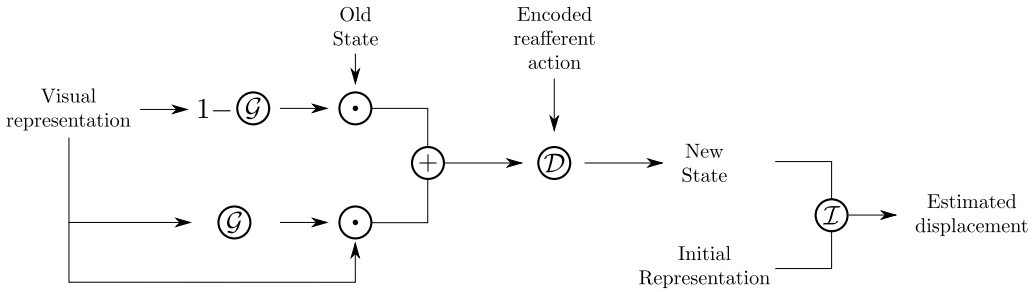


Figure 8.4: Minimal model for a Resetting Path Integrator, based on a Direct-Inverse model of environment dynamics. We assume that the agent is able to see correctly on the first step of the trajectory, and to keep a stable memory of this initial observation; this initial state is then updated by either using the direct model and the encoded reafferent action, or the new visual representation (resetting); the choice between those two behaviors is determined by the gating module $\mathcal{G}$.

$\boldsymbol{s}_t$[6] to a scalar between 0 (no resetting) and 1 (full resetting) that is used to interpolate between the proposed new state and the representation of the current observation, yielding the revised version of the update eqn (8.7):

$$\boldsymbol{H}_{t+1} = \Big(\mathcal{G} \circ \mathcal{V}(\boldsymbol{s}_t)\, \mathcal{D}(\boldsymbol{h}_t, \mathcal{P}(\boldsymbol{a}_t)) + [1 - \mathcal{G} \circ \mathcal{V}(\boldsymbol{s}_t)]\, \mathcal{V}(\boldsymbol{s}_{t+1})\,;\, \mathcal{V}(\boldsymbol{s}_0)\Big). \tag{8.9}$$

Of course, if reliable images were always available, the optimal solution would correspond to $\mathcal{G} = 0$ at all times, that is, to resetting at every time step and never using the direct model. In standard situations, where reliable visual information may be lacking, the recurrent nature of the network will allow for correct performance in-between resetting steps by keeping the internal state close to what the agent would observe from the environment. We therefore expect the internal state of the network to strongly depend on the current value of the position, but not on the trajectory used to get there[7], hence being a valid candidate for a cognitive map. More subtly, if the visual information received is ambiguous, *e.g.* the local set of landmarks seen by the retina is the same as in another part of the environment, see Section 8.4.2, we expect that the internal state should be able to lift the ambiguity in the observations through integration of previous motion, bridging the gap between regions of reliable visual information, a phenomenon which we actually observe in experiments.

To combine the direct-inverse and PI losses, training is done on their weighted sum

$$\mathcal{L}^{tot}(\mathcal{V}, \mathcal{P}, \mathcal{D}, \mathcal{I}, \mathcal{G}) = \alpha_{PI}\mathcal{L}^{PI}(\mathcal{V}, \mathcal{P}, \mathcal{D}, \mathcal{I}, \mathcal{G}) + \alpha_D \mathcal{L}^D(\mathcal{V}, \mathcal{P}, \mathcal{D}) + \alpha_I \mathcal{L}^I(\mathcal{V}, \mathcal{I}) \tag{8.10}$$

computed on random trajectories (see Appendix B.3 for more details on the parameters).

8.4 Results

In this section, we will analyze both the performance and the representations that emerge in networks trained on Path Integration loss of eqn. (8.1), using the environment layout represented in Figure 8.5, by looking at five different metrics. First, the average path

[6] Formally, this network could also take the current state $\boldsymbol{H}_t$ as input, but in practice this made the training more unstable without any noticeable improvement in performance.

[7] In time steps where strong resetting occurs ($\mathcal{G} \sim 0$), this statement is true since the state is exactly the representation of the current observation.

integration error, computed (1) on short trajectories ($T = 5$), during which no image is presented to the network and (2) on long trajectories ($T = 100$) with images available every five steps. We expect short and long-term errors to be of the same order of magnitude in the case of a network that can perform resetting, while the latter will be much larger than the former if no resetting behavior has been learned. Then, the average (over neurons) of the *coefficient of determinations* R_i^2 for the linear regression of the absolute the activity of individual neurons i participating to (3) the visual representation or (4) the internal state from the absolute position, and (5) of the (internal-state) neuron i from the relative displacement within the trajectory. These individual R_i^2 scores are found to be close to 1, either for the absolute or the relative positions, indicating whether neuron i is carrying allocentric or egocentric representation. Notice that $R^2 \sim 0$ would not mean that the neuron state would not convey any positional information, but that the latter would not be accessible to a linear decoder.

We will compare these five metrics under different training conditions: a "vanilla" LSTM model and our RPI model, trained on the PI loss in eqn. (8.1) only; an "improved" LSTM model[8] and our RPI model, trained end-to-end with the direct/inverse losses[9].

8.4.1 Performance of path integrators and nature of representations

The results on the five metrics obtained in the snake-path environment of Figure 5 are reported in Table 8.1; a similar experiment carried out in a more complicated layout is presented in Appendix B.7. Four conclusions can be drawn from those results: (1) Both recurrent structures (LSTM and RPI) are able to learn resetting behaviors, yielding similar short and long-term errors, see Figures 8.5 and 8.6; (2) Training with Direct and Inverse losses (as regularization) is necessary for the emergence of resetting; (3) Our RPI model yields internal states with much higher positional tuning than LSTMs; (4) Representations depend on absolute position in networks that perform resetting, and on relative position along the trajectory in networks that do not (see Appendix B.8 for details).

Despite RPIs being less expressive than their LSTM counterparts, they do not seem to achieve significantly worse performance (although more hyperparameter optimization would be required to confirm this statement). In addition, RPIs converge to solutions that are more easily interpretable, both in terms of tuning to the absolute position (illustrated by the higher R^2 scores), and of gating dynamics, see Figure 8.5. In Appendix B.9, we show that the value of the gate in a trained RPI is directly related to the cognitive mechanism of resetting, with the strength of resetting increasing when the training conditions incorporate more noise in the proprioceptive signals, as well as when the visual representations are more and more perturbed. In LSTMs, however, the reset and input gates are only weakly correlated with resetting, and instead might contribute to the computation of the direct model, see Appendix B.10 for details.

8.4.2 Disambiguation of ambiguous environment by RPI representations

Next, we considered a highly ambiguous situation where two rooms, located at the opposite ends of the environment, are designed to provide strictly identical visual cues. In that case,

[8] Several variants of LSTM were considered with varying degrees of success, see Appendix B.5 for details.

[9] Using these losses only for pretraining is possible, but leads to catastrophic forgetting, see Appendix B.6.

	Resetting Path Integrator		Long Short Term Memory	
	All losses	No model losses	Vanilla	Improved
Error (short)	0.021 ± 0.017	0.033 ± 0.025	0.014 ± 0.011	0.027 ± 0.018
Error (long)	0.026 ± 0.022	0.43 ± 0.36	0.16 ± 0.15	0.056 ± 0.038
R^2 (visual)	0.99 ± 0.048	0.11 ± 0.091	0.33 ± 0.18	0.96 ± 0.085
R^2 (PI, absolute)	0.98 ± 0.053	0.33 ± 0.08	0.3 ± 0.097	0.57 ± 0.22
R^2 (PI, relative)	0.35 ± 0.077	0.82 ± 0.11	0.77 ± 0.21	0.34 ± 0.14

Table 8.1: Comparison between our Resetting Path Integrator model and standard LSTM in the SnakePath environment. When trained without the model losses, both architectures fail to establish a proper resetting strategy, leading to higher error rates when tested on long trajectories ($T = 100$) than on short ones ($T = 5$), and the internal state during PI is a linear function of displacement along the trajectory, rather than of absolute position in the environment as is the case when model losses are used. Errors bars were estimated from 20 realizations of the training which differ both by initialization of the network weights, and drawn training trajectories.

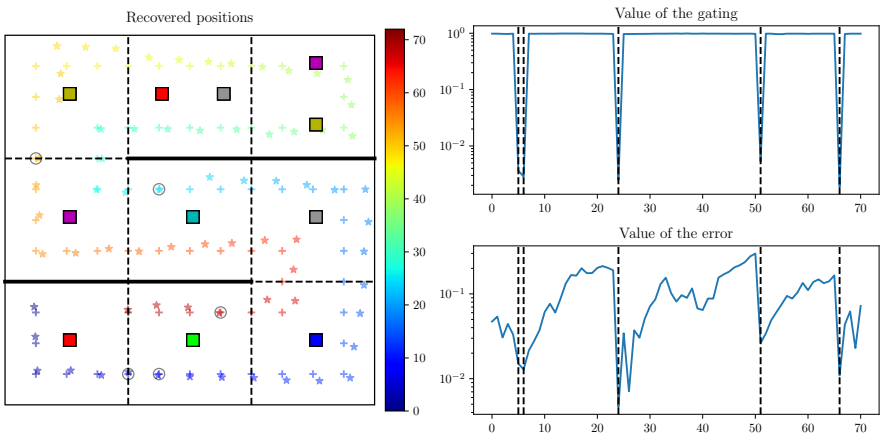


Figure 8.5: Example of Path Integration trajectory Left: Crosses represent the true position of the agent, while stars represent the one evaluated through Path Integration; black circles are placed around the positions at which an image was provided; time along the trajectory is represented by the color of the symbols. Top right: logarithm of the value of the resetting gate as a function of time along the trajectory. Bottom right: error between the true and reconstructed position. Vertical dashed lines indicate the time-steps at which the image was available to the network and not corrupted. In this example, actions are not drawn from the "free foraging" random policy but chosen to force exploration of the entire environment to better evaluate generalization at long distances, and reafferent actions are exact, so that errors are due only to the network itself.

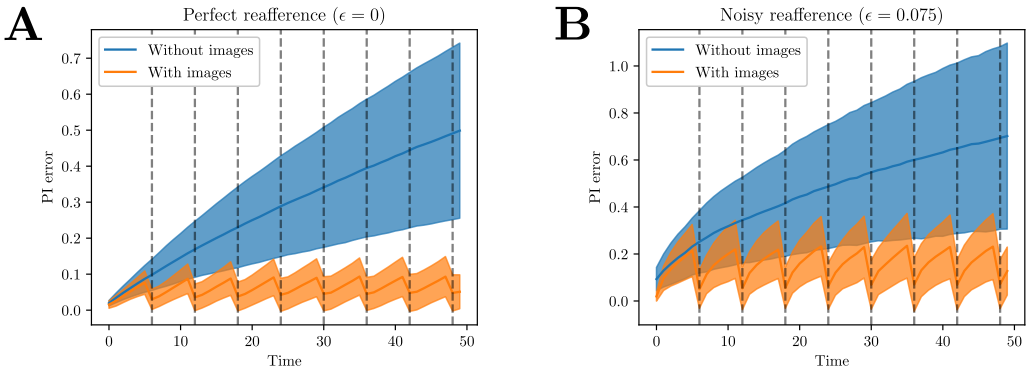


Figure 8.6: Path Integration errors achieved by our Resetting Path Integrator, with occasionally available retinal images (orange) and without images (blue). **A**: the reafferent action (proprioceptive signal) is exactly equal to the true one. **B**: a small amplitude Gaussian noise $\boldsymbol{\epsilon}$ is added to the reafferent action. Dashed vertical lines indicate steps at which images were presented, kept equal across 512 trajectories for each of the 8 networks used in the averaging. The qualitative agreement between those two plots, as well as results from Appendix B.9, suggest that our procedure is robust to small reafference errors, which have the same effect as direct model errors.

inverse models of the full environment give very large reconstruction errors when either of the images are located within one of these rooms.[10]

However, training a Resetting Path Integrator on this environment in an end-to-end fashion remains possible, as shown in Appendix B.11. The resulting networks exhibit three important properties: 1) the internal states observed during Path Integration are different in the two ambiguous rooms, as illustrated in Figure 8.7; 2) the PI error does not show any noticeable increase when the agent enters one of the ambiguous rooms, see Supplementary Figure B.11; 3) the resetting mechanism is not triggered for images coming from the ambiguous rooms, as illustrated in Supplementary Figure B.12.

The last observation was to be expected: as visual representations are identical between the two rooms, performing a resetting would, on average, result in a loss of spatial information with respect to keeping the state updated through the direct model. The first and second observations are non-trivial. To correctly perform Path Integration in the ambiguous rooms, our RPI network created new states, differing from those coming from the visual cues, that aim at bridging the gaps between "visually informative" regions. These networks have therefore managed to construct a representation of absolute position in the environment that does not rely only on local landmarks, but also draws from proprioceptive information and effectively fuses these sensors; intuitively, the Path Integrator is able to differentiate between two visually identical rooms by remembering how it got there, a highly desirable property for cognitive maps.

8.5 Conclusion

Results. In this study, we have demonstrated how a Recurrent Neural Network can be used to construct a cognitive map of a continuous spatial environment, by fusing unreli-

[10] If we train and test only on images coming from adjacent rooms, an inverse model can perform well as there is no couple of images that would correspond to two different reconstructed positions. The resulting direct–inverse models are however not suitable for Path Integration without retraining, as their long-range performance is severely limited by the ambiguous rooms.

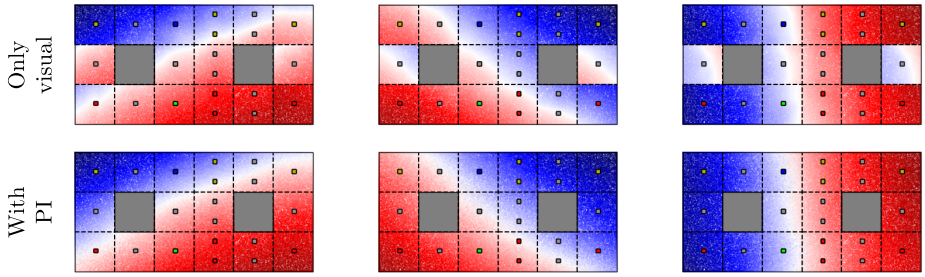


Figure 8.7: Comparison between representative neurons in the visual module $\mathcal{V}$ (top row) and the internal state $\boldsymbol{h}$ observed during Path Integration (bottom row) as a function of position within our ambiguous environment. The "dynamic" representation constructed during PI lifts the ambiguity between the two opposite rooms of the middle row, which contain the same landmark and are surrounded by identical rooms. Each column represents the normalized activation of a single neuron.

able proprioceptive and intermittently available external inputs through the task of Path Integration. We examined several ways of performing this fusion, using either off-the-shelf LSTM networks, or our proposed Resetting Path Integrator model, based on direct-inverse models of the environment dynamics, and including a single, scalar gating mechanism allowing for resetting (clearing) the internal state of the network and replacing it with an external signal. While all studied architectures and training procedures manage to perform Path Integration on short trajectories, only those regularized through the addition of the direct-inverse losses learned to efficiently use resetting and achieved similar error levels on very long and on short trajectories, thus overcoming error accumulation. The idea of incorporating high-level knowledge about desirable aspects of the internal states dynamics through regularization is close to the one of Haviv et al. (2019). We observe that the internal neural states are qualitatively different between path-integrator networks that learn resetting and those that do not: while the former are very close to linear functions of the absolute position in the environment, the latter are closer to linear functions of the displacement along the trajectory, see Table 1. This subtle difference is crucial, and implies that only networks capable of resetting have learned a "cognitive map" (Tolman, 1948; Spiers and Barry, 2015), which could *a priori* be transferred towards other spatially structured tasks.

Future directions. Contrary to previous works on PI in artificial agents that used highly spatially-structured inputs, relying on hypothesis about the existence of either place cells (for example in Arleo et al. (2000); Banino et al. (2018)) or grid cells (Zhao et al., 2020), our approach does not make such assumptions, and is, to our knowledge, the first one to allow for study of the emergence of these representations during training. Our simplistic environment setup did not result in emergence of either of those types of cells, but instead on a "top-down" map, which accurately depicts the Euclidean (by opposition to topological) structure of the environment. In other words, positions that are close in the layout (viewed from above) are close also in the map, though they could be far from each other in terms of "minimum number of steps", *e.g.* when separated by a wall that would need to be walked around. It is then a logical next step to move towards more realistic environments, using real first-person view, and allowing for rotations and translations, *e.g.*

Malmo (Johnson et al., 2016) or VizDoom (Wydmuch et al., 2018), whose interplay might be responsible for the particular coding scheme of place and grid cells (Harsh et al., 2020; Benna and Fusi, 2020).

Another major direction of research concerns the role of Path Integration as an end-goal: while it is assumed that such a task can be used to generate high-quality cognitive maps (as confirmed by our study), there is no evidence that this task is ever performed "intentionally". Preliminary experiments show that the recurrent representations constructed by the networks can be used for Reinforcement Learning tasks, such as goal-oriented navigation (moving towards a specific position in the environment), much more efficiently than through direct training. This result was to be expected, since representation learning is known to be a limiting factor in RL (Anderson et al., 2015). In a follow-up study, we will focus on incorporating the Path Integration loss (as well as the direct and inverse losses) as regularization terms in the policy learning algorithm; this approach is similar to the Intrinsic Curiosity Module of Pathak et al. (2017), in which the model errors were used as an exploration incentive, as well as zero-shot learning through environment models (Ha and Schmidhuber, 2018). All those methods can *a priori* be used at the same time.

Combining those two directions of research, making both tasks and environments more realistic, in particular close to what can be studied in live animals, will be key to understanding the intricacies of cognitive maps and of their relation to behavior.

Chapter 9

General conclusion and perspectives

Cognitive manifolds As we mentioned in Chapter 3, one of the major questions in the fields of Deep Learning and Neurobiology is to understand how and why certain **representations** emerge in the information processing systems of agents trying to perform specific tasks. Such studies are motivated by the discovery within the brain of behaving animals of specialized neurons, whose activity faithfully encodes the value of certain sensory or cognitive variables (*e.g.* place cells (O'Keefe and Nadel, 1978), speed cells (Kropff et al., 2015)). More recently, the development of new imaging (Helmchen and Denk, 2005) and electrophysiology (Jun et al., 2017) techniques have allowed computational neuroscientists to look at population-level coding, in which neurons represent information collectively rather than individually (*e.g.* the head-direction population (Turner-Evans et al., 2017)).

Considering, for simplicity, that the activity of each of the n neurons in the population is an unconstrained scalar, the set of all possible population states is a priori $\mathbb{R}^n$. However, in practice, the connectivity between neurons is such that not all of these states are practically observed; it is often observed that the states live close to a low-dimensional manifold, of dimension $d \ll n$ (Mastrogiuseppe and Ostojic (2018); Schuessler et al. (2020a); Schuessler et al. (2020b); Feulner and Clopath (2021)). Such structures also emerge naturally from "black-box" training of Artificial Neural Networks (Sussillo and Barak (2012); Fanthomme and Monasson (2021)), prompting the question of the relation between biological and artificial models. One qualitative argument for low-dimensionality is that within a high-dimensional stimulus such as an image, only a few independent variables are actually relevant for the task that the agent is trying to perform; for example, in the case of Path Integration (see Figure 9.1), only the position within the planar map, a 2–dimensional quantity, matters and most of the visual information can be discarded without any consequence on the quality of the behavior

Studying representations in animals ensures that said representations are behaviorally relevant; however, the experimental setups required to do such analysis are both expensive and hard to operate, making direct study difficult. To avoid this pitfall, a widely adopted and successful approach that emerged in the last few years consists in training Artificial Neural Networks on biology-inspired tasks, leveraging the strength of black-box training via Gradient Descent, and to study representations in those trained networks (see Yang and Wang (2020) for a review). This approach allows infinite precision in the analysis of the representations, but relies on the ability to train the networks and makes the assumption that obtained representations are biologically relevant. Both of those hypotheses have to be challenged through extensive comparison with experiments, but can at least serve as a

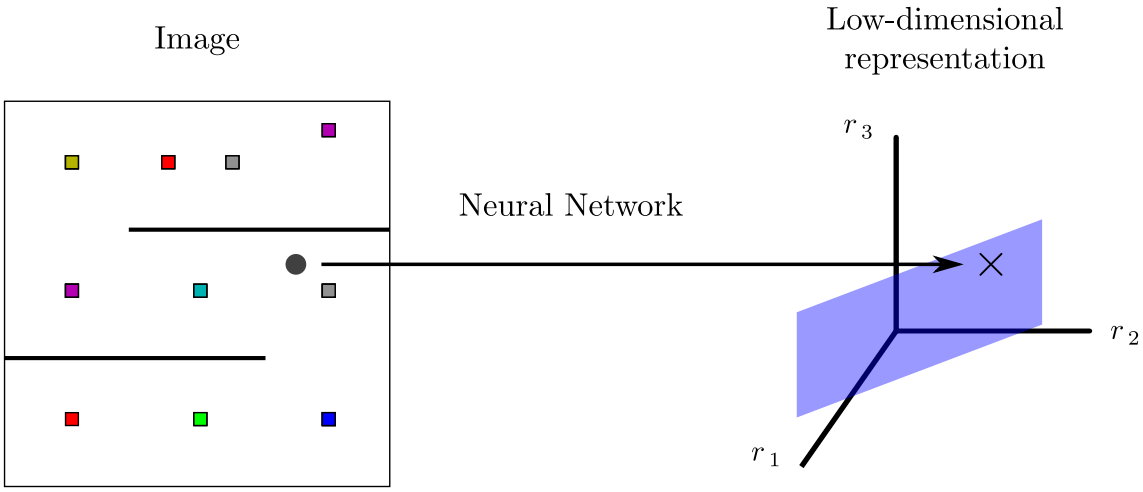


Figure 9.1: Example of a low-dimensional representation, in which a high-dimensional input (left, the image sensed when the agent is located at the position indicated by the gray circle) is **represented** as a point in a low-dimensional "cognitive" manifold of neuron firing-rates (right, the cross in the blue $2D$ plane). Most importantly, this mapping is done while preserving all information about the location of the agent within the environment, which is the only relevant information for a simple spatial navigation task.

guide to orient neuroscience research.

In our work on Neural Integration, the task we trained the networks on is very abstract, which allowed a deep theoretical understanding of the representations and the link between the internal dynamics of an RNN and the evolution operator of the external system it represents. Such analysis revealed the relevance of low-dimensional cognitive manifolds, and yielded insights on the link between the activation function of the network and the geometry of the cognitive manifolds. In our work on Path Integration, we studied how an RNN can be used to integrate both proprioceptive and visual signals into a single **cognitive map** of its environment. This approach to sensor fusion proved successful in generating meaningful representations of partially observable environments, which could be leveraged for more involved tasks such as goal-oriented navigation. However, nothing resembling grid cells (Moser et al., 2008) could be found in our trained networks; one possible explanation for this is that such representations could be relevant only within a particular motor control scheme: our agents were sensing their environment through a top-down perspective, and controlling their movements accordingly; this scheme is obviously very different from the one of an agent evolving in a real environment, where movement is divided into two degrees of freedom (rotation and forward velocity). While studying such control schemes would *a priori* be possible within our framework of direct-inverse modeling and integration, it would require a massive increase in computational resources necessary for simulation. Notably, such experiments would require a first-person rendering engine, and a physics engine allowing two-dimensional movement within a three-dimensional environment, such as the ones developed as part of the VizDoom (Wydmuch et al., 2018) or Malmö (Johnson et al., 2016) projects (implementing interfaces respectively to the Doom and Minecraft video games).

Representations of operators In the previous section, we considered representations mostly as static objects, obtained by mapping the state $\boldsymbol{s}_t \in \mathcal{S}$ (*e.g.* an image, a physical quantity, the current value of a computation) of an external system to an internal state

$\boldsymbol{h} = \mathcal{R}(\boldsymbol{s}) \in \mathcal{C} \subset \mathbb{R}^n$ in the cognitive manifold. We will now focus on adding structure to this cognitive manifold, by creating operators acting on it coherently with operators acting on the external system. This idea is very related to the one of "computation through latent dynamics" suggested by Sussillo (2014) in RNNs, although it can be adapted to non-recurrent networks and to situations in which the representation network and the evolution operators are implemented by different networks and possibly trained separately (as is the case in our approach to Path Integration).

We assume that a discrete evolution operator $\boldsymbol{\tau}_{\boldsymbol{x}}^{ext} : \mathcal{S} \to \mathcal{S}$ is defined that describes the evolution of the system from one step to the next[1]; this operator will, in all generality, be parametric, *i.e.* depend on external quantities, the inputs $\boldsymbol{x}_t$. In the case of integration, $\boldsymbol{\tau}^{ext}$ described the evolution of the vector of integrals after the arrival of the new inputs; in the case of the environment models, it described the evolution of the visual signal sensed by the agent after taking a new action; an extreme example could be the evolution of a qubit after applying a pulse of a given amplitude, duration and frequency. It is important to note that this operator does not need to have an exact, known mathematical expression; for all intents and purposes, it is enough to be able to **simulate** it, either numerically or even through real-world experimentation.

We are then interested in using an Artificial Neural Network to generate as faithful an approximation as possible of $\boldsymbol{\tau}_{\boldsymbol{x}}^{ext}$. More precisely, we define the architecture of this network so that it can accept both a state in the cognitive manifold $\mathcal{C}$ and an external input $\boldsymbol{x}$ (the simplest solution being concatenating the two), and return a vector of the same dimension as the state; we denote as $\boldsymbol{\tau}_{\boldsymbol{x}}^{ANN} : \mathcal{C} \to \mathbb{R}^n$ the corresponding parametric operator. The training of this model consists in tuning the parameters $\boldsymbol{\theta}$ of the ANN so that for all states and all external inputs, the effect of first mapping the external system to the cognitive manifold and then applying the internal evolution operator (*i.e.* doing $\boldsymbol{\tau}_{\boldsymbol{x}}^{ANN} \circ \mathcal{R}$) is the same as applying the external evolution operator and then performing the mapping to the internal state (*i.e.* doing $\mathcal{R} \circ \boldsymbol{\tau}_{\boldsymbol{x}}^{ext}$). This is represented in Figure 9.2.

If all operators acting on the external system are correctly approximated by their counterparts on the representations, then a complete internal model of the system has been generated by the networks. Such a model could then be used for abstract reasoning, planning or zero-shot learning, and intuitively corresponds to what we as humans consider as "understanding a phenomenon". The question of performing numerous tasks with a single neural network is being actively investigated, from Yang et al. (2019) who focuses biologically inspired tasks, to Team et al. (2021) who trained agents for multiple rule-based games in tridimensional environments, and the representations that emerge can be related to Universal Value Function Approximators (Schaul et al., 2015). These experiments are also deeply related to the problem of "learning-to-learn", in which knowledge of a previous problem can be used to improve learning speed on a new, related task, a form of transfer learning.

Imposing the covariance conditions Based on the ideas of the previous section, we introduced in Fanthomme and Monasson (2021) a new loss function for Recurrent Neural Networks that allowed *ex nihilo* training of perfect, generalizing integrators by forcing their internal dynamics (more precisely, their evolution operator) to correspond exactly to the one prescribed by computation of a decaying integral. To express the faithful operator representation conditions, one must be able to sample the states of the cognitive manifold $\mathcal{C}$; doing so requires the choice of an ansatz for the form of this manifold, and the one

[1] Continuous temporal evolution could be recovered, at least formally, by taking infinitesimal duration steps.

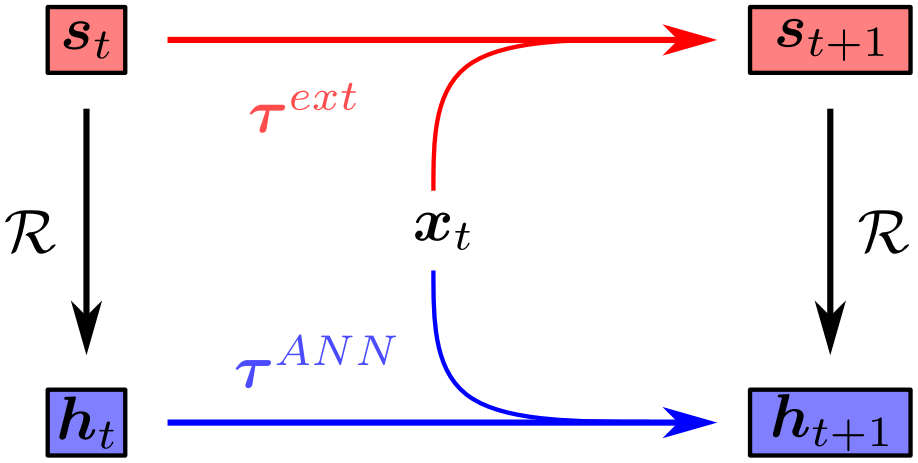


Figure 9.2: Illustration of the concept of representation of an operator. A parametric operator $\boldsymbol{\tau}^{ext}$, dependent on inputs $\boldsymbol{x}$, acts on the external system state $\boldsymbol{s}$ (top row, red); this external state can be mapped to an internal state $\boldsymbol{h}$ by a representation network $\mathcal{R}$ (see Figure 9.1), and another parametric operator $\boldsymbol{\tau}^{ANN}$ can be defined on those states. The internal operator is a representation of the external one if and only if the two paths joining the top-left and bottom-right corners of this diagram are identical (*i.e.* representing the state after applying the external operator is the same as first representing the state, and then applying the internal operator).

we chose is the "current-linear representation", which corresponds exactly to what we observed during our analysis of trained Neural Integrators. This ansatz assumes that the internal state of the network is given by:

$$\boldsymbol{h}_t = f(\boldsymbol{R}\boldsymbol{y}),$$

where $\boldsymbol{R}$ is a $n \times d$ matrix, $\boldsymbol{y}$ the d–dimensional vector of outputs, and f the non-linear activation function of the network. In practice, $\boldsymbol{R}$ can be expressed easily using the weight-matrix $\boldsymbol{W}$, allowing us to parametrize the cognitive manifold.

In addition to this loss enforcing correct internal dynamics to the internal representations, one often has to impose conditions on the output of a "decoding" network given the current value of the representation, which is particularly useful in avoiding that training generates trivial representations. For example, in the case of the forward environment models, a particularly simple solution consists in mapping all external images to the null vector, and have the forward operator do nothing. The introduction of the backward operator can be seen as a form of decoding of the representation, which makes this trivial solution invalid.

The full expression of the proxy loss, including both the internal evolution term and the decoding conditions, can be found in equation 7.36. Interestingly, this proxy loss could **a priori** easily be modified to accommodate other internal dynamics or decoding schemes, and even stochastic activations of the neurons, *e.g.* by grouping neurons into subpopulations and defining our proxy loss on the "averaged" neurons.

Bounding errors: resetting vs finite memory When considering deep or recurrent representations, the question of error accumulation (respectively, across layers or across time) becomes of paramount importance (Von Neumann, 2016; Mozeika et al., 2010). For instance, when performing integrals with decay constant γ, an input (or an error in the

integration) contributes to the value of the output for a time of order $-\ln(\gamma)^{-1}$, which diverges when γ goes to 1. Because of this, in practice training of Recurrent Neural Integrators with decays close to 1 requires additional precautions (avoiding too long input trajectories, using low learning-rates) and is often unstable. In some circumstances, such as evidence accumulation (Wong et al., 2007), this can be a desirable feature: arrival of a certain stimulus raises the confidence level of the agent in performing a given action; if the threshold is not met, and no additional "evidence" signal arrives for a certain time, it is important that the confidence decreases back to its original level to avoid "hair-trigger" reaction the next time evidence signal arrives. This would **a priori** be a major limitation when it comes to performing Path Integration, since in that case it is not acceptable to forget previous movements. The solution we proposed, and which naturally arises in state-of-the-art architecture, is to perform **resetting** operations whenever possible. In the case of Path Integration, resetting is performed using visual inputs, which are considered as ground-truth for the position of the agent, and this mechanism results in networks for which errors accumulate only on a finite duration (related to the frequency of availability of visual inputs), despite the integration taking place in the non-decaying regime $\gamma = 1$. Coming back to our example of decision-making, such resetting also could be relevant after the reaction threshold has been reached, *e.g.* to enforce a refractory period by resetting the confidence level lower than its initial value. Therefore, both error-bounding mechanisms that we studied can be relevant in a biological context, and the interplay between the two has to be taken into account.

Future directions Despite our best efforts, a lot of work remains to be done to better understand the physics of Recurrent Neural Networks, both artificial and biological. As a fundamental research direction, testing the limits of our "data-free" approach to training RNNs would be important, introducing more biological considerations such as stochasticity of firing and structural connectivity constraints. More empirically, introducing Reinforcement Learning procedures into our experiments on spatial navigation is both feasible and particularly interesting, as those results could shed light on whether Path Integration is a biologically relevant goal, or merely a by-product of agents learning other spatially structured behaviors such as foraging or predator escape. For example, one could perform experiments using real animals (either mice or primates) navigating our virtual environment, recording neurons during reward-free exploration and when training the animals on a particular task, using the results on artificial systems as a compass to help make sense of these extremely complicated datasets. Additionally, making our virtual environment closer to what biological agents might experiment (a 3–dimensional world, in which both rotations and translations can exist) could allow for different types of cognitive maps than the ones we observed here, and help understand the relevance of place and grid cells.

Bibliography

1. Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D. G., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., Wicke, M., Yu, Y., and Zheng, X. (2016). TensorFlow: A system for large-scale machine learning. *arXiv:1605.08695 [cs]*.

2. Abdel-Hamid, O., Mohamed, A., Jiang, H., Deng, L., Penn, G., and Yu, D. (2014). Convolutional Neural Networks for Speech Recognition. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 22(10):1533–1545.

3. Ackley, D. H., Hinton, G. E., and Sejnowski, T. J. (1985). A learning algorithm for boltzmann machines. *Cognitive Science*, 9(1):147–169.

4. Advani, M. S. and Saxe, A. M. (2017). High-dimensional dynamics of generalization error in neural networks. *arXiv:1710.03667 [physics, q-bio, stat]*.

5. Akam, T. and Kullmann, D. M. (2014). Oscillatory multiplexing of population codes for selective communication in the mammalian brain. *Nature Reviews Neuroscience*, 15(2):111–122.

6. Aksay, E., Olasagasti, I., Mensh, B. D., Baker, R., Goldman, M. S., and Tank, D. W. (2007). Functional dissection of circuitry in a neural integrator. *Nature Neuroscience*, 10(4):494–504.

7. Albus, J. S. (1971). A theory of cerebellar function. *Mathematical Biosciences*, 10(1):25–61.

8. Alemi, A., Baldassi, C., Brunel, N., and Zecchina, R. (2015). A Three-Threshold Learning Rule Approaches the Maximal Capacity of Recurrent Neural Networks. *PLOS Computational Biology*, 11(8):e1004439.

9. Almagro Armenteros, J. J., Sonderby, C. K., Sonderby, S. K., Nielsen, H., and Winther, O. (2017). DeepLoc: Prediction of protein subcellular localization using deep learning. *Bioinformatics*, 33(21):3387–3395.

10. Amit, D. J., Gutfreund, H., and Sompolinsky, H. (1985). Storing Infinite Numbers of Patterns in a Spin-Glass Model of Neural Networks. *Physical Review Letters*, 55(14):1530–1533.

11. Amodei, D., Ananthanarayanan, S., Anubhai, R., Bai, J., Battenberg, E., Case, C., Casper, J., Catanzaro, B., Cheng, Q., Chen, G., Chen, J., Chen, J., Chen, Z., Chrzanowski, M., Coates, A., Diamos, G., Ding, K., Du, N., Elsen, E., Engel, J., Fang, W., Fan, L., Fougner, C., Gao, L., Gong, C., Hannun, A., Han, T., Johannes,

L., Jiang, B., Ju, C., Jun, B., LeGresley, P., Lin, L., Liu, J., Liu, Y., Li, W., Li, X., Ma, D., Narang, S., Ng, A., Ozair, S., Peng, Y., Prenger, R., Qian, S., Quan, Z., Raiman, J., Rao, V., Satheesh, S., Seetapun, D., Sengupta, S., Srinet, K., Sriram, A., Tang, H., Tang, L., Wang, C., Wang, J., Wang, K., Wang, Y., Wang, Z., Wang, Z., Wu, S., Wei, L., Xiao, B., Xie, W., Xie, Y., Yogatama, D., Yuan, B., Zhan, J., and Zhu, Z. (2016). Deep Speech 2 : End-to-End Speech Recognition in English and Mandarin. In *International Conference on Machine Learning*, pages 173–182. PMLR.

12. Anderson, C. W., Lee, M., and Elliott, D. L. (2015). Faster reinforcement learning after pretraining deep networks to predict state dynamics. In *2015 International Joint Conference on Neural Networks (IJCNN)*, pages 1–7.

13. Anderson, P. W. (1958). Absence of Diffusion in Certain Random Lattices. *Physical Review*, 109(5):1492–1505.

14. Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Abbeel, P., and Zaremba, W. (2018). Hindsight Experience Replay. *arXiv:1707.01495 [cs]*.

15. Arleo, A., Smeraldi, F., Hug, S., and Gerstner, W. (2000). Place Cells and Spatial Navigation Based on 2D Visual Feature Extraction, Path Integration, and Reinforcement Learning. *Advances in Neural Information Processing Systems*, 13.

16. Arnold, D. B. and Robinson, D. A. (1991). A learning network model of the neural integrator of the oculomotor system. *Biological Cybernetics*, 64(6):447–454.

17. Arnold, D. B. and Robinson, D. A. (1997). The oculomotor integrator: Testing of a neural network model. *Experimental Brain Research*, 113(1):57–74.

18. Aronov, D., Nevers, R., and Tank, D. W. (2017). Mapping of a non-spatial dimension by the hippocampal–entorhinal circuit. *Nature*, 543(7647):719–722.

19. Arora, S., Cohen, N., Hu, W., and Luo, Y. (2019). Implicit Regularization in Deep Matrix Factorization. *arXiv:1905.13655 [cs, stat]*.

20. Auer, P., Cesa-Bianchi, N., and Fischer, P. (2002). Finite-time Analysis of the Multiarmed Bandit Problem. *Machine Learning*, 47(2):235–256.

21. Balaji, B., Mallya, S., Genc, S., Gupta, S., Dirac, L., Khare, V., Roy, G., Sun, T., Tao, Y., Townsend, B., Calleja, E., Muralidhara, S., and Karuppasamy, D. (2019). DeepRacer: Educational Autonomous Racing Platform for Experimentation with Sim2Real Reinforcement Learning. *arXiv:1911.01562 [cs]*.

22. Banino, A., Barry, C., Uria, B., Blundell, C., Lillicrap, T., Mirowski, P., Pritzel, A., Chadwick, M. J., Degris, T., Modayil, J., Wayne, G., Soyer, H., Viola, F., Zhang, B., Goroshin, R., Rabinowitz, N., Pascanu, R., Beattie, C., Petersen, S., Sadik, A., Gaffney, S., King, H., Kavukcuoglu, K., Hassabis, D., Hadsell, R., and Kumaran, D. (2018). Vector-based navigation using grid-like representations in artificial agents. *Nature*, 557(7705):429.

23. Barabási, A.-L., Albert, R., and Jeong, H. (1999). Mean-field theory for scale-free random networks. *Physica A: Statistical Mechanics and its Applications*, 272(1-2):173–187.

24. Barak, O. (2017). Recurrent neural networks as versatile tools of neuroscience research. *Current Opinion in Neurobiology*, 46:1–6.

25. Barrat-Charlaix, P., Muntoni, A., Shimagaki, K., Weigt, M., and Zamponi, F. (2021). Sparse generative modeling via parameter-reduction of boltzmann machines: application to protein-sequence families. *Physical Review E*, 104.

26. Bartlett, P. L., Montanari, A., and Rakhlin, A. (2021). Deep learning: A statistical viewpoint. *arXiv:2103.09177 [cs, math, stat]*.

27. Barton, J., Cocco, S., De Leonardis, E., and Monasson, R. (2014). Large pseudocounts and l2-norm penalties are necessary for the mean-field inference of ising and potts models. *Physical Review E*, 90.

28. Barton, J., De Leonardis, E., Coucke, A., and Cocco, S. (2016). Ace: adaptive cluster expansion for maximum entropy graphical model inference. *Bioinformatics*, 32.

29. Bassett, D. S. and Bullmore, E. (2006). Small-World Brain Networks. *The Neuroscientist*, 12(6):512–523.

30. Battista, A. and Monasson, R. (2020). Capacity-Resolution Trade-Off in the Optimal Learning of Multiple Low-Dimensional Manifolds by Attractor Neural Networks. *Physical Review Letters*, 124(4):048302.

31. Battiti, R. (1992). First- and Second-Order Methods for Learning: Between Steepest Descent and Newton's Method. *Neural Computation*, 4(2):141–166.

32. Baylor, D. A., Lamb, T. D., and Yau, K. W. (1979). Responses of retinal rods to single photons. *The Journal of Physiology*, 288:613–634.

33. Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. (2013). The Arcade Learning Environment: An Evaluation Platform for General Agents. *Journal of Artificial Intelligence Research*, 47:253–279.

34. Bellman, R. (1954). The theory of dynamic programming. *Bulletin of the American Mathematical Society*, 60(6):503–515.

35. Bengio, Y., Courville, A., and Vincent, P. (2013a). Representation Learning: A Review and New Perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828.

36. Bengio, Y., Mesnil, G., Dauphin, Y., and Rifai, S. (2013b). Better Mixing via Deep Representations. In *International Conference on Machine Learning*, pages 552–560. PMLR.

37. Benna, M. K. and Fusi, S. (2020). Are place cells just memory cells? memory compression leads to spatial tuning and history dependence.

38. Bicanski, A. and Burgess, N. (2018). A neural-level model of spatial memory and imagery. *eLife*, 7:e33752.

39. Bottou, L. (2010). Large-Scale Machine Learning with Stochastic Gradient Descent. In Lechevallier, Y. and Saporta, G., editors, *Proceedings of COMPSTAT'2010*, pages 177–186, Heidelberg. Physica-Verlag HD.

40. Bourgade, P. (2018). Random band matrices. *arXiv:1807.03031 [math-ph]*.

41. Brent, R. P. (2013). Algorithms for Minimization Without Derivatives. Courier Corporation.

42. Brette, R. and Gerstner, W. (2005). Adaptive Exponential Integrate-and-Fire Model as an Effective Description of Neuronal Activity. *Journal of Neurophysiology*, 94(5):3637–3642.

43. Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). OpenAI Gym. *arXiv:1606.01540 [cs]*.

44. Brown, P. K. and Wald, G. (1964). Visual Pigments in Single Rods and Cones of the Human Retina. *Science*, 144(3614):45–52.

45. Campbell, M. G., Ocko, S. A., Mallory, C. S., Low, I. I., Ganguli, S., and Giocomo, L. M. (2018). Principles governing the integration of landmark and self-motion cues in entorhinal cortical codes for navigation. *Nature neuroscience*, 21(8):1096–1106.

46. Cannon, S. C. and Robinson, D. A. (1987). Loss of the neural integrator of the oculomotor system from brain stem lesions in monkey. *Journal of Neurophysiology*, 57(5):1383–1409.

47. Cannon, S. C., Robinson, D. A., and Shamma, S. (1983). A proposed neural network for the integrator of the oculomotor system. *Biological Cybernetics*, 49(2):127–136.

48. Casati, G., Molinari, L., and Izrailev, F. (1990). Scaling properties of band random matrices. *Physical Review Letters*, 64(16):1851–1854.

49. Chaplot, D. S., Gandhi, D., Gupta, S., Gupta, A., and Salakhutdinov, R. (2020). Learning to Explore using Active Neural SLAM. *arXiv:2004.05155 [cs]*.

50. Chau Nguyen, H., Zecchina, R., and Berg, J. (2017). Inverse statistical problems: from the inverse ising problem to data science. *Advances in Physics*, 66.

51. Chen, G., Lu, Y., King, J. A., Cacucci, F., and Burgess, N. (2019). Differential influences of environment and self-motion on place and grid cell firing. *Nature Communications*, 10(1):630.

52. Chevalier-Boisvert, M., Willems, L., and Pal, S. (2018). Minimalistic Gridworld Environment for OpenAI Gym.

53. Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. *arXiv:1406.1078 [cs, stat]*.

54. Chomsky, N. (1959). On certain formal properties of grammars. *Information and Control*, 2(2):137–167.

55. Chung, J., Gulcehre, C., Cho, K., and Bengio, Y. (2014). Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. *arXiv:1412.3555 [cs]*.

56. Cleeremans, A., Servan-Schreiber, D., and McClelland, J. L. (1989). Finite State Automata and Simple Recurrent Networks. *Neural Computation*, 1(3):372–381.

57. Cocco, S., Feinauer, C., Figliuzzi, M., Monasson, R., and Weigt, M. (2018). Inverse statistical physics of protein sequences: A key issues review. *Reports on Progress in Physics*, 81(3):032601.

58. Cocco, S. and Monasson, R. (2011). Adaptive Cluster Expansion for Inferring Boltzmann Machines with Noisy Data. *Physical Review Letters*, 106(9):090601.

59. Cocco, S., Monasson, R., and Weigt, M. (2013). From Principal Component to Direct Coupling Analysis of Coevolution in Proteins: Low-Eigenvalue Modes are Needed for Structure Prediction. *PLOS Computational Biology*, 9(8):e1003176.

60. Collins, J., Sohl-Dickstein, J., and Sussillo, D. (2017). Capacity and Trainability in Recurrent Neural Networks. *arXiv:1611.09913 [cs, stat]*.

61. Connor, J., Martin, R., and Atlas, L. (1994). Recurrent neural networks and robust time series prediction. *IEEE Transactions on Neural Networks*, 5(2):240–254.

62. Cooper, B. G., Manka, T. F., and Mizumori, S. J. (2001). Finding your way in the dark: The retrosplenial cortex contributes to spatial memory and navigation without visual cues. *Behavioral neuroscience*, 115(5):1012.

63. Corneil, D., Gerstner, W., and Brea, J. (2018). Efficient Model-Based Deep Reinforcement Learning with Variational State Tabulation. *arXiv:1802.04325 [cs, stat]*.

64. Creswell, A., White, T., Dumoulin, V., Arulkumaran, K., Sengupta, B., and Bharath, A. A. (2018). Generative Adversarial Networks: An Overview. *IEEE Signal Processing Magazine*, 35(1):53–65.

65. Dauphin, Y., Pascanu, R., Gulcehre, C., Cho, K., Ganguli, S., and Bengio, Y. (2014). Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. *arXiv:1406.2572 [cs, math, stat]*.

66. Dayan, P. and Abbott, L. (2001). Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems, volume 15 of *Computational Neuroscience Series*. MIT press,.

67. Decelle, A. and Furtlehner, C. (2021). Restricted Boltzmann machine: Recent advances and mean-field theory. *Chinese Physics B*, 30(4):040202.

68. Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum Likelihood from Incomplete Data via the EM Algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38.

69. Denker, J., Schwartz, D. B., Wittner, B., Solla, S., Howard, R., Jackel, L., and Hopfield, J. (1987). Large Automatic Learning, Rule Extraction, and Generalization. *Complex Syst.*

70. Denton, E. L. and Birodkar, v. (2017). Unsupervised Learning of Disentangled Representations from Video. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems 30*, pages 4414–4423. Curran Associates, Inc.

71. Deschênes, M., Moore, J., and Kleinfeld, D. (2012). Sniffing and whisking in rodents. *Current Opinion in Neurobiology*, 22(2):243–250.

72. Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv:1810.04805 [cs]*.

73. Doeller, C. F., Barry, C., and Burgess, N. (2010). Evidence for grid cells in a human memory network. *Nature*, 463(7281):657–661.

74. Eggermont, J. J. (2001). Between sound and perception: Reviewing the search for a neural code. *Hearing Research*, 157(1):1–42.

75. Ekeberg, M., Hartonen, T., and Aurell, E. (2014). Fast pseudolikelihood maximization for direct-coupling analysis of protein structure from many homologous amino-acid sequences. *Journal of Computational Physics*, 276:341–356.

76. Ekeberg, M., Lövkvist, C., Lan, Y., Weigt, M., and Aurell, E. (2013). Improved contact prediction in proteins: Using pseudolikelihoods to infer Potts models. *Physical Review E*, 87(1):012707.

77. Elman, J. L. (1990). Finding Structure in Time. *Cognitive Science*, 14(2):179–211.

78. Elman, J. L. (1991). Distributed representations, simple recurrent networks, and grammatical structure. *Machine Learning*, 7(2):195–225.

79. Epstein, R. A., Patai, E. Z., Julian, J. B., and Spiers, H. J. (2017). The cognitive map in humans: Spatial navigation and beyond. *Nature Neuroscience*, 20(11):1504–1513.

80. Etienne, A. S. and Jeffery, K. J. (2004). Path integration in mammals. *Hippocampus*, 14(2):180–192.

81. Etienne, A. S., Maurer, R., and Séguinot, V. (1996). Path integration in mammals and its interaction with visual landmarks. *Journal of Experimental Biology*, 199(1):201–209.

82. Fanthomme, A. and Monasson, R. (2021). Low-Dimensional Manifolds Support Multiplexed Integrations in Recurrent Neural Networks. *Neural Computation*, pages 1–50.

83. Felsenstein, J. and Churchill, G. A. (1996). A Hidden Markov Model approach to variation among sites in rate of evolution. *Molecular Biology and Evolution*, 13(1):93–104.

84. Feulner, B. and Clopath, C. (2021). Neural manifold under plasticity in a goal driven learning behaviour. *PLOS Computational Biology*, 17(2):e1008621.

85. Freedman, D. J., Riesenhuber, M., Poggio, T., and Miller, E. K. (2003). A Comparison of Primate Prefrontal and Inferior Temporal Cortices during Visual Categorization. *Journal of Neuroscience*, 23(12):5235–5246.

86. Friedman, J., Hastie, T., and Tibshirani, R. (2008). Sparse inverse covariance estimation with the graphical lasso. *Biostatistics (Oxford, England)*, 9(3):432–441.

87. Fujimoto, S., van Hoof, H., and Meger, D. (2018). Addressing Function Approximation Error in Actor-Critic Methods. *arXiv:1802.09477 [cs, stat]*.

88. Funahashi, K.-i. and Nakamura, Y. (1993). Approximation of dynamical systems by continuous time recurrent neural networks. *Neural Networks*, 6(6):801–806.

89. Fyhn, M., Hafting, T., Treves, A., Moser, M.-B., and Moser, E. I. (2007). Hippocampal remapping and grid realignment in entorhinal cortex. *Nature*, 446(7132):190–194.

90. Gallego, J. A., Perich, M. G., Miller, L. E., and Solla, S. A. (2017). Neural Manifolds for the Control of Movement. *Neuron*, 94(5):978–984.

91. Ganguli, S., Bisley, J., Roitman, J., Shadlen, M., Goldberg, M. E., and Miller, K. D. (2008). One-Dimensional Dynamics of Attention and Decision Making in LIP. *Neuron*.

92. Ganguli, S. and Sompolinsky, H. (2012). Compressed Sensing, Sparsity, and Dimensionality in Neuronal Information Processing and Data Analysis. *Annual Review of Neuroscience*, 35(1):485–508.

93. Gardner, E. (1988). The space of interactions in neural network models. *Journal of Physics A: Mathematical and General*, 21(1):257–270.

94. Gardner, E. and Derrida, B. (1988). Optimal storage properties of neural network models. *Journal of Physics A: Mathematical and General*, 21(1):271–284.

95. Gershman, S. J. (2018). The Successor Representation: Its Computational Logic and Neural Substrates. *Journal of Neuroscience*, 38(33):7193–7200.

96. Gershman, S. J., Moore, C. D., Todd, M. T., Norman, K. A., and Sederberg, P. B. (2012). The Successor Representation and Temporal Context. *Neural Computation*, 24(6):1553–1568.

97. Goldt, S., Mézard, M., Krzakala, F., and Zdeborová, L. (2020). Modelling the influence of data structure on learning in neural networks: The hidden manifold model. *Physical Review X*, 10(4):041044.

98. Golledge, G., R. (2003). Human wayfinding and cognitive maps. In *The Colonization of Unfamiliar Landscapes*, pages 49–54. Routledge edition.

99. Goodfellow, I., Lee, H., Le, Q., Saxe, A., and Ng, A. (2009). Measuring Invariances in Deep Networks. In *Advances in Neural Information Processing Systems*, volume 22. Curran Associates, Inc.

100. Graves, A. and Schmidhuber, J. (2005). Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural Networks*, 18(5):602–610.

101. Graves, A., Wayne, G., and Danihelka, I. (2014). Neural Turing Machines. *arXiv:1410.5401 [cs]*.

102. Graves, A., Wayne, G., Reynolds, M., Harley, T., Danihelka, I., Grabska-Barwińska, A., Colmenarejo, S. G., Grefenstette, E., Ramalho, T., Agapiou, J., Badia, A. P., Hermann, K. M., Zwols, Y., Ostrovski, G., Cain, A., King, H., Summerfield, C., Blunsom, P., Kavukcuoglu, K., and Hassabis, D. (2016). Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626):471–476.

103. Gupta, S., Tolani, V., Davidson, J., Levine, S., Sukthankar, R., and Malik, J. (2019). Cognitive Mapping and Planning for Visual Navigation. *arXiv:1702.03920 [cs]*.

104. Ha, D. and Schmidhuber, J. (2018). World Models. *arXiv:1803.10122 [cs, stat]*.

105. Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. (2018). Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. In *International Conference on Machine Learning*, pages 1861–1870. PMLR.

106. Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P., and Levine, S. (2019). Soft Actor-Critic Algorithms and Applications. *arXiv:1812.05905 [cs, stat]*.

107. Hahnloser, R. H. R., Kozhevnikov, A. A., and Fee, M. S. (2002). An ultra-sparse code underliesthe generation of neural sequences in a songbird. *Nature*, 419(6902):65–70.

108. Haldane, A. and Levy, R. M. (2019). Influence of multiple-sequence-alignment depth on potts statistical models of protein covariation. *Phys. Rev. E*, 99:032405.

109. Hamaguchi, K. and Mooney, R. (2012). Recurrent Interactions between the Input and Output of a Songbird Cortico-Basal Ganglia Pathway Are Implicated in Vocal Sequence Variability. *Journal of Neuroscience*, 32(34):11671–11687.

110. Harsh, M., Tubiana, J., Cocco, S., and Monasson, R. (2020). 'Place-cell' emergence and learning of invariant data with restricted Boltzmann machines: Breaking and dynamical restoration of continuous symmetries in the weight space. *Journal of Physics A: Mathematical and Theoretical*, 53(17):174002.

111. Haviv, D., Rivkind, A., and Barak, O. (2019). Understanding and Controlling Memory in Recurrent Neural Networks. In *International Conference on Machine Learning*, pages 2663–2671. PMLR.

112. He, K., Zhang, X., Ren, S., and Sun, J. (2015). Deep Residual Learning for Image Recognition. *arXiv:1512.03385 [cs]*.

113. He, K., Zhang, X., Ren, S., and Sun, J. (2016). Identity Mappings in Deep Residual Networks. *arXiv:1603.05027 [cs]*.

114. Heess, N., TB, D., Sriram, S., Lemmon, J., Merel, J., Wayne, G., Tassa, Y., Erez, T., Wang, Z., Eslami, S. M. A., Riedmiller, M., and Silver, D. (2017). Emergence of Locomotion Behaviours in Rich Environments. *arXiv:1707.02286 [cs]*.

115. Helmchen, F. and Denk, W. (2005). Deep tissue two-photon microscopy. *Nature Methods*, 2(12):932–940.

116. Hinton, G. E. (2002). Training Products of Experts by Minimizing Contrastive Divergence. *Neural Computation*, 14(8):1771–1800.

117. Hinton, G. E. (2012). A Practical Guide to Training Restricted Boltzmann Machines. In Montavon, G., Orr, G. B., and Müller, K.-R., editors, *Neural Networks: Tricks of the Trade: Second Edition*, Lecture Notes in Computer Science, pages 599–619. Springer, Berlin, Heidelberg.

118. Hochreiter, S. and Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780.

119. Hodgkin, A. L. and Huxley, A. F. (1952). A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of Physiology*, 117(4):500–544.

120. Hopf, T. A., Ingraham, J. B., Poelwijk, F. J., Schärfe, C. P., Springer, M., Sander, C., and Marks, D. S. (2017). Mutation effects predicted from sequence co-variation. *Nature biotechnology*, 35(2):128–135.

121. Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences of the United States of America*, 79(8):2554–2558.

122. Hubel, D. H. and Wiesel, T. N. (1962). Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of Physiology*, 160(1):106–154.

123. Humphreys, G. W., Duncan, J., Treisman, A., O'Keefe, J., Burgess, N., Donnett, J. G., Jeffery, K. J., and Maguire, E. A. (1998). Place cells, navigational accuracy, and the human hippocampus. *Philosophical Transactions of the Royal Society of London. Series B: Biological Sciences*, 353(1373):1333–1340.

124. Hussein, A., Gaber, M. M., Elyan, E., and Jayne, C. (2017). Imitation learning: A survey of learning methods. *ACM Computing Surveys (CSUR)*, 50(2):1–35.

125. Iacoboni, M., Koski, L. M., Brass, M., Bekkering, H., Woods, R. P., Dubeau, M.-C., Mazziotta, J. C., and Rizzolatti, G. (2001). Reafferent copies of imitated actions in the right superior temporal cortex. *Proceedings of the National Academy of Sciences*, 98(24):13995–13999.

126. Ising, E. (1925). Beitrag zur Theorie des Ferromagnetismus. *Zeitschrift für Physik*, 31(1):253–258.

127. Ito, H. T. (2018). Prefrontal–hippocampal interactions for spatial navigation. *Neuroscience Research*, 129:2–7.

128. Jaeger, H. (2001). The" echo state" approach to analysing and training recurrent neural networks-with an erratum note'. *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report*, 148.

129. Jaeger, H. and Haas, H. (2004). Harnessing Nonlinearity: Predicting Chaotic Systems and Saving Energy in Wireless Communication. *Science*, 304(5667):78–80.

130. Johnson, M., Hofmann, K., Hutton, T., and Bignell, D. (2016). The Malmo platform for artificial intelligence experimentation. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, IJCAI'16, pages 4246–4247, New York, New York, USA. AAAI Press.

131. Jordan, M. I. (1997). Chapter 25 - Serial Order: A Parallel Distributed Processing Approach. In Donahoe, J. W. and Packard Dorsel, V., editors, *Advances in Psychology*, volume 121 of *Neural-Network Models of Cognition*, pages 471–495. North-Holland.

132. Jun, J. J., Steinmetz, N. A., Siegle, J. H., Denman, D. J., Bauza, M., Barbarits, B., Lee, A. K., Anastassiou, C. A., Andrei, A., Aydın, Ç., Barbic, M., Blanche, T. J., Bonin, V., Couto, J., Dutta, B., Gratiy, S. L., Gutnisky, D. A., Häusser, M., Karsh, B., Ledochowitsch, P., Lopez, C. M., Mitelut, C., Musa, S., Okun, M., Pachitariu, M., Putzeys, J., Rich, P. D., Rossant, C., Sun, W.-l., Svoboda, K., Carandini, M., Harris, K. D., Koch, C., O'Keefe, J., and Harris, T. D. (2017). Fully integrated silicon probes for high-density recording of neural activity. *Nature*, 551(7679):232–236.

133. Kabashima, Y. and Saad, D. (1998). Belief propagation vs. TAP for decoding corrupted messages. *Europhysics Letters (EPL)*, 44(5):668–674.

134. Kakade, S. and Langford, J. (2002). Approximately Optimal Approximate Reinforcement Learning. In *Proceedings of the Nineteenth International Conference on Machine Learning*, ICML '02, pages 267–274, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

135. Karpathy, A., Johnson, J., and Fei-Fei, L. (2015). Visualizing and Understanding Recurrent Networks. *arXiv:1506.02078 [cs]*.

136. Kawato, M. (1999). Internal models for motor control and trajectory planning. *Current Opinion in Neurobiology*, 9(6):718–727.

137. Kell, A. J. E., Yamins, D. L. K., Shook, E. N., Norman-Haignere, S. V., and McDermott, J. H. (2018). A Task-Optimized Neural Network Replicates Human Auditory Behavior, Predicts Brain Responses, and Reveals a Cortical Processing Hierarchy. *Neuron*, 98(3):630–644.e16.

138. Kim, S. S., Rouault, H., Druckmann, S., and Jayaraman, V. (2017). Ring attractor dynamics in the Drosophila central brain. *Science*, 356(6340):849–853.

139. Kingma, D. P. and Ba, J. (2017). Adam: A Method for Stochastic Optimization. *arXiv:1412.6980 [cs]*.

140. Kirk, E., D. (2004). Optimal Control Theory: An Introduction. Courier Corporation.

141. Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., Hassabis, D., Clopath, C., Kumaran, D., and Hadsell, R. (2017). Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526.

142. Konda, V. R. and Tsitsiklis, J. N. (2000). Actor-critic algorithms. In *Advances in Neural Information Processing Systems*, pages 1008–1014.

143. Kosmatopoulos, E., Polycarpou, M., Christodoulou, M., and Ioannou, P. (1995). High-order neural network structures for identification of dynamical systems. *IEEE Transactions on Neural Networks*, 6(2):422–431.

144. Kosterlitz, J. M., Thouless, D. J., and Jones, R. C. (1976). Spherical Model of a Spin-Glass. *Physical Review Letters*, 36(20):1217–1220.

145. Kropff, E., Carmichael, J. E., Moser, M.-B., and Moser, E. I. (2015). Speed cells in the medial entorhinal cortex. *Nature*, 523(7561):419–424.

146. Kuleshov, V. and Precup, D. (2014). Algorithms for multi-armed bandit problems. *arXiv:1402.6028 [cs]*.

147. Lai, T. and Robbins, H. (1985). Asymptotically efficient adaptive allocation rules. *Advances in Applied Mathematics*, 6(1):4–22.

148. Lapicque, L. (1907). L'excitation électrique des nerfs considérée comme une polarisation. *Journal de Physiologie et Pathologie Générale*, 9.

149. Larochelle, H. and Bengio, Y. (2008). Classification using discriminative restricted Boltzmann machines. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pages 536–543, New York, NY, USA. Association for Computing Machinery.

150. Leavitt, M. L. and Morcos, A. (2020a). Selectivity considered harmful: Evaluating the causal impact of class selectivity in DNNs. *arXiv:2003.01262 [cs, q-bio, stat]*.

151. Leavitt, M. L. and Morcos, A. S. (2020b). On the relationship between class selectivity, dimensionality, and robustness. *arXiv:2007.04440 [cs, stat]*.

152. LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1989). Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation*, 1(4):541–551.

153. Lecun, Y., Denker, J., and Solla, S. (1989). Optimal Brain Damage, volume 2.

154. LeCun, Y., Kanter, I., and Solla, S. (1991a). Second Order Properties of Error Surfaces: Learning Time and Generalization. In *Advances in Neural Information Processing Systems*, volume 3. Morgan-Kaufmann.

155. LeCun, Y., Kanter, I., and Solla, S. A. (1991b). Eigenvalues of covariance matrices: Application to neural-network learning. *Physical Review Letters*, 66(18):2396–2399.

156. Lee, D. D., Reis, B. Y., Seung, H. S., and Tank, D. W. (1997). Nonlinear Network Models of the Oculomotor Integrator. In Bower, J. M., editor, *Computational Neuroscience*, pages 371–377. Springer US, Boston, MA.

157. Lee, H., Grosse, R., Ranganath, R., and Ng, A. Y. (2009). Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 609–616, New York, NY, USA. Association for Computing Machinery.

158. Lee, H.-Y., Tseng, H.-Y., Huang, J.-B., Singh, M., and Yang, M.-H. (2018). Diverse Image-to-Image Translation via Disentangled Representations. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 35–51.

159. Levin, E., Tishby, N., and Solla, S. (1990). A statistical approach to learning and generalization in layered neural networks. *Proceedings of the IEEE*, 78(10):1568–1574.

160. Li, J., Monroe, W., and Jurafsky, D. (2017). Understanding Neural Networks through Representation Erasure. *arXiv:1612.08220 [cs]*.

161. Li, L., Chu, W., Langford, J., and Schapire, R. E. (2010). A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th International Conference on World Wide Web*, WWW '10, pages 661–670, New York, NY, USA. Association for Computing Machinery.

162. Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2019). Continuous control with deep reinforcement learning. *arXiv:1509.02971 [cs, stat]*.

163. Ling, Z.-H., Kang, S.-Y., Zen, H., Senior, A., Schuster, M., Qian, X.-J., Meng, H. M., and Deng, L. (2015). Deep Learning for Acoustic Modeling in Parametric Speech Generation: A systematic review of existing techniques and future trends. *IEEE Signal Processing Magazine*, 32(3):35–52.

164. Lipton, Z. C., Berkowitz, J., and Elkan, C. (2015). A Critical Review of Recurrent Neural Networks for Sequence Learning. *arXiv:1506.00019 [cs]*.

165. Little, W. A. (1974). The existence of persistent states in the brain. *Mathematical Biosciences*, 19(1):101–120.

166. Liu, X., Powell, D. K., Wang, H., Gold, B. T., Corbly, C. R., and Joseph, J. E. (2007). Functional Dissociation in Frontal and Striatal Areas for Processing of Positive and Negative Reward Information. *Journal of Neuroscience*, 27(17):4587–4597.

167. Louizos, C., Welling, M., and Kingma, D. P. (2018). Learning Sparse Neural Networks through L_0 Regularization. *arXiv:1712.01312 [cs, stat]*.

168. Luong, M.-T., Le, Q. V., Sutskever, I., Vinyals, O., and Kaiser, L. (2016). Multi-task Sequence to Sequence Learning. *arXiv:1511.06114 [cs, stat]*.

169. Maass, W., Joshi, P., and Sontag, E. D. (2007). Computational Aspects of Feedback in Neural Circuits. *PLOS Computational Biology*, 3(1):e165.

170. Maass, W., Natschläger, T., and Markram, H. (2002). Real-Time Computing Without Stable States: A New Framework for Neural Computation Based on Perturbations. *Neural Computation*, 14(11):2531–2560.

171. Maaswinkel, H. and Whishaw, I. Q. (1999). Homing with locale, taxon, and dead reckoning strategies by foraging rats: Sensory hierarchy in spatial navigation. *Behavioural Brain Research*, 99(2):143–152.

172. Maaten, L. V. D., Postma, E., and Herik, J. (2009). Dimensionality Reduction: A Comparative Review. *undefined*.

173. MacKay, D. J. (2003). Information theory, inference and learning algorithms. Cambridge university press.

174. Mallat, S. (1996). Wavelets for a vision. *Proceedings of the IEEE*, 84(4):604–614.

175. Mante, V., Sussillo, D., Shenoy, K. V., and Newsome, W. T. (2013). Context-dependent computation by recurrent dynamics in prefrontal cortex. *Nature*, 503(7474):78–84.

176. Markowitz, J. E., Iii, W. A. L., Guitchounts, G., Velho, T., Lois, C., and Gardner, T. J. (2015). Mesoscopic Patterns of Neural Activity Support Songbird Cortical Sequences. *PLOS Biology*, 13(6):e1002158.

177. Markram, H., Lübke, J., Frotscher, M., and Sakmann, B. (1997). Regulation of Synaptic Efficacy by Coincidence of Postsynaptic APs and EPSPs. *Science*, 275(5297):213–215.

178. Marr, D. (1969). A theory of cerebellar cortex. *The Journal of Physiology*, 202(2):437–470.

179. Mastrogiuseppe, F. and Ostojic, S. (2018). Linking connectivity, dynamics and computations in low-rank recurrent neural networks. *Neuron*, 99(3):609–623.e29.

180. Matsumoto, M. and Nishimura, T. (1998). Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation*, 8(1):3–30.

181. McDermott, J. H. and Simoncelli, E. P. (2011). Sound Texture Perception via Statistics of the Auditory Periphery: Evidence from Sound Synthesis. *Neuron*, 71(5):926–940.

182. McIntosh, L., Maheswaranathan, N., Nayebi, A., Ganguli, S., and Baccus, S. (2016). Deep Learning Models of the Retinal Response to Natural Scenes. In *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc.

183. McNaughton, B. L., Battaglia, F. P., Jensen, O., Moser, E. I., and Moser, M.-B. (2006). Path integration and the neural basis of the 'cognitive map'. *Nature Reviews Neuroscience*, 7(8):663–678.

184. Merfeld, D. M., Zupan, L., and Peterka, R. J. (1999). Humans use internal models to estimate gravity and linear acceleration. *Nature*, 398(6728):615–618.

185. Metropolis, N. and S. Ulam (1949). The monte carlo method. *Journal of the American Statistical Association*, 44(247):335–341.

186. Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous Methods for Deep Reinforcement Learning. *arXiv:1602.01783 [cs]*.

187. Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing Atari with Deep Reinforcement Learning. *arXiv:1312.5602 [cs]*.

188. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533.

189. Mohamed, A.-r., Dahl, G. E., and Hinton, G. (2012). Acoustic Modeling Using Deep Belief Networks. *IEEE Transactions on Audio, Speech, and Language Processing*, 20(1):14–22.

190. Momennejad, I., Russek, E. M., Cheong, J. H., Botvinick, M. M., Daw, N. D., and Gershman, S. J. (2017). The successor representation in human reinforcement learning. *Nature Human Behaviour*, 1(9):680–692.

191. Montavon, G., Samek, W., and Müller, K.-R. (2018). Methods for interpreting and understanding deep neural networks. *Digital Signal Processing*, 73:1–15.

192. Morcos, F., Pagnani, A., Lunt, B., Bertolino, A., Marks, D. S., Sander, C., Zecchina, R., Onuchic, J. N., Hwa, T., and Weigt, M. (2011). Direct-coupling analysis of residue coevolution captures native contacts across many protein families. *Proceedings of the National Academy of Sciences*, 108(49):E1293–E1301.

193. Moser, E. I., Kropff, E., and Moser, M.-B. (2008). Place Cells, Grid Cells, and the Brain's Spatial Representation System. *Annual Review of Neuroscience*, 31(1):69–89.

194. Mozeika, A., Saad, D., and Raymond, J. (2010). Noisy random Boolean formulae: A statistical physics perspective. *Physical Review E*, 82(4):041112.

195. Nair, V. and Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, pages 807–814, Madison, WI, USA. Omnipress.

196. Narendra, K. and Parthasarathy, K. (1990). Identification and control of dynamical systems using neural networks. *IEEE Transactions on Neural Networks*, 1(1):4–27.

197. Negele, J. W. (1982). The mean-field theory of nuclear structure and dynamics. *Reviews of Modern Physics*, 54(4):913–1015.

198. O'Keefe, J. and Nadel, L. (1978). The Hippocampus as a Cognitive Map. Oxford: Clarendon Press.

199. Okubo, T. S., Mackevicius, E. L., Payne, H. L., Lynch, G. F., and Fee, M. S. (2015). Growth and splitting of neural sequences in songbird vocal development. *Nature*, 528(7582):352–357.

200. Ólafsdóttir, H. F., Carpenter, F., and Barry, C. (2016). Coordinated grid and place cell replay during rest. *Nature Neuroscience*, 19(6):792–794.

201. Olah, C., Satyanarayan, A., Johnson, I., Carter, S., Schubert, L., Ye, K., and Mordvintsev, A. (2018). The Building Blocks of Interpretability. *Distill*, 3(3):e10.

202. Ono, S., Okanoya, K., and Seki, Y. (2016). Hierarchical emergence of sequence sensitivity in the songbird auditory forebrain. *Journal of Comparative Physiology A*, 202(3):163–183.

203. Onsager, L. (1944). Crystal Statistics. I. A Two-Dimensional Model with an Order-Disorder Transition. *Physical Review*, 65(3-4):117–149.

204. Opper, M. and Saad, D. (2001). Advanced mean field methods: Theory and practice. MIT press.

205. Pan, S. J. and Yang, Q. (2010). A Survey on Transfer Learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359.

206. Passingham, R. E. (1993). The frontal lobes and voluntary action. The Frontal Lobes and Voluntary Action. Oxford University Press, New York, NY, US.

207. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. *arXiv:1912.01703 [cs, stat]*.

208. Pathak, D., Agrawal, P., Efros, A. A., and Darrell, T. (2017). Curiosity-driven Exploration by Self-supervised Prediction. *arXiv:1705.05363 [cs, stat]*.

209. Pearl, J. (1982). Reverend Bayes on inference engines: A distributed hierarchical approach. Cognitive Systems Laboratory, School of Engineering and Applied Science ....

210. Pearlmutter, B. (1995). Gradient calculations for dynamic recurrent neural networks: A survey. *IEEE Transactions on Neural Networks*, 6(5):1212–1228.

211. Pearson, K. (1901). On lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572.

212. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, É. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12(85):2825–2830.

213. Posani, L., Cocco, S., Ježek, K., and Monasson, R. (2017). Functional connectivity models for decoding of spatial representations from hippocampal CA1 recordings. *Journal of Computational Neuroscience*, 43(1):17–33.

214. Potts, R. B. (1952). Some generalized order-disorder transformations. *Mathematical Proceedings of the Cambridge Philosophical Society*, 48(1):106–109.

215. Prescott, T. J. (1996). Spatial Representation for Navigation in Animats. *Adaptive Behavior*, 4(2):85–123.

216. Radford, A., Jozefowicz, R., and Sutskever, I. (2017). Learning to Generate Reviews and Discovering Sentiment. *arXiv:1704.01444 [cs]*.

217. Raffin, A., Hill, A., Ernestus, M., Gleave, A., Kanervisto, A., and Dormann, N. (2021). Stable-baselines3. DLR-RM.

218. Raghu, M., Poole, B., Kleinberg, J., Ganguli, S., and Sohl-Dickstein, J. (2017). On the Expressive Power of Deep Neural Networks. In *International Conference on Machine Learning*, pages 2847–2854. PMLR.

219. Ravikumar, P., Wainwright, M. J., and Lafferty, J. D. (2010). High-dimensional Ising model selection using l1-regularized logistic regression. *The Annals of Statistics*, 38(3):1287–1319.

220. Redish, A. D. and Touretzky, D. S. (1997). Cognitive maps beyond the hippocampus. *Hippocampus*, 7(1):15–35.

221. Richards, B. A., Lillicrap, T. P., Beaudoin, P., Bengio, Y., Bogacz, R., Christensen, A., Clopath, C., Costa, R. P., de Berker, A., Ganguli, S., Gillon, C. J., Hafner, D., Kepecs, A., Kriegeskorte, N., Latham, P., Lindsay, G. W., Miller, K. D., Naud, R., Pack, C. C., Poirazi, P., Roelfsema, P., Sacramento, J., Saxe, A., Scellier, B., Schapiro, A. C., Senn, W., Wayne, G., Yamins, D., Zenke, F., Zylberberg, J., Therien, D., and Kording, K. P. (2019). A deep learning framework for neuroscience. *Nature Neuroscience*, 22(11):1761–1770.

222. Rigotti, M., Barak, O., Warden, M. R., Wang, X.-J., Daw, N. D., Miller, E. K., and Fusi, S. (2013). The importance of mixed selectivity in complex cognitive tasks. *Nature*, 497(7451):585–590.

223. Rizzato, F., Coucke, A., de Leonardis, E., Barton, J. P., Tubiana, J., Monasson, R., and Cocco, S. (2020). Inference of compressed potts graphical models. *Phys. Rev. E*, 101:012309.

224. Robinson, D. (1968). The oculomotor control system: A review. *Proceedings of the IEEE*, 56(6):1032–1049.

225. Robinson, D. A. (1989). Integrating with Neurons. *Annual Review of Neuroscience*, 12(1):33–45.

226. Romo, R., Brody, C. D., Hernández, A., and Lemus, L. (1999). Neuronal correlates of parametric working memory in the prefrontal cortex. *Nature*, 399(6735):470–473.

227. Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408.

228. Rossier, J., Haeberli, C., and Schenk, F. (2000). Auditory cues support place navigation in rats when associated with a visual cue. *Behavioural Brain Research*, 117(1-2):209–214.

229. Roussel, C., Cocco, S., and Monasson, R. (2021). Barriers and Dynamical Paths in Alternating Gibbs Sampling of Restricted Boltzmann Machines. *arXiv:2107.06013 [cond-mat]*.

230. Rubinstein, R. Y. and Kroese, D. P. (2016). Simulation and the monte carlo method, volume 10. John Wiley & Sons.

231. Ruder, S. (2017). An overview of gradient descent optimization algorithms. *arXiv:1609.04747 [cs]*.

232. Rumelhart, D. E., McClelland, J. L., and Group, P. R. (1986). Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Foundations, volume 1. A Bradford Book, Cambridge, MA, USA.

233. Saad, D., editor (1999). On-Line Learning in Neural Networks. Publications of the Newton Institute. Cambridge University Press, Cambridge.

234. Saad, D. and Solla, S. A. (1995). Exact solution for on-line learning in multilayer neural networks. *Physical Review Letters*, 74(21):4337.

235. Salakhutdinov, R. and Hinton, G. (2009). Deep Boltzmann Machines. In *Artificial Intelligence and Statistics*, pages 448–455. PMLR.

236. Saxe, A. M., Koh, P. W., Chen, Z., Bhand, M., Suresh, B., and Ng, A. Y. (2011). On random weights and unsupervised feature learning. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ICML'11, pages 1089–1096, Madison, WI, USA. Omnipress.

237. Saxe, A. M., McClelland, J. L., and Ganguli, S. (2014). Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv:1312.6120 [cond-mat, q-bio, stat]*.

238. Schaul, T., Horgan, D., Gregor, K., and Silver, D. (2015). Universal Value Function Approximators. In *International Conference on Machine Learning*, pages 1312–1320. PMLR.

239. Schmidt, J., Marques, M. R. G., Botti, S., and Marques, M. A. L. (2019). Recent advances and applications of machine learning in solid-state materials science. *npj Computational Materials*, 5(1):1–36.

240. Schuessler, F., Dubreuil, A., Mastrogiuseppe, F., Ostojic, S., and Barak, O. (2020a). Dynamics of random recurrent networks with correlated low-rank structure. *Physical Review Research*, 2(1):013111.

241. Schuessler, F., Mastrogiuseppe, F., Dubreuil, A., Ostojic, S., and Barak, O. (2020b). The interplay between randomness and structure during learning in RNNs. *arXiv:2006.11036 [q-bio]*.

242. Schulman, J., Levine, S., Moritz, P., Jordan, M. I., and Abbeel, P. (2017a). Trust Region Policy Optimization. *arXiv:1502.05477 [cs]*.

243. Schulman, J., Moritz, P., Levine, S., Jordan, M., and Abbeel, P. (2018). High-Dimensional Continuous Control Using Generalized Advantage Estimation. *arXiv:1506.02438 [cs]*.

244. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017b). Proximal Policy Optimization Algorithms. *arXiv:1707.06347 [cs]*.

245. Seung, H. S. (1996). How the brain keeps the eyes still. *Proceedings of the National Academy of Sciences*, 93(23):13339–13344.

246. Song, H. F., Yang, G. R., and Wang, X.-J. (2016). Training Excitatory-Inhibitory Recurrent Neural Networks for Cognitive Tasks: A Simple and Flexible Framework. *PLOS Computational Biology*, 12(2):e1004792.

247. Spiers, H. J. and Barry, C. (2015). Neural systems supporting navigation. *Current Opinion in Behavioral Sciences*, 1:47–55.

248. Stachenfeld, K. L., Botvinick, M., and Gershman, S. J. (2014). Design Principles of the Hippocampal Cognitive Map. *Advances in Neural Information Processing Systems*, 27.

249. Stoyan, D. (1987). Berry, D. A. and B. Fristedt: Bandit Problems. Sequential Allocation of Experiments. Monographs on Statistics and Applied Probability. Chapman and Hall, London/New York 1985, 275 S. *Biometrical Journal*, 29(1):20–20.

250. Summaira, J., Li, X., Shoib, A. M., Li, S., and Abdul, J. (2021). Recent Advances and Trends in Multimodal Deep Learning: A Review. *arXiv:2105.11087 [cs]*.

251. Sussillo, D. (2014). Neural circuits as computational dynamical systems. *Current Opinion in Neurobiology*, 25:156–163.

252. Sussillo, D. and Abbott, L. (2009). Generating coherent patterns of activity from chaotic neural networks. *Neuron*, 63(4):544–557.

253. Sussillo, D. and Barak, O. (2012). Opening the Black Box: Low-Dimensional Dynamics in High-Dimensional Recurrent Neural Networks. *Neural Computation*, 25(3):626–649.

254. Sutton, R. S. and Barto, A. G. (1998). Reinforcement learning: an introduction. Adaptive Computation and Machine Learning series. MIT press,, Cambridge (Mass.), Etats-Unis d'Amérique, Royaume-Uni de Grande-Bretagne et d'Irlande du Nord.

255. Tanaka, G., Yamane, T., Héroux, J. B., Nakane, R., Kanazawa, N., Takeda, S., Numata, H., Nakano, D., and Hirose, A. (2019). Recent advances in physical reservoir computing: A review. *Neural Networks*, 115:100–123.

256. Taube, J. S., Muller, R. U., and Ranck, J. B. (1990). Head-direction cells recorded from the postsubiculum in freely moving rats. I. Description and quantitative analysis. *Journal of Neuroscience*, 10(2):420–435.

257. Team, O. E. L., Stooke, A., Mahajan, A., Barros, C., Deck, C., Bauer, J., Sygnowski, J., Trebacz, M., Jaderberg, M., Mathieu, M., McAleese, N., Bradley-Schmieg, N., Wong, N., Porcel, N., Raileanu, R., Hughes-Fitt, S., Dalibard, V., and Czarnecki, W. M. (2021). Open-Ended Learning Leads to Generally Capable Agents. *arXiv:2107.12808 [cs]*.

258. Tieleman, T. (2008). Training restricted Boltzmann machines using approximations to the likelihood gradient. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pages 1064–1071, New York, NY, USA. Association for Computing Machinery.

259. Todorov, E., Erez, T., and Tassa, Y. (2012). MuJoCo: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033.

260. Tolman, E. C. (1948). Cognitive maps in rats and men. *Psychological Review*, 55(4):189–208.

261. Tubiana, J. and Monasson, R. (2017). Emergence of Compositional Representations in Restricted Boltzmann Machines. *Physical Review Letters*, 118(13):138301.

262. Turner-Evans, D., Wegener, S., Rouault, H., Franconville, R., Wolff, T., Seelig, J. D., Druckmann, S., and Jayaraman, V. (2017). Angular velocity integration in a fly heading circuit. *eLife*, 6:e23496.

263. Uria, B., Ibarz, B., Banino, A., Zambaldi, V., Kumaran, D., Hassabis, D., Barry, C., and Blundell, C. (2020). The Spatial Memory Pipeline: A model of egocentric to allocentric understanding in mammalian brains. Preprint, Neuroscience.

264. Varga, A. and Moore, R. (1990). Hidden Markov model decomposition of speech and noise. In *International Conference on Acoustics, Speech, and Signal Processing*, pages 845–848 vol.2.

265. Vinyals, O., Ewalds, T., Bartunov, S., Georgiev, P., Vezhnevets, A. S., Yeo, M., Makhzani, A., Küttler, H., Agapiou, J., Schrittwieser, J., Quan, J., Gaffney, S., Petersen, S., Simonyan, K., Schaul, T., van Hasselt, H., Silver, D., Lillicrap, T., Calderone, K., Keet, P., Brunasso, A., Lawrence, D., Ekermo, A., Repp, J., and Tsing, R. (2017). StarCraft II: A New Challenge for Reinforcement Learning. *arXiv:1708.04782 [cs]*.

266. Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., Carey, C. J., Polat, I., Feng, Y., Moore, E. W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E. A., Harris, C. R., Archibald, A. M., Ribeiro, A. H., Pedregosa, F., and van Mulbregt, P. (2020). SciPy 1.0: Fundamental algorithms for scientific computing in Python. *Nature Methods*, 17(3):261–272.

267. Von Neumann, J. (2016). Probabilistic logics and the synthesis of reliable organisms from unreliable components. In *Automata Studies.(AM-34), Volume 34*, pages 43–98. Princeton University Press.

268. Wainwright, M. J. and Jordan, M. I. (2008). Graphical Models, Exponential Families, and Variational Inference. *Foundations and Trends® in Machine Learning*, 1(1-2):1–305.

269. Wan, L., Zeiler, M., Zhang, S., Cun, Y. L., and Fergus, R. (2013). Regularization of Neural Networks using DropConnect. In *International Conference on Machine Learning*, pages 1058–1066. PMLR.

270. Ward, T., Bolt, A., Hemmings, N., Carter, S., Sanchez, M., Barreira, R., Noury, S., Anderson, K., Lemmon, J., Coe, J., Trochim, P., Handley, T., and Bolton, A. (2020). Using Unity to Help Solve Intelligence. *arXiv:2011.09294 [cs]*.

271. Warren, W. C., Clayton, D. F., Ellegren, H., Arnold, A. P., Hillier, L. W., Künstner, A., Searle, S., White, S., Vilella, A. J., Fairley, S., Heger, A., Kong, L., Ponting, C. P., Jarvis, E. D., Mello, C. V., Minx, P., Lovell, P., Velho, T. A. F., Ferris, M., Balakrishnan, C. N., Sinha, S., Blatti, C., London, S. E., Li, Y., Lin, Y.-C., George, J., Sweedler, J., Southey, B., Gunaratne, P., Watson, M., Nam, K., Backstrom, N., Smeds, L., Nabholz, B., Itoh, Y., Whitney, O., Pfenning, A. R., Howard, J., Volker, M., Skinner, B. M., Griffin, D. K., Ye, L., McLaren, W. M., Flicek, P., Quesada, V., Velasco, G., Lopez-Otin, C., Puente, X. S., Olender, T., Lancet, D., Smit, A. F. A., Hubley, R., Konkel, M. K., Walker, J. A., Batzer, M. A., Gu, W., Pollock, D. D., Chen, L., Cheng, Z., Eichler, E. E., Stapley, J., Slate, J., Ekblom, R., Birkhead, T., Burke, T., Burt, D., Scharff, C., Adam, I., Richard, H., Sultan, M., Soldatov, A., Lehrach, H., Edwards, S. V., Yang, S.-P., Li, X., Graves, T., Fulton, L., Nelson, J., Chinwalla, A., Hou, S., Mardis, E. R., and Wilson, R. K. (2010). The genome of a songbird. *Nature*, 464(7289):757–762.

272. Watkins, C. J. C. H. and Dayan, P. (1992). Q-learning. *Machine Learning*, 8(3):279–292.

273. Whittington, J. C. R., Muller, T. H., Mark, S., Chen, G., Barry, C., Burgess, N., and Behrens, T. E. J. (2020). The Tolman-Eichenbaum Machine: Unifying Space

and Relational Memory through Generalization in the Hippocampal Formation. *Cell*, 183(5):1249–1263.e23.

274. Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3):229–256.

275. Wise, S. P. and Murray, E. A. (2000). Arbitrary associations between antecedents and actions. *Trends in Neurosciences*, 23(6):271–276.

276. Wishart, J. (1928). The generalised product moment distribution in samples from a normal multivariate population. *Biometrika*, 20A(1-2):32–52.

277. Wolpert, D. M. and Miall, R. C. (1996). Forward Models for Physiological Motor Control. *Neural Networks: The Official Journal of the International Neural Network Society*, 9(8):1265–1279.

278. Wolpert, D. M., Miall, R. C., and Kawato, M. (1998). Internal models in the cerebellum. *Trends in Cognitive Sciences*, 2(9):338–347.

279. Wong, K.-F., Huk, A. C., Shadlen, M. N., and Wang, X.-J. (2007). Neural circuit dynamics underlying accumulation of time-varying evidence during perceptual decision making. *Frontiers in Computational Neuroscience*, 1.

280. Wong, K.-F. and Wang, X.-J. (2006). A Recurrent Network Mechanism of Time Integration in Perceptual Decisions. *Journal of Neuroscience*, 26(4):1314–1328.

281. Wright, S. J. (2015). Coordinate descent algorithms. *Mathematical Programming*, 151(1):3–34.

282. Wu, F. Y. (1982). The Potts model. *Reviews of Modern Physics*, 54(1):235–268.

283. Wydmuch, M., Kempka, M., and Jaśkowski, W. (2018). ViZDoom Competitions: Playing Doom from Pixels. *arXiv:1809.03470 [cs, stat]*.

284. Yang, G. R., Joglekar, M. R., Song, H. F., Newsome, W. T., and Wang, X.-J. (2019). Task representations in neural networks trained to perform many cognitive tasks. *Nature Neuroscience*, 22(2):297–306.

285. Yang, G. R. and Wang, X.-J. (2020). Artificial Neural Networks for Neuroscientists: A Primer. *Neuron*, 107(6):1048–1070.

286. Young, J. Z. (1938). The Functioning of the Giant Nerve Fibres of the Squid. *Journal of Experimental Biology*, 15(2):170–185.

287. Zaremba, W., Sutskever, I., and Vinyals, O. (2015). Recurrent Neural Network Regularization. *arXiv:1409.2329 [cs]*.

288. Zeigler, H. P. and Marler, P., editors (2004). Behavioral neurobiology of birdsong. Behavioral Neurobiology of Birdsong. New York Academy of Sciences, New York, NY, US.

289. Zhang, J., Tai, L., Liu, M., Boedecker, J., and Burgard, W. (2020). Neural SLAM: Learning to Explore with External Memory. *arXiv:1706.09520 [cs]*.

290. Zhang, Q.-s. and Zhu, S.-c. (2018). Visual interpretability for deep learning: A survey. *Frontiers of Information Technology & Electronic Engineering*, 19(1):27–39.

291. Zhao, D., Zhang, Z., Lu, H., Cheng, S., Si, B., and Feng, X. (2020). Learning Cognitive Map Representations for Navigation by Sensory-Motor Integration. *IEEE transactions on cybernetics*.

Appendix A

Recurrent Neural Integrators

A.1 Fully averaged loss for linear single-channel integrators

For an arbitrary input sequence $(x_t)_{0\leq t\leq T-1}$, we compute through induction the value of the output at any time t as:

$$\forall t \in \mathbb{N},\, y_t = \sum_{q=0}^{t} x_{t-q} \boldsymbol{d}^T \boldsymbol{W}^{q+1} \boldsymbol{e} := \sum_{q=0}^{t} x_{t-q} \mu_{q+1} \tag{A.1}$$

The target output is $\overline{y}_t = s\sum_{q=0}^{t} x_{t-q}\gamma^{q+1}$, so that the square error is:

$$\begin{aligned} \epsilon_t^2 &= (y_t - \overline{y}_t)^2 \\ &= [\sum_{q=0}^{t} x_{t-q}(\mu_{q+1} - s\gamma^{q+1})]^2 \\ &= \sum_{q,p=0}^{t} x_{t-q}x_{t-p}(\mu_{q+1} - s\gamma^{q+1})(\mu_{p+1} - s\gamma^{p+1}) \end{aligned} \tag{A.2}$$

The loss to minimize is the average of the sum of those errors along input sequences of length T:

$$\begin{aligned} \mathcal{L}(\boldsymbol{W}) &= \left\langle \sum_{t=0}^{T-1} \epsilon_t^2 \right\rangle = \left\langle \sum_{t=0}^{T-1} \sum_{p,q=0}^{t} x_{t-q}x_{t-p}(\mu_{q+1} - s\gamma^{q+1})(\mu_{p+1} - s\gamma^{p+1}) \right\rangle \\ &= \left\langle \sum_{t=0}^{T-1} \sum_{p,q=0}^{T-1} x_{t-q}x_{t-p}\mathbf{1}_{q\leq t}\mathbf{1}_{p\leq t} \right\rangle (\mu_{q+1} - s\gamma^{q+1})(\mu_{p+1} - s\gamma^{p+1}) \\ &= \sum_{p,q=0}^{T-1} \left\langle \sum_{t=0}^{T-1} x_{t-q}x_{t-p}\mathbf{1}_{q\leq t}\mathbf{1}_{p\leq t} \right\rangle (\mu_{q+1} - s\gamma^{q+1})(\mu_{p+1} - s\gamma^{p+1}) \\ &:= \sum_{p,q=1}^{T} \boldsymbol{\chi}_{qp}(\mu_q - s\gamma^q)(\mu_p - s\gamma^p) \end{aligned} \tag{A.3}$$

where we introduced the time-integrated correlation matrix $\boldsymbol{\chi}$.

$\boldsymbol{\chi}$ is symmetric, and it is easily shown that:

$$\begin{aligned}\forall \boldsymbol{v} \in \mathbb{R}^T, \boldsymbol{v}^\dagger \boldsymbol{\chi} \boldsymbol{v} &= \sum_{p,q=0}^{T-1} v_q \chi_{qp} v_p = \sum_{p,q=0}^{T-1} \left\langle \sum_{t=0}^{T-1} v_q x_{t-q} x_{t-p} \mathbf{1}_{q\leq t} \mathbf{1}_{p\leq t} v_p \right\rangle \\ &= \left\langle \sum_{t=0}^{T-1} (\sum_{q=0}^{T-1} v_q x_{t-q} \mathbf{1}_{q\leq t})(\sum_{p=0}^{T-1} x_{t-p} \mathbf{1}_{p\leq t} v_p) \right\rangle \\ &= \sum_{t=0}^{T-1} \left\langle \left(\sum_{p=0}^{t} x_{t-p}\, v_p \right)^2 \right\rangle \ .\end{aligned} \tag{A.4}$$

Therefore, $\boldsymbol{\chi}$ is non-negative. Assuming now that $\boldsymbol{v}$ is such that the quadratic form above vanishes, the term corresponding to $t = 0$, equal to $v_0^2 \langle x_0^2 \rangle$ vanishes, entailing that $v_0 = 0$ as soon as the input is assumed to have positive probability to be non-zero at this first time-step. Then, the $t = 1$ contribution, $\langle (x_0\, v_1 + x_1\, v_0)^2 \rangle = \langle (x_0\, v_1)^2 \rangle$ also vanishes, which implies that $v_1 = 0$. By recursion over t, all the components of $\boldsymbol{v}$ must vanish, which shows that $\boldsymbol{\chi}$ is definite positive.

A.2 Gradient and Hessian of the linear single channel loss

We have:

$$\begin{aligned}\nabla_{W_{ij}} \mathcal{L} &= \sum_{q,p=1}^{T} \chi_{qp} \left[(\mu_q - s\gamma^q) \frac{\partial \mu_p}{\partial W_{ij}} + (\mu_p - s\gamma^p) \frac{\partial \mu_q}{\partial W_{ij}} \right] \\ &= 2 \sum_{q,p=1}^{T} \chi_{qp} (\mu_q - s\gamma^q) \frac{\partial \mu_p}{\partial W_{ij}}.\end{aligned} \tag{A.5}$$

We compute through induction:

$$\frac{\partial W_{ij}^p}{\partial W_{kl}} = \sum_{m=0}^{p-1} W_{ik}^m W_{lj}^{p-1-m} \quad \text{hence} \quad \frac{\partial \mu_p}{\partial W_{kl}} = \sum_{i,j=1}^{n} \sum_{m=0}^{p-1} d_i W_{ik}^m W_{lj}^{p-1-m} e_j \tag{A.6}$$

So that the gradient of $\mathcal{L}$ with respect to $\boldsymbol{W}$ is:

$$\nabla_{W_{ij}} \mathcal{L} = 2 \sum_{q,p=1}^{T} \chi_{qp} (\mu_q - s\gamma^q) \sum_{m=0}^{p-1} \sum_{\alpha,\beta} (d_\alpha W_{\alpha i}^m)(W_{j\beta}^{p-1-m} e_\beta) \tag{A.7}$$

We now want to compute the Hessian $\mathcal{H}$ of this loss:

$$\begin{aligned}\mathcal{H}_{ij,kl} = \frac{\partial \mathcal{L}}{\partial W_{ij} \partial W_{kl}} &= 2 \sum_{q,p=1}^{T} \chi_{qp} (\mu_q - s\gamma^q) \frac{\partial}{\partial W_{kl}} \left[\sum_{m=0}^{p-1} \sum_{\alpha,\beta} (d_\alpha W_{\alpha i}^m)(W_{j\beta}^{p-1-m} e_\beta) \right] \\ &+ 2 \sum_{q,p=1}^{T} \chi_{qp} \frac{\partial \mu_q}{\partial W_{kl}} \sum_{m=0}^{p-1} \sum_{\alpha,\beta} (d_\alpha W_{\alpha i}^m)(W_{j\beta}^{p-1-m} e_\beta)\end{aligned} \tag{A.8}$$

We will only be interested in the value of the Hessian at global minima of $\mathcal{L}$, so that the first term in this equation will not contribute. In that case, we find:

$$\mathcal{H}_{ij,kl} = 2 \sum_{q,p=1}^{T} \chi_{qp} \left[\sum_{m=0}^{p-1} \sum_{\alpha,\beta} (d_\alpha W_{\alpha i}^m)(W_{j\beta}^{p-1-m} e_\beta) \right] \left[\sum_{\tilde{\alpha},\tilde{\beta}=1}^{n} \sum_{\tilde{m}=0}^{q-1} d_{\tilde{\alpha}} W_{\tilde{\alpha}k}^{\tilde{m}} W_{l\tilde{\beta}}^{q-1-\tilde{m}} e_{\tilde{\beta}} \right] \tag{A.9}$$

A.3 Two special cases of null initialization

Exact solution for $T = 1$. We begin by the simplest case possible, when the epochs are of length 1. The gradient descent updates become in that case:

$$\Delta W_{ij} = -2\eta(\boldsymbol{d}^\dagger \boldsymbol{W} \boldsymbol{e} - s\gamma) d_i e_j \tag{A.10}$$

Hence, we have at any time $\boldsymbol{W} = \omega \boldsymbol{d}\boldsymbol{e}^T$, so that we can study the optimization dynamics on the scalar ω only :

$$\Delta\omega = -2\eta(\omega ||\boldsymbol{e}||^2 ||\boldsymbol{d}||^2 - s\gamma) \tag{A.11}$$

Therefore, after τ steps of optimization, the coefficient ω is equal to

$$\omega^{(\tau)} = \frac{s\gamma}{||\boldsymbol{e}||^2||\boldsymbol{d}||^2}[1 - (1 - 2\eta||\boldsymbol{e}||^2||\boldsymbol{d}||^2)^\tau] \tag{A.12}$$

This dynamics is stable if and only if $\eta < ||\boldsymbol{e}||^{-2}||\boldsymbol{d}||^{-2}$. When it is, the network converges exponentially fast to $\boldsymbol{W} = \frac{\gamma s}{||\boldsymbol{e}||^2||\boldsymbol{d}||^2}\boldsymbol{d}\boldsymbol{e}^T$, which gives the following moments :

$$\forall k \geq 1, \mu_k = \gamma s (\frac{\gamma s\, \boldsymbol{e}\cdot\boldsymbol{d}}{||\boldsymbol{e}||^2||\boldsymbol{d}||^2})^{k-1} \tag{A.13}$$

Therefore, in that case, we converge towards a solution that achieves the desired scaling s, but has a decay constant that is fixed by the initial choice of s, $\boldsymbol{e}$ and $\boldsymbol{d}$.

Same encoder and decoder, $T > 1$, uncorrelated inputs. For this part, we assume that $\boldsymbol{e} = \boldsymbol{d}$. Considering that the first update in that case is proportional to $\boldsymbol{e}\boldsymbol{e}^T$, all subsequent ones are too, so we know that $\boldsymbol{W}^{(t)} = \omega(t)\boldsymbol{e}\boldsymbol{e}^\dagger$ and the dynamics can be studied on the scalar ω only.

Because of this, all moments are given by $\mu_k = \omega^k ||\boldsymbol{e}||^{2k+2} = ||\boldsymbol{e}||^2(\omega||\boldsymbol{e}||^2)^k$. The corresponding scale and decay are respectively $||\boldsymbol{e}||^2$ and $\omega||\boldsymbol{e}||^2$, and because of this it is only possible to obtain a Generalizing Integrator at scale $s = ||\boldsymbol{e}||^2$.

The fixed points of the gradient descent dynamics are the real roots of the following polynomial P:

$$\frac{\Delta\omega}{\eta} = 2\sum_{k=1}^{T} k(T+1-k)(\omega^k||\boldsymbol{e}||^{2k+2} - s\gamma^k)\omega^{k-1}||\boldsymbol{e}||^{2k-2} := P(\omega) \tag{A.14}$$

Choosing $s = ||\boldsymbol{e}||^2$, we can check with Mathematica that for any value of T larger than 2, this polynomial has a single real root at $\omega = \gamma/||\boldsymbol{e}||^2$, which is a generalizable minimum. Since the leading order term in this polynomial is of odd degree, we know that $\lim_{w\to\pm\infty} P(\omega) = \pm\infty$, and therefore the derivative of P at $\omega = \gamma/||\boldsymbol{e}||^2$ is positive, so that this value of ω is an attractive fixed-point of the dynamics. As before, the dynamics is convergent only if the learning rate is smaller than $P'(\gamma/||\boldsymbol{e}||^2)^{-1}$.

A.4 Moments in the low-rank parametrization

We have defined ω so that

$$\boldsymbol{W} = \sum_{a,b=1}^{2} \omega_{a,b}\overline{\boldsymbol{v}_a}\,\overline{\boldsymbol{v}_b}^\dagger. \tag{A.15}$$

Because of the orthogonality conditions $\overline{\boldsymbol{v}_a}^\dagger \overline{\boldsymbol{v}_b} = \delta_{a,b}$, we can easily compute the powers of $\boldsymbol{W}$ as:

$$\boldsymbol{W}^k = \sum_{a,b=1}^{2} (\omega^k)_{a,b} \overline{\boldsymbol{v}_a}\, \overline{\boldsymbol{v}_b}^\dagger, \tag{A.16}$$

which yields the following moments:

$$\begin{aligned} \mu_k &= \boldsymbol{d}^\dagger \boldsymbol{W}^k \boldsymbol{e} = \boldsymbol{d}^\dagger \sum_{a,b=1}^{2} \omega_{a,b}^k \overline{\boldsymbol{v}_a}\, \overline{\boldsymbol{v}_b}^\dagger \boldsymbol{e} \\ &= \sum_{a,b=1}^{2} \sqrt{\boldsymbol{\Sigma}}_{1,a} \omega_{a,b}^k \sqrt{\boldsymbol{\Sigma}}_{b,2} = (\sqrt{\boldsymbol{\Sigma}} \omega^k \sqrt{\boldsymbol{\Sigma}})_{1,2} \end{aligned} \tag{A.17}$$

We now assume ω to be diagonalizable as $\omega = \boldsymbol{P}_\omega \boldsymbol{\Lambda}_\omega \boldsymbol{P}_\omega^{-1}$, so that:

$$\begin{aligned} \mu_k &= (\sqrt{\boldsymbol{\Sigma}} \omega^k \sqrt{\boldsymbol{\Sigma}})_{1,2} = (\sqrt{\boldsymbol{\Sigma}} \boldsymbol{P}_\omega \boldsymbol{\Lambda}_\omega^k \boldsymbol{P}_\omega^{-1} \sqrt{\boldsymbol{\Sigma}})_{1,2} \\ &= \sum_{i=1}^{2} \lambda_i^k (\sqrt{\boldsymbol{\Sigma}} \boldsymbol{P}_\omega)_{1,i} (\boldsymbol{P}_\omega^{-1} \sqrt{\boldsymbol{\Sigma}})_{i,2} = \sum_{i=1}^{2} \lambda_i^k (\boldsymbol{P}_\omega^\dagger \sqrt{\boldsymbol{\Sigma}})_{i,1} (\boldsymbol{P}_\omega^{-1} \sqrt{\boldsymbol{\Sigma}})_{i,2} \\ &:= \sum_{i=1}^{2} g_i \lambda_i^k \end{aligned} \tag{A.18}$$

Using a reasoning similar to the one of Section 7.3.1, we find the same conditions (7.14) for Generalizing Integrators, but with a new expression of the g coefficients.

A.5 Generalizing Integrators in null initialization space

In this section, we seek to determine all matrices ω that correspond to Generalizing Integrators at decay γ. A first, obvious condition is that at least one of the eigenvalues of ω has to be equal to γ. Without loss of generality, we will consider this eigenvalue to be the first one, and we will denote the other λ.

The matrix $\boldsymbol{P}_\omega$ that diagonalizes ω can be parametrized as:

$$\boldsymbol{P}_\omega = \begin{pmatrix} u_1 & v_1 \\ u_2 & v_2 \end{pmatrix} \tag{A.19}$$

Each column of $\boldsymbol{P}_\omega$ can be independently multiplied by a non-zero scalar and yield the same ω. There are therefore three cases: either u_1 or v_1 is null (but not both, since $\boldsymbol{P}$ would then not be invertible), or both are non-zero.

We also recall that

$$g_1 + g_2 = \sum_{i=1}^{2} [(\sqrt{\Sigma} \boldsymbol{P}_\omega)_{1,i} (\boldsymbol{P}_\omega^{-1} \sqrt{\Sigma})_{i,2}] = \Sigma_{1,2} = \boldsymbol{d}^\dagger \boldsymbol{e} \tag{A.20}$$

Case $u_1 \neq 0$ and $v_1 \neq 0$.

In that case, the modal matrix $\boldsymbol{P}_\omega$ of $\omega^\perp$ will be parametrized as follows, with $\alpha \neq \beta$ to ensure invertibility:

$$\boldsymbol{P}_\omega = \begin{pmatrix} 1 & 1 \\ \alpha & \beta \end{pmatrix} \tag{A.21}$$

This results in the following parametrization of the space E_γ of two by two matrices with at least one eigenvalue equal to γ :

$$\begin{aligned} E_\gamma &= \left\{ \Gamma(\lambda,\alpha,\beta) = \begin{pmatrix} 1 & 1 \\ \alpha & \beta \end{pmatrix} \begin{pmatrix} \gamma & 0 \\ 0 & \lambda \end{pmatrix} \begin{pmatrix} 1 & 1 \\ \alpha & \beta \end{pmatrix}^{-1} ; (\lambda,\alpha,\beta) \in \mathbb{R}^3, \alpha \neq \beta \right\} \\ &= \left\{ \frac{1}{\alpha-\beta} \begin{pmatrix} \alpha\lambda - \beta\gamma & \gamma - \lambda \\ \alpha\beta(\lambda-\gamma) & \alpha\gamma - \beta\lambda \end{pmatrix} ; (\lambda,\alpha,\beta) \in \mathbb{R}^3, \alpha \neq \beta \right\} \end{aligned} \tag{A.22}$$

If $\lambda = \gamma$: All values of α and β correspond to $\gamma\mathbf{1}$, a perfect integrator at scale $s^* = \boldsymbol{d}^\dagger \boldsymbol{e}$.

If $\lambda \neq \gamma$ and $\lambda \neq 0$: In that case, we need to impose $g_2(\alpha,\beta) = 0$, otherwise the second eigenvalue will contribute to the output and the system will not be a Generalizing Integrator. This will in turn impose that $g_1 = \boldsymbol{d}^\dagger \boldsymbol{e}$, and hence these will be integrators at scale $s = \boldsymbol{d}^\dagger \boldsymbol{e}/\gamma$. We find that:

$$g_2[\alpha,\beta] = Z \frac{(\alpha-\beta_0)(\beta-\alpha_0)}{\alpha-\beta} \tag{A.23}$$

where:

$$\begin{cases} \alpha_0 &= -\dfrac{(\kappa-l_-)r_- + (\kappa+l_-)r_+}{2\boldsymbol{d}^\dagger\boldsymbol{e}(r_+-r_-)} \\ \beta_0 &= \dfrac{(\kappa+l_-)r_- + (\kappa-l_-)r_+}{2\boldsymbol{d}^\dagger\boldsymbol{e}(r_+-r_-)} \\ Z &= \dfrac{[\boldsymbol{d}^\dagger\boldsymbol{e}(r_+-r_-)]^2}{2\kappa^2} \\ l_\pm &= ||\boldsymbol{d}||^2 \pm ||\boldsymbol{e}||^2 \\ \kappa &= \sqrt{l_-^2 + 4(\boldsymbol{d}^\dagger\boldsymbol{e})^2} \in]0, l_+[\\ r_\pm &= \sqrt{l_+ \pm \kappa} \end{cases} \tag{A.24}$$

Hence, there are two manifolds of points satisfying $g_2(\alpha,\beta) = 0$:

$$\begin{cases} \mathcal{M}_\alpha = \{\Gamma(\lambda,\alpha,\alpha_0), (\lambda,\alpha) \in \mathbb{R}^2\} \\ \mathcal{M}_\beta = \{\Gamma(\lambda,\beta_0,\beta), (\lambda,\beta) \in \mathbb{R}^2\} \end{cases} \tag{A.25}$$

where Γ is defined in equation (A.22). These two manifolds intersect along a 1-Dimensional manifold:

$$\mathcal{M}_\alpha \cap \mathcal{M}_\beta = \{\Gamma(\lambda,\beta_0,\alpha_0), \lambda \in \mathbb{R}\} \tag{A.26}$$

If $\lambda = 0$: The system will always be a perfect integrator at decay γ, since no other eigenvalue can contribute to the output. Its scale is determined by :

$$s = g_1(\alpha,\beta) = Z \frac{(\alpha-\alpha_0)(\beta-\beta_0)}{\beta-\alpha} \tag{A.27}$$

From this result, we deduce the following:

- For any given value of $s \notin \{0, \boldsymbol{d}^\dagger\boldsymbol{e}\}$, the set of values of α,β that give $c_1[\alpha,\beta] = s$ is a one-dimensional manifold, as can be seen in Figure A.1. A parametrization of

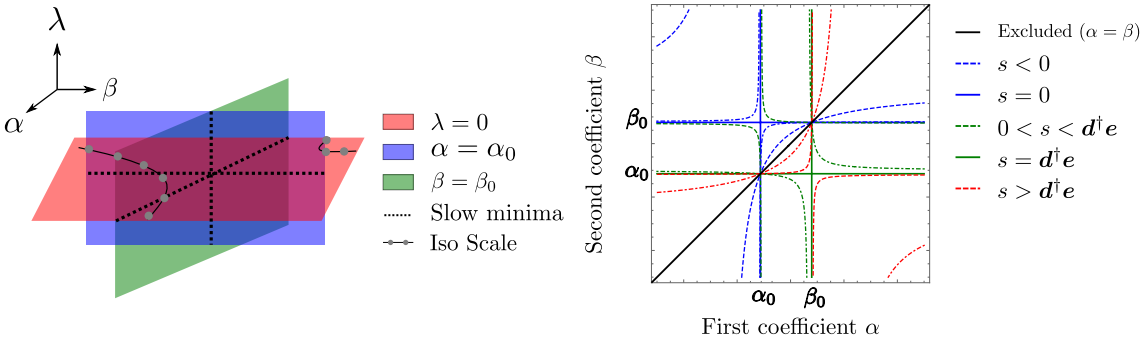


Figure A.1: Structure of the minima in the null initialization subspace. On the left, we present the structure of the manifolds of 2×2–matrices with exactly one eigenvalue equal to γ, and both eigenvectors with non-zero first components. The coefficients α and β parametrize the eigenvectors, and λ is the second eigenvalue. The "slow minima", which are located at the intersections of our manifolds, are the ones towards which convergence of Gradient Descent can be algebraically slow, see Section A.8. On the right, we detail the structure of the isoscale manifolds in the $\lambda = 0$ submanifold, and show that they are indeed one-dimensional as long as $s \notin \{0, \boldsymbol{d}^\dagger \boldsymbol{e}\}$. While the lines appear to cross, they do so only on the $\alpha = \beta$ line which is a singularity of our parametrization and therefore non-physical.

this manifold can be obtained by inverting equation (A.27) as the set $\Gamma(0, \alpha, \beta_s(\alpha))$ where Γ is defined in equation (A.22) and:

$$\beta_s(\alpha) = \frac{Z(\alpha - \alpha_0)\beta_0 - \alpha s}{Z(\alpha - \alpha_0) - s}. \tag{A.28}$$

- When $s = 0$, the two solutions are $\alpha = \alpha_0$ or $\beta = \beta_0$, no matter the value of the other parameter.
- When $s = \boldsymbol{d}^\dagger \boldsymbol{e}$, equation (A.27) is not invertible and the condition $g_1 = s$ is satisfied if and only if $\beta = \alpha_0$ or $\alpha = \beta_0$, which are exactly the intersection of the $\mathcal{M}_\alpha$ (resp. $\mathcal{M}_\beta$) manifolds of equation (A.25) with the $\lambda = 0$ subspace.

From ω to $\boldsymbol{W}$

We have shown that in the generic case $s \notin \{0, \boldsymbol{d}^\dagger \boldsymbol{e}\}$, the Generalizing Integrators at scale s correspond to ω of rank 1 and form a 1–dimensional manifold.

Such matrices ω can be parametrized as:

$$\begin{aligned}
\mathcal{M}_{\lambda=0} &= \left\{ \frac{\gamma}{\beta - \alpha} \begin{pmatrix} \beta & -1 \\ \alpha\beta & -\alpha \end{pmatrix} ; (\alpha, \beta) \in \mathbb{R}^2, \alpha \neq \beta \right\} \\
&= \Bigg\{ \sigma_\omega \boldsymbol{x}\boldsymbol{y}^\dagger; \quad \sigma_\omega = \frac{\gamma\sqrt{(\alpha^2+1)(\beta^2+1)}}{\beta - \alpha}, \\
&\qquad \boldsymbol{x} = \frac{1}{\sqrt{\alpha^2+1}} \begin{pmatrix} 1 \\ \alpha \end{pmatrix}, \quad \boldsymbol{y} = \frac{1}{\sqrt{\beta^2+1}} \begin{pmatrix} \beta \\ -1 \end{pmatrix}, \\
&\qquad (\alpha, \beta) \in \mathbb{R}^2, \alpha \neq \beta \Bigg\}
\end{aligned} \tag{A.29}$$

where $\boldsymbol{x}$ and $\boldsymbol{y}$ are respectively the left and right eigenvector of the corresponding rank 1 matrix. When ω is of that form, we have:

$$\boldsymbol{W} = \sigma_\omega \sum_{a,b} \boldsymbol{x}_a \boldsymbol{y}_b \overline{\boldsymbol{v}_a}\, \overline{\boldsymbol{v}_b}^\dagger = \sigma_\omega (\sum_a \boldsymbol{x}_a \overline{\boldsymbol{v}_a})(\sum_b \boldsymbol{y}_b \overline{\boldsymbol{v}_b}^\dagger) := \sigma \boldsymbol{l}\boldsymbol{r}^\dagger, \tag{A.30}$$

so that $\boldsymbol{W}(\omega)$ is of rank 1 too.

Case $u_1 = 0$

The matrix $\boldsymbol{P}_\omega$ that diagonalizes ω will be parametrized as:

$$\boldsymbol{P}_\omega = \begin{pmatrix} 0 & 1 \\ 1 & v \end{pmatrix} \tag{A.31}$$

and is always invertible no matter the value of $v \neq 0$.

Now, there are two degrees of freedom λ and v, and the corresponding matrices are:

$$\omega = \begin{pmatrix} \lambda & 0 \\ (\gamma - \lambda)v & \gamma \end{pmatrix} \tag{A.32}$$

As before, the scale s is determined by c_γ, which depends linearly on v. Hence, the choice of scale fixes v, and either λ has to be chosen either equal to 0 when s is different from $\boldsymbol{d}^\dagger \boldsymbol{e}$ (yielding a single solution) or it remains free if the choice of scale imposes $c_\lambda = 0$ (yielding a $1D$ manifold).

A perfectly analogous reasoning can be applied when $v_1 = 0$ and $u_1 \neq 0$, and in both cases the manifolds of solution are of lower dimension than their counterparts which have non-zero u_1 and u_2; we discard those solutions as we expect them to be smooth limits of the generic case.

A.6 Gradients and Hessian in the low-rank parametrization

We will now explicitly compute the derivative of the loss with respect to the 2×2–matrix ω. We previously found that:

$$\begin{aligned} \mu_q &= \boldsymbol{d}^\dagger \boldsymbol{W}^q \boldsymbol{e} = (\sqrt{\boldsymbol{\Sigma}}\, \omega^q \sqrt{\boldsymbol{\Sigma}})_{12} \\ &= \sum_{a,b=1}^{2} \sqrt{\boldsymbol{\Sigma}}_{1a} \omega^q_{ab} \sqrt{\boldsymbol{\Sigma}}_{b2} \end{aligned} \tag{A.33}$$

and we have that:

$$\frac{\partial \omega^q_{a,b}}{\partial \omega_{ij}} = \sum_{m=0}^{q-1} \omega^m_{ai} \omega^{q-1-m}_{jb} \tag{A.34}$$

so that:

$$\frac{\partial \mu_q}{\partial \omega_{ij}} = \sum_{m=0}^{q-1} (\sqrt{\boldsymbol{\Sigma}} \omega^m)_{1i} (\omega^{q-1-m} \sqrt{\boldsymbol{\Sigma}})_{j2} \tag{A.35}$$

which allows us to compute the gradient of the loss with respect to the coefficients of ω through:

$$\frac{\partial \mathcal{L}}{\partial \omega_{ij}} = 2 \sum_{q,p=1}^{T} \chi_{q,p} (\mu_q - s\gamma^q) \frac{\partial \mu_p}{\partial \omega_{ij}} \tag{A.36}$$

We can now compute the Hessian of the loss, which will allow us to derive formulas for stability and speed of convergence of Gradient Descent. This Hessian can be decomposed as follows:

$$\mathcal{H}_{ij,kl} = \frac{\partial \mathcal{L}}{\partial \omega_{ij} \partial \omega_{kl}} = \sum_{q,p=1}^{T} \chi_{q,p} \left[\frac{\partial \mu_q}{\partial \omega_{ij}} \frac{\partial \mu_p}{\partial \omega_{kl}} + (\mu_q - s\gamma^q) \frac{\partial \mu_p}{\partial \omega_{ij} \partial \omega_{kl}} \right] \tag{A.37}$$

We want to estimate this Hessian at rank 1 generalizing integrators, so that the second part will always be zero. Therefore, the Hessian is simply:

$$\begin{aligned} \mathcal{H}_{ij,kl} = \sum_{q,p=1}^{T} \chi_{q,p} & \left[\sum_{m=0}^{q-1} (\sqrt{\boldsymbol{\Sigma}} \omega^m)_{1i} (\omega^{q-1-m} \sqrt{\boldsymbol{\Sigma}})_{j2} \right] \\ & \times \left[\sum_{\tilde{m}=0}^{p-1} (\sqrt{\boldsymbol{\Sigma}} \omega^{\tilde{m}})_{1k} (\omega^{p-1-\tilde{m}} \sqrt{\boldsymbol{\Sigma}})_{l2} \right] \end{aligned} \tag{A.38}$$

A.7 Minimum convergence time

We are now interested in studying the dynamics of convergence towards the GIs $\boldsymbol{W}^*$ in the null initialization subspace. To do so, we use a Taylor expansion of the loss around one of its minima:

$$\mathcal{L}(\boldsymbol{W}^* + \delta \boldsymbol{W}) = \sum_{i,j,k,l=1}^{n} \delta \boldsymbol{W}_{ij} \, \mathcal{H}_{ij,kl}(\boldsymbol{W}^*) \, \delta \boldsymbol{W}_{kl} \tag{A.39}$$

Seeing $\delta \boldsymbol{W}$ as a vector and $\mathcal{H}$ as a (symmetric) matrix, we can diagonalize $\mathcal{H}$ with real eigenvalues λ_I and normalized eigenvectors $\boldsymbol{u}_I$, and express $\delta \boldsymbol{W} = \sum_{I=1}^{n^2} \delta_I \boldsymbol{u}_I$ in that basis so that:

$$\mathcal{L}(\boldsymbol{W}^* + \delta \boldsymbol{W}) = \sum_{I=1}^{n^2} \lambda_I \, \delta_I^2 \tag{A.40}$$

Since our loss is positive for any weight-matrix $\boldsymbol{W}$, we expect that all eigenvalues of the Hessian computed at a GI be positive. We also expect that (in the generic case $s \neq \boldsymbol{d}^\dagger \boldsymbol{e}$) one of them is 0, corresponding to the local tangent to the 1–dimensional manifold of GIs.

Writing the GD dynamics on the perturbation $\boldsymbol{\delta}$, we find that:

$$\delta_I^{(\tau+1)} = (1 - \eta \lambda_I) \delta_I^{(\tau)}, \tag{A.41}$$

hence $\boldsymbol{\delta}$ will see each of its components either be conserved (if it corresponds to a null eigenvalue) or evolve exponentially. This exponential evolution is convergent as long as $|1 - \eta \lambda_I| < 1$, and monotonic as long as $\eta < 1/\lambda_I$. Choosing the optimal learning rate for the full system $\eta^* = 1/\lambda_{\max}$, the slowest component of $\boldsymbol{\delta}$ evolves as

$$(1 - \eta^* \lambda_{min})^\tau = (1 - \lambda_{min}/\lambda_{max})^\tau \simeq e^{\tau \ln(1 - \mathcal{C}^{-1})} \simeq e^{-\tau/\mathcal{C}},$$

hence the characteristic convergence time will be $\mathcal{C} = \lambda_{max}/\lambda_{min}$[1]

Since in the null initialization case the weights are parametrized by a 2×2–matrix, the Hessian is 4×4 and its spectrum can easily be computed numerically by using equation

[1] λ_{min} is the minimum **non-zero** eigenvalue.

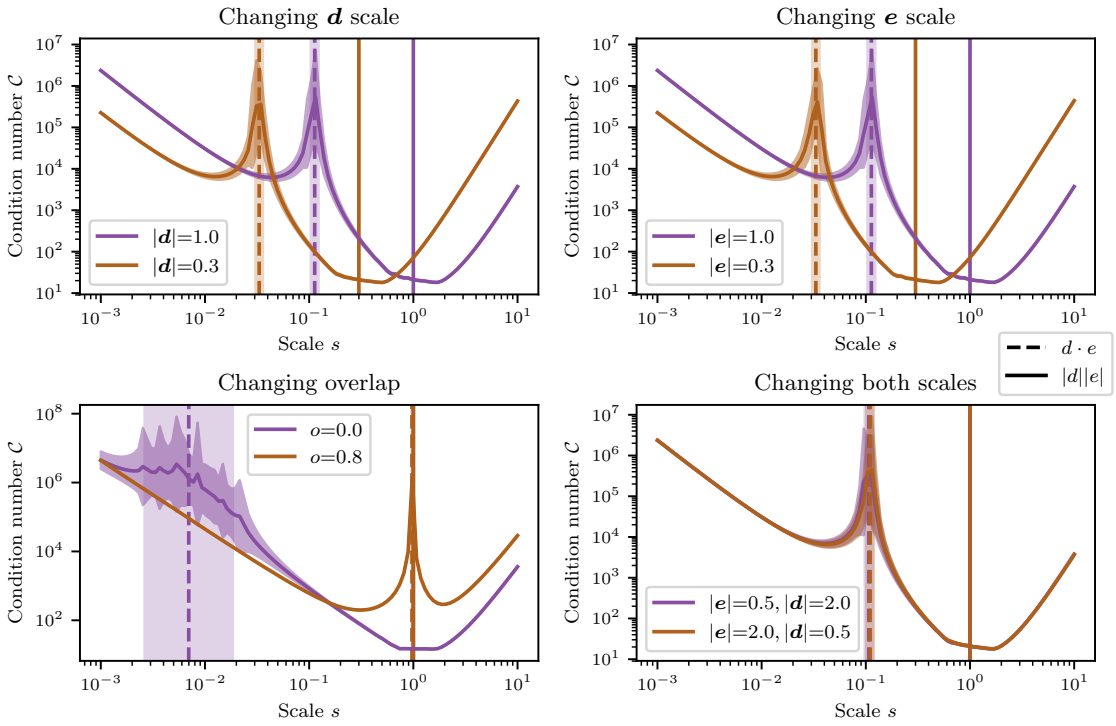


Figure A.2: Evolution of the minimum convergence time as a function of the scale.

(A.38). We therefore performed the following study: fixing the L_2 norm of the encoder and decoder as well as their dot product[2], we evaluate the spectrum of $\mathcal{H}$ and deduce from it the condition number $\mathcal{C}$ along each manifold of rank 1 s–scaled GIs; we find that a minimum exists for α, β (see Appendix A.5) of order 1, and this value will be a lower bound for the convergence time to **any** GI at scale s using Gradient Descent. We plot the value of this bound as a function of s for different initial choices of i/o–vectors in Figure A.2.

A.8 Algebraic convergence for specific scale value

From the previous analysis, it seems that when $s = \boldsymbol{d}^\dagger \boldsymbol{e}$ the $\lambda = 0$ manifold of solutions is hard to reach. Numerically, we see that Gradient Descent converges to a solution that lies in the union of the two $2D$ manifolds described earlier, corresponding to $\boldsymbol{W}$ of rank a priori 2. If we initialize with a random, non-zero, $\boldsymbol{W}$ in the null initialization subspace, we converge exponentially; if we start from $\boldsymbol{W} = 0$, the convergence is algebraic as a power law τ^{-2} instead of exponential (see Figure A.3).

To understand this phenomenon, we will consider the continuous-time, non-linear differential equation on the coefficients of ω: $\dot{\omega} = -\partial_\omega \mathcal{L}$. Let us also introduce the two

[2]In order to generate $\boldsymbol{e}$ and $\boldsymbol{d}$ with macroscopic overlaps (larger than $n^{-1/2}$), we first draw them independently, normalize them to 1, then modify the decoder as $\boldsymbol{d} = o\boldsymbol{e} + (1-o)\boldsymbol{d}$ where o is the overlap; for large enough n, the dot product $\boldsymbol{d}^\dagger \boldsymbol{e}$ will be close to this overlap. We then independently rescale them to the desired norm.

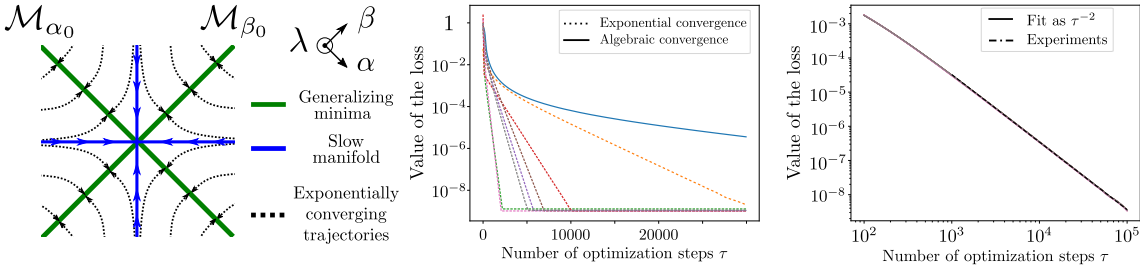


Figure A.3: Explanation of the dynamics of convergence towards a minimum at $s = \boldsymbol{d}^\dagger \boldsymbol{e}$. (Left) Informal representation of GD trajectories. Two types of trajectories converging to GIs exist: starting from a random $\boldsymbol{W}$ in the null initialization subspace (middle), we usually converge outside the intersection of the two manifolds exponentially fast, with a fairly wide range of times of convergence depending on the precise starting point; in rare cases, that random starting point lies on (or close enough to) the slow manifold on which convergence is as a power law τ^{-2}. We illustrate this algebraic convergence by starting from $\boldsymbol{W} = 0$ (right), which is experimentally found to be on the slow manifold. Both experimental curves show 8 different realizations of the training, with same learning rate, norms of vectors and overlap (but scale chosen exactly to $\boldsymbol{d}^\dagger \boldsymbol{e}$ after $\boldsymbol{e}$ and $\boldsymbol{d}$ have been drawn for that particular realization of the experiment).

manifolds of GIs at scale $s = \boldsymbol{d}^\dagger \boldsymbol{e}$:

$$\begin{aligned} \mathcal{M}_{\alpha_0}(\alpha, \lambda) &= \frac{1}{\alpha - \alpha_0} \begin{pmatrix} \alpha\lambda - \alpha_0\gamma & \gamma - \lambda \\ \alpha\alpha_0(\lambda - \gamma) & \alpha\gamma - \alpha_0\lambda \end{pmatrix} \\ \mathcal{M}_{\beta_0}(\beta, \lambda) &= \frac{1}{\beta - \beta_0} \begin{pmatrix} \beta\gamma - \beta_0\lambda & \lambda - \gamma \\ \beta\beta_0(\gamma - \lambda) & \beta\lambda - \alpha_0\gamma \end{pmatrix} \end{aligned} \tag{A.42}$$

In the following, we refer respectively to the union and the intersection of those manifolds as $\mathcal{U}$ and $\mathcal{I}$. If we consider any GI $\boldsymbol{M}$ in $\mathcal{U} \setminus \mathcal{I}$, numerical experiments show that the Hessian has exactly two null and two strictly positive eigenvalues; the two null directions, which give us the linearized center space E_c around $\boldsymbol{M}$ in which convergence is at most as a power law, correspond to the local tangent of the manifold of minima and are therefore not relevant: convergence of the loss is exponential.

On the other hand, if $\boldsymbol{M} \in \mathcal{I}$, the Hessian exhibits three null eigenvalues (because the manifolds of minima intersect non-tangentially), so that the center space E_c is now of dimension 3. Since the GIs are only two $2D$ planes, there exists an invariant manifold along which convergence is not exponential. Denoting as x the coordinate along that slow direction, the Center Manifold Theorem ensures that the evolution of x is given by:

$$\dot{x} = g(x), \tag{A.43}$$

where g is a polynomial of order at least 2 with neither constant nor first order term.

Assuming that the order two term is non-zero, we get that locally, for x close to 0, $\dot{x} = ax^2$. Integrating over time, we get that x evolves as τ^{-1}. We then look at the value of the loss when ω is equal to a generalizing minimum $\boldsymbol{M}$ plus a small perturbation $\boldsymbol{X}$ of

order x:

$$\begin{aligned}\mathcal{L} &= \sum_{q,p=1}^{T} \chi_{qp}(\mu_q - s\gamma^q)(\mu_p - s\gamma^p) \\ &= \sum_{q,p=1}^{T} \chi_{qp}\left[\sqrt{\boldsymbol{\Sigma}}\,(\boldsymbol{M}+\boldsymbol{X})^q\,\sqrt{\boldsymbol{\Sigma}})_{12} - s\gamma^q\right]\left[\sqrt{\boldsymbol{\Sigma}}\,(\boldsymbol{M}+\boldsymbol{X})^p\,\sqrt{\boldsymbol{\Sigma}})_{12} - s\gamma^p\right] \\ &= \sum_{q,p=1}^{T} \chi_{qp}\left[(\sqrt{\boldsymbol{\Sigma}}\,\boldsymbol{M}^q\,\sqrt{\boldsymbol{\Sigma}})_{12} - s\gamma^q + \mathcal{O}(x)\right]\left[(\sqrt{\boldsymbol{\Sigma}}\,\boldsymbol{M}^p\,\sqrt{\boldsymbol{\Sigma}})_{12} - s\gamma^p + \mathcal{O}(x)\right] \\ &= \mathcal{O}(x^2)\end{aligned} \tag{A.44}$$

Therefore, if the quadratic term of g is non-zero, x scales as τ^{-1} and the loss as τ^{-2} when τ is large, as is observed experimentally. It should be noted that this result does not depend on the value of T nor on the choice of $\boldsymbol{e}$ and $\boldsymbol{d}$, as observed experimentally too.

Therefore, algebraic convergence is observed only when very strict conditions are met:

- the gradient descent starts from a very specific subspace, the pre-image of the intersection, which we will refer to as a "slow manifold". It is of lower dimension than the initial space of weight-matrices, so that random initial conditions will almost never satisfy this criterion.
- the system always remains on the trajectory of GD. In particular, this means that η and the noise on the computed updates need to be small enough that we don't accidentally leave the slow manifold, which would then lead to exponential convergence.

A.9 Single-channel ReLU proxy loss gradients and Hessian

We have shown in Section 7.4.2 that the two following pairs of conditions are enough to guarantee perfect integration of arbitrary signals:

$$\begin{cases} \boldsymbol{d}^\dagger \lfloor \pm \boldsymbol{W}\boldsymbol{e} \rfloor_+ &= \pm s\gamma \\ \boldsymbol{W} \lfloor \pm \boldsymbol{W}\boldsymbol{e} \rfloor_+ &= \pm\gamma \boldsymbol{W}\boldsymbol{e} \end{cases} \tag{A.45}$$

We define the **proxy** loss as the sum of four terms corresponding to the residuals of those equalities:

$$\mathcal{L}^{proxy} = \mathcal{L}^1_+ + \mathcal{L}^1_- + \mathcal{L}^2_+ + \mathcal{L}^2_- \tag{A.46}$$

where $\mathcal{L}^1_\pm = (\boldsymbol{d}^\dagger \lfloor \pm \boldsymbol{W}\boldsymbol{e} \rfloor_+ - \pm s\gamma)^2$ and $\mathcal{L}^2_\pm = |\boldsymbol{W} \lfloor \pm \boldsymbol{W}\boldsymbol{e} \rfloor_+ - \pm\gamma \boldsymbol{W}\boldsymbol{e}|^2$.

The gradients of these quantities are computed as :

$$\begin{cases} \dfrac{\partial \mathcal{L}^1_\pm}{\partial \boldsymbol{W}_{ij}} = & 2(\boldsymbol{d}^\dagger \lfloor \pm \boldsymbol{W}\boldsymbol{e} \rfloor_+ - \pm s\gamma)(\pm \boldsymbol{d}_i \boldsymbol{H}(\pm \boldsymbol{W}\boldsymbol{e})_i \boldsymbol{e}_j) \\ \dfrac{\partial \mathcal{L}^2_\pm}{\partial \boldsymbol{W}_{ij}} = & 2(\boldsymbol{W} \lfloor \pm \boldsymbol{W}\boldsymbol{e} \rfloor_+ - \pm\gamma \boldsymbol{W}\boldsymbol{e})|_i (\lfloor \pm \boldsymbol{W}\boldsymbol{e} \rfloor_+ - \pm s\gamma)|_j \\ & \pm 2 e_j \sum_a (\boldsymbol{W} \lfloor \pm \boldsymbol{W}\boldsymbol{e} \rfloor_+ - \pm\gamma \boldsymbol{W}\boldsymbol{e})|_a \boldsymbol{W}_{ai} \boldsymbol{H}(\pm \boldsymbol{W}\boldsymbol{e})|_i \end{cases} \tag{A.47}$$

where $\boldsymbol{H}$ is the elementwise Heaviside function, that takes a vector as input and returns a vector whose k-th component is 0 if the k-th component of the input was strictly negative, 1 if it was strictly positive, and .5 if it is exactly 0.

The Hessian of the $\mathcal{L}^1$ terms is readily computed as :

$$\begin{aligned}\frac{\partial \mathcal{L}^1_\pm}{\partial \boldsymbol{W}_{ij}\partial \boldsymbol{W}_{kl}} =& 2\boldsymbol{d}_i \boldsymbol{H}(\pm \boldsymbol{W}\boldsymbol{e})|_i \boldsymbol{e}_j \boldsymbol{d}_k \boldsymbol{H}(\pm \boldsymbol{W}\boldsymbol{e})|_k \boldsymbol{e}_l) \\ &+ 2(\boldsymbol{d}^\dagger \lfloor \pm \boldsymbol{W}\boldsymbol{e}\rfloor_+ - \pm s\gamma)\delta_{ik} d_i e_j e_l \boldsymbol{\delta}(\pm \boldsymbol{W}\boldsymbol{e})|_i\end{aligned} \tag{A.48}$$

where δ is the discrete Dirac distribution δ_{ik} which is one if $i = k$ and 0 otherwise, and $\boldsymbol{\delta}$ the componentwise Dirac distribution such that $\boldsymbol{\delta}(\boldsymbol{v})$ is a vector of same shape as $\boldsymbol{v}$ whose components are 1 if the corresponding component of $\boldsymbol{v}$ is 0, and 0 otherwise. This part of the Hessian is indeed symmetric by exchange of ij and kl because of the δ_{ik} in the second term.

The Hessian of the $\mathcal{L}^2$ terms is more complicated, but can be found to be :

$$\begin{aligned}\frac{1}{2}\frac{\partial \mathcal{L}^2_\pm}{\partial \boldsymbol{W}_{ij}\partial \boldsymbol{W}_{kl}} =& \sum_a \boldsymbol{W}_{ai}\boldsymbol{W}_{ak} e_j e_l \boldsymbol{H}(\pm \boldsymbol{W}\boldsymbol{e})_i \boldsymbol{H}(\pm \boldsymbol{W}\boldsymbol{e})_k \\ &+ \sum_a (\boldsymbol{W}\lfloor \pm \boldsymbol{W}\boldsymbol{e}\rfloor_+ - \pm\gamma \boldsymbol{W}\boldsymbol{e})|_a \boldsymbol{W}_{ai}\delta_{ik}\boldsymbol{\delta}(\pm \boldsymbol{W}\boldsymbol{e})|_i e_j e_l \\ &+ \delta_{ik}(\lfloor \pm \boldsymbol{W}\boldsymbol{e}\rfloor_+ - \pm\gamma \boldsymbol{e})|_j \ (\lfloor \pm \boldsymbol{W}\boldsymbol{e}\rfloor_+ - \pm\gamma \boldsymbol{e})|_l \\ &+ \pm[\delta_{jk}(\boldsymbol{W}\lfloor \pm \boldsymbol{W}\boldsymbol{e}\rfloor_+ - \pm\gamma \boldsymbol{W}\boldsymbol{e})|_i e_l \boldsymbol{H}(\pm \boldsymbol{W}\boldsymbol{e}) + ij \Leftrightarrow kl] \\ &+ \pm[(\lfloor \pm \boldsymbol{W}\boldsymbol{e}\rfloor_+ - \pm\gamma \boldsymbol{e})|_j \boldsymbol{W}_{ik} e_l \boldsymbol{H}(\pm \boldsymbol{W}\boldsymbol{e})|_k + ij \Leftrightarrow kl]\end{aligned} \tag{A.49}$$

Combining those four terms, we get the Hessian of our full proxy loss:

$$\begin{aligned}\frac{1}{2}\frac{\partial \mathcal{L}^{proxy}_\pm}{\partial \boldsymbol{W}_{ij}\partial \boldsymbol{W}_{kl}} =& d_i e_j d_k e_l [\boldsymbol{H}(\boldsymbol{W}\boldsymbol{e})|_i \boldsymbol{H}(\boldsymbol{W}\boldsymbol{e})|_k + \boldsymbol{H}(-\boldsymbol{W}\boldsymbol{e})|_i \boldsymbol{H}(-\boldsymbol{W}\boldsymbol{e})|_k] \\ &+ e_j e_l \sum_a \boldsymbol{W}_{ai}\boldsymbol{W}_{ak}[\boldsymbol{H}(\boldsymbol{W}\boldsymbol{e})|_i \boldsymbol{H}(\boldsymbol{W}\boldsymbol{e})|_k + \boldsymbol{H}(-\boldsymbol{W}\boldsymbol{e})|_i \boldsymbol{H}(-\boldsymbol{W}\boldsymbol{e})|_k] \\ &+ \delta_{ik}\left[(\lfloor \boldsymbol{W}\boldsymbol{e}\rfloor_+ - \gamma \boldsymbol{e})|_j (\lfloor \boldsymbol{W}\boldsymbol{e}\rfloor_+ - \gamma \boldsymbol{e})|_l\right] \\ &+ \delta_{ik}\left[(\lfloor -\boldsymbol{W}\boldsymbol{e}\rfloor_+ + \gamma \boldsymbol{e})|_j (\lfloor -\boldsymbol{W}\boldsymbol{e}\rfloor_+ + \gamma \boldsymbol{e})|_j\right] \\ &+ \left[\delta_{jk}(\boldsymbol{W}^2\boldsymbol{e} - 2\gamma \boldsymbol{W}\boldsymbol{e})|_i e_l + ij \Leftrightarrow kl\right] \\ &+ [\boldsymbol{W}_{ik}(\boldsymbol{W}\boldsymbol{e} - 2\gamma \boldsymbol{e})|_j e_l + ij \Leftrightarrow kl] \\ &+ \delta_{ik} d_i e_j e_l \boldsymbol{d}^\dagger |\boldsymbol{W}\boldsymbol{e}| \boldsymbol{\delta}(\boldsymbol{W}\boldsymbol{e})|i \\ &+ \delta_{ik} e_j e_l \boldsymbol{d}^\dagger |\boldsymbol{W}\boldsymbol{e}| \sum_a \boldsymbol{W}_{ai}(\boldsymbol{W}|\boldsymbol{W}\boldsymbol{e}|)|_a \boldsymbol{\delta}(\boldsymbol{W}\boldsymbol{e})|i\end{aligned} \tag{A.50}$$

Contrary to the linear null initialization case where the Hessian was a 4×4–matrix, we have no way to a priori reduce the number of degrees of freedom, and $\mathcal{H}$ is a $n^2 \times n^2$–matrix. We are therefore restricted to a very low number of neurons (around 50 in our case) for the diagonalization to remain computationally tractable. Another major obstacle is that we do not have an analytical expression of the GI manifolds at which we want to evaluate the Hessian. We adopted the following methodology: first, we train networks on the proxy loss using Gradient Descent at low learning rate and wait for convergence; we evaluate the largest eigenvalue λ_+ of $\mathcal{H}$ at the obtained weight-matrix, but do not compute the lowest ones as they are both prone to numerical errors, and not necessarily positive as some small negative eigenvalues will exist when we are only close to a GI; we compute an "effective" lowest eigenvalue by fitting the decay of the loss during GD at learning rate

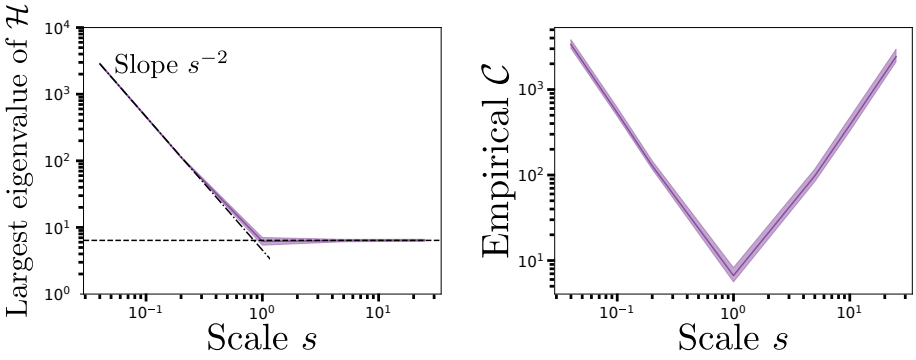


Figure A.4: Experimentally determined values of the highest eigenvalue of the Hessian around the GI manifold, determining the optimal stable learning rate for GD, and of the empirical convergence time as a function of the scale. Numerical experiments carried out with $n = 50$ neurons, independent encoder and decoder of norm 1.

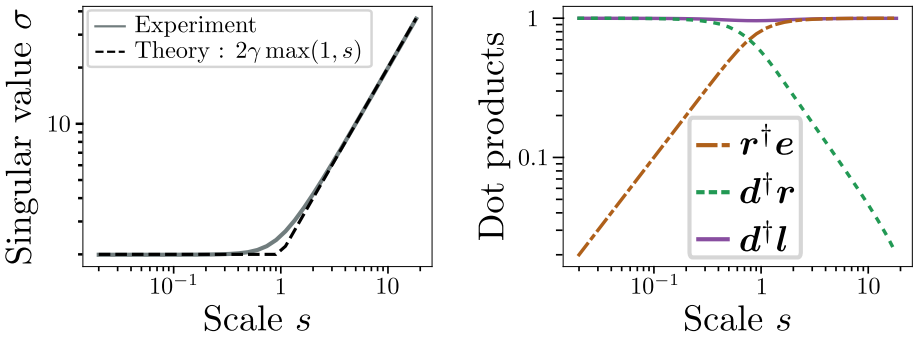


Figure A.5: Behavior of the singular value σ and dot products of the singular vectors $\boldsymbol{l}, \boldsymbol{r}$ with $\boldsymbol{e}, \boldsymbol{d}$ as functions of s. The figure was obtained with $n = 1000$, independently drawn encoder and decoder, and aggregated across 6 realizations of GD on the proxy loss. Error bars are not reported as they are not distinguishable from the line width.

$\eta < 1/\lambda_+$, and deduce the corresponding minimum convergence time. The results of these numerical simulations are presented in Figure A.4. We performed tests on larger networks to verify if the inferred maximum stable learning rate remained valid, as well as the order of magnitude of the convergence time, yielding the expected results.

A.10 Analysis of rank-1 ReLU generalizing integrators

Training of the ReLU RNNs, either on real data or on the proxy loss (7.31), leads to GIs exhibiting one dominant singular value. As in the linear case, we write $\boldsymbol{W} = \sigma \boldsymbol{l}\boldsymbol{r}^\dagger$; with no loss of generality, we can impose $\boldsymbol{r}^\dagger \boldsymbol{e} > 0$ by multiplying $\boldsymbol{r}$ and $\boldsymbol{l}$ by -1. Conditions (7.30) become

$$\begin{cases} \sigma\, \boldsymbol{r}^\dagger \boldsymbol{l}_\pm & = \pm\gamma \\ \sigma (\boldsymbol{r}^\dagger \boldsymbol{e})(\boldsymbol{d}^\dagger \boldsymbol{l}_\pm) & = \pm s\gamma \end{cases} \quad \text{where } \boldsymbol{l}_\pm = \lfloor \pm \boldsymbol{l} \rfloor_+. \tag{A.51}$$

Using Cauchy-Schwarz inequality, and denoting as $\mathbf{1}_\pm$ the vector whose component i is equal to 1 if $\boldsymbol{l}_i$ is of the corresponding sign and 0 otherwise, we have:

$$|\boldsymbol{d}^\dagger \boldsymbol{l}_\pm| = |(\boldsymbol{d}\mathbf{1}_\pm)^\dagger (\pm \boldsymbol{l}\mathbf{1}_\pm)| \leq |\boldsymbol{d}\mathbf{1}_\pm||\boldsymbol{l}\mathbf{1}_\pm|. \tag{A.52}$$

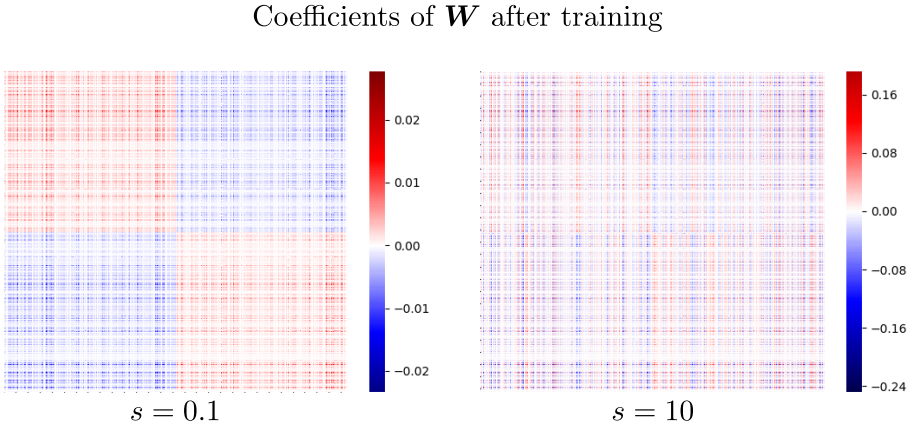


Figure A.6: Weight-matrix $\boldsymbol{W}$ of a single-channel ReLU network, visualized as a discrete heatmap. Neurons were reordered so that the indices of the "+" population are from 0 to $n/2$, while the "-" population goes from $n/2$ to n. When $s = 0.1$, the sign of W_{ij} is fully determined by whether i and j are in the same cluster; when $s = 10$, this observation is no longer true. We also note that, as expected, the coefficients of $\boldsymbol{W}$ are larger when s increases as the norm of $\boldsymbol{W}$ scales as $\max(1, s)$. This figure was obtained for independent encoder and decoder of scale 1, $n = 1000$.

Assuming half of the components of $\boldsymbol{l}$ are positive and half are negative, as confirmed by numerical studies, and given that $|\boldsymbol{d}| = |\boldsymbol{l}| = 1$, both norms on the right-hand side are equal to $2^{-1/2}$ in the large n limit, yielding $|\boldsymbol{d}^\dagger \boldsymbol{l}_\pm| \leq 1/2$. Since $0 \leq \boldsymbol{r}^\dagger \boldsymbol{e} \leq 1$, we conclude that $\sigma \geq 2s\gamma$. Similarly, we have that $|\boldsymbol{r}^\dagger \boldsymbol{l}_\pm| \leq 1/2$, implying $\sigma \geq 2\gamma$. These conditions can then be summarized into $\sigma \geq 2\gamma \max(s, 1)$.

Experimentally, we find that this lower bound is closely followed when s is either large or small. Since $\boldsymbol{W}$ is of rank 1, its Frobenius norm is equal to σ, and we argue that the saturation of this lower-bound on σ is a manifestation of the conjecture of (Arora et al., 2019) that gradient descent implicitly favors solutions with small matrix norm. Therefore, we have for any scale s significantly different from 1:

$$\sigma = 2\gamma \max(s, 1). \tag{A.53}$$

Numerical experiments show that, for a wide range of scales, $\boldsymbol{l}$ and $\boldsymbol{d}$ are almost equal. Hence, $\boldsymbol{d}^\dagger \boldsymbol{l}_\pm = \pm|\boldsymbol{d}\lfloor \pm \boldsymbol{d}\rfloor_+|^2 \simeq \pm 1/2$, entailing $\boldsymbol{r}^\dagger \boldsymbol{e} \simeq \min(s, 1)$. For $s \ll 1$, $\boldsymbol{r}$ is almost aligned with $\boldsymbol{d}$, while for $s > 1$ we have $\boldsymbol{r} \simeq \boldsymbol{e}$ (This statement holds for uncorrelated encoder and decoder). Our theoretical predictions are in very good agreement with numerical experiments, as shown in Figure A.5. Notice that the change of direction of $\boldsymbol{r}$ with s has consequences on the signs of the couplings: W_{ij} is positive for pairs of neurons within the "+" and "-" populations and negative in between at small s, but is essentially random at large s, see Figure A.6.

A.11 Transfer learning: context-dependent selectivity

In order to illustrate the versatility of the current-linear representations described in the main text, we implement a simple example of transfer learning to context-dependent selectivity, inspired by (Mante et al., 2013). The idea is the following: a pretrained 3–channels integrator is used to integrate 3 time-series x_0, x_1, x_2 (respectively, the motion evidence, color evidence and contextual cue in the experiment described by (Mante et al., 2013))

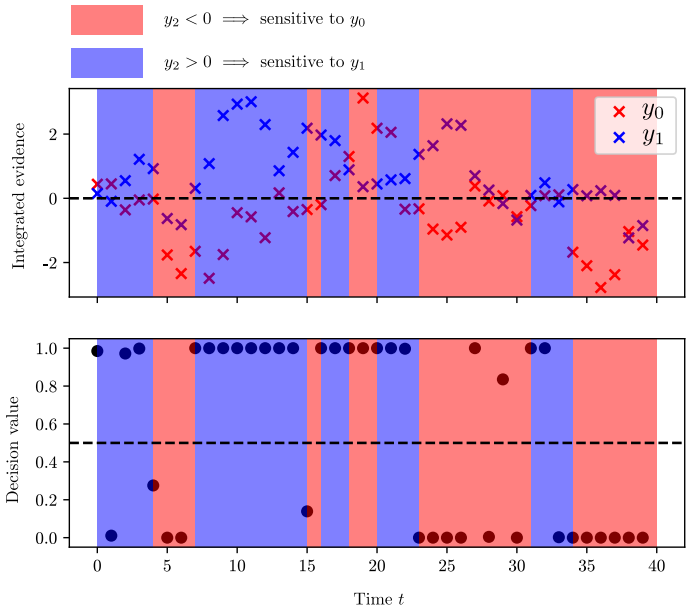


Figure A.7: Output of an online context-dependent classifier. The task is the following: the network receives $D = 3$ input channels; when the integral y_2 is negative, the network must output 0 if $y_0 < 0$ and 1 if $y_0 > 0$; when the integral y_2 is positive, the network must output 0 if $y_1 < 0$, and 1 if $y_1 > 0$. This result is obtained by training a sigmoidal decoding layer on the internal states of a fixed sigmoidal network pretrained through batch–SGD.

into their decaying integrals y_0, y_1, y_2, potentially with different decay constants. The cue integral y_2 is used to determine to which integral, y_0 or y_1, the network must be sensitive: when y_2 is negative, the network must output 0 if $y_0 < 0$ and 1 if $y_0 > 0$; when the integral y_2 is positive, the network must output 0 if $y_1 < 0$, and 1 if $y_1 > 0$.

To train this network, we first train the 3–channels integrator using any of the methods described in the main text. We then use it as a fixed input transformer, mapping a 3–dimensional time-series to a n–dimensional one (the state $\boldsymbol{h}_t$ at any time-step). For each time-step, the value of the expected output is determined using the aforementioned rules on $\boldsymbol{y}$, and the classification output is obtained as $out_t = (1 + e^{-50(\boldsymbol{u}^\dagger \boldsymbol{h}_t - 0.1)})^{-1}$. The trainable parameter of this new decoding layer is the vector $\boldsymbol{u}$. It is easy to learn the value of $\boldsymbol{u}$ through batch–SGD using the supervised learning procedure described here, and the resulting networks behave as shown in Figure A.7.

Appendix B

Cognitive maps for Path Integration

B.1 The Continuous GridWorld environment

All experiments presented in this article were performed using the GridWorld environment class, defined in the environment.py file. While this environment is neither particularly efficient to run, as it is coded in pure python, nor very expressive, it still allows for a wide variety of interesting situations and is performant enough to not be an unreasonable bottleneck in most situations. It also follows the basic specifications of the OpenAi Gym framework, which makes it easy to extend it and test basic Reinforcement Learning tasks such as goal-driven navigation.

While we do not provide level editors or tools to procedurally generate new layouts, they can be added by hand in the environment register, under a new key corresponding to the map name, as a dictionary containing the following attributes:

- the number of rooms in the environment;
- a list containing the position of the room centers in the surrounding environment (the environment can be rescaled as a whole when instantiated, so it is easier to assume all rooms to be of size 1×1);[1]
- a list of room exits, such that the i-th component is a list containing all exits from room i. An exit is defined as a room edge (either vertical or horizontal) that is connected to another room edge, and the link between coordinates in the two rooms is established by giving the coordinates of the same physical point in the two rooms (the center of the common room edge);
- a list of possible item layouts, detailing what items are visible (and at which position from the room center) within each room, allowing us to simulate visual occlusion, or even special effects (such as switching the light off when the agent is in a particular room). All items are point-like light emitters, and only differ through their color.

[1] While this environment is designed to be used for 2–dimensional spatial navigation, we consider the room centers to have three components; the first two are the (x, y) coordinates which can be modified by our two-dimensional actions, while the third one can be used to lift ambiguity between two rooms that would be located at the same position but would differ in another way meaningful for the environment (for example, to change the environment after the agent visits a particular room). This functionality is not used in the experiments presented in this study.

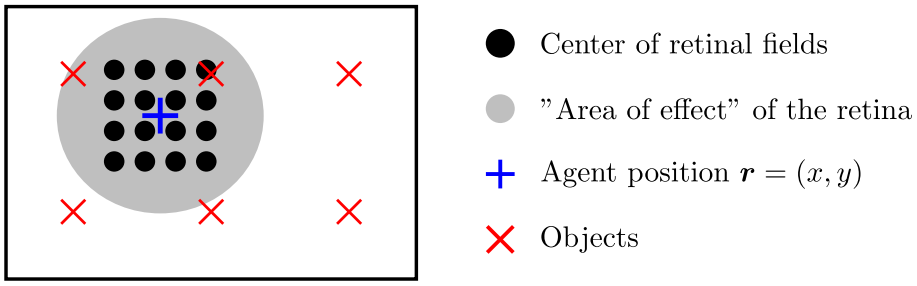


Figure B.1: Example of retina: a regular square lattice of Difference of Gaussians fields. We describe as the "area of effect" of the retina the zone in which an object would contribute to the activity vector of the retina as a whole above some arbitrary threshold, *e.g.* 10^{-2}.

Given the connection graph of the rooms and the item layout, the GridWorld class can be used to generate trajectories from sequences of actions, as well as to generate a human-interpretable rendering of the arrangement of rooms and items which can be used as a basis for more involved plots (most notably, trajectories and value of neuron activity as a function of position in the environment).

The inputs to the network, which we refer to simply as images, are obtained from a list of rooms and positions within these rooms, by using a retina of the type described in Appendix B.2.

We provide several environments and layouts, some of which were used only for preliminary tests but which we retain for the sake of completeness.

B.2 Retina

Individual retinal fields We begin by introducing the Difference of Gaussians retinal neurons (Dayan and Abbott, 2001): their receptive fields have a **center** $\boldsymbol{c}$, two widths (σ_+, σ_-), and their activity g when a single visual cue is present at distance $\delta\boldsymbol{r}$ from the center is a difference of Gaussians:

$$g(\boldsymbol{r} = \boldsymbol{c} + \delta\boldsymbol{r}) = \frac{A}{\sqrt{2\pi\sigma_+}} \exp^{-\frac{\delta\boldsymbol{r}^2}{2\sigma_+^2}} - \frac{B}{\sqrt{2\pi\sigma_-}} \exp^{-\frac{\delta\boldsymbol{r}^2}{2\sigma_-^2}} \tag{B.1}$$

When two or more visual cues are presented in the image, the activation of the neurons will be the sum of the activations for each individual object.

Retinal array We will consider as **retina** a square array of these retinal fields (see Figure B.1), all with $A = B$ for simplicity. The value of A is determined so that, on average over the position of a single object within a square room of width 1, the average norm of the retina activity vector is equal to 1. For experiments, we will use an array of $64^2 = 4096$ cells; for the values of the two widths, we choose $(\sigma_+ = 0.4, \sigma_- = 0.5)$.

Color retina In order to be able to differentiate between two objects, we associate to each a "color", i.e. a vector in $[0, 1]^3$. This color is perceived by the retina in the following way: there are three "copies" of our retina, one for each color "channel"; the object activates each "channel" proportionately to the object's value in that color. This makes it so that each position $\boldsymbol{r}$ in the environments corresponds to one image $\boldsymbol{i}$ of size $(3, 64, 64)$ for the 64×64–retinas that we use.

Optimal linear reconstruction of an arbitrary function of position Let us consider the family of functions $\{(g_1(\boldsymbol{r}), g_2(\boldsymbol{r}), \dots g_n(\boldsymbol{r})\}$, where $g_i(\boldsymbol{r})$ describes the activation of neuron i when a cue is located at position $\boldsymbol{r}$. A linear model of an arbitrary function f (of the cue position) from the state of our retina can be written as a linear combination of those functions:

$$\hat{f}(\boldsymbol{r}) = \sum_i \alpha_{f,i} g_i(\boldsymbol{r}) \tag{B.2}$$

The associated reconstruction error on a domain $\mathcal{D} \subset \mathbb{R}^2$ can then be written:

$$\begin{aligned}
\mathcal{E}(\boldsymbol{\alpha}_f) &= \int_{\mathcal{D}} \left[\sum_i \alpha_{f,i} g_i(\boldsymbol{r}) - f(\boldsymbol{r}) \right]^2 \\
&= \int_{\mathcal{D}} \left[\sum_{i,j} \alpha_{f,i} \alpha_{f,j} g_i(\boldsymbol{r}) g_j(\boldsymbol{r}) - 2 f(\boldsymbol{r}) \sum_i \alpha_{f,i} g_i(\boldsymbol{r}) + f(\boldsymbol{r})^2 \right] \\
&= \sum_{i,j} \alpha_{f,i} \alpha_{f,j} \int_{\mathcal{D}} g_i(\boldsymbol{r}) g_j(\boldsymbol{r}) - 2 \sum_i \alpha_{f,i} \int_{\mathcal{D}} f(\boldsymbol{r}) g_i(\boldsymbol{r}) + \int_{\mathcal{D}} f(\boldsymbol{r})^2 \\
&:= \sum_{i,j} \alpha_{f,i} \alpha_{f,j} I_{ij} - 2 \sum_i \alpha_{f,i} h_{f,i} + C
\end{aligned} \tag{B.3}$$

The minimum of this loss with respect to the parameters $\boldsymbol{\alpha}_f$ satisfy:

$$\forall i, \frac{\partial \mathcal{E}}{\partial \alpha_{f,i}} = 2 \left[\sum_j I_{ij} \alpha_{f,j} - h_{f,i} \right] = 0 \tag{B.4}$$

and therefore the optimal linear decoder is obtained as $\boldsymbol{\alpha}_f = \boldsymbol{I}_f^{-1} \boldsymbol{h}_f$.

Numerical approximation of the integrals over $\mathcal{D}$ are easily obtained by averaging over measures on a grid lattice, up to some numerical precision limitations. This decoding is not particularly fast, since determining $\boldsymbol{I}$ is computationally expensive.

Expected optimal performance When the environment consists of a single room, containing a single object, we can easily reconstruct the relative position of the object with respect to the center of the retina using either the direct linear solving of the previous paragraph or gradient descent on a more parametrized structure, *e.g.* a dense or convolutional neural network. This experiment gives us an idea of the maximum performance that can be expected from any inverse model based on our retina.

When using a 64^2 array spanning $[-0.5, 0.5]$ and reconstructing the position of an object placed arbitrarily in $[-1, 1]$, the Root Mean Square error for the optimal linear reconstruction is around 10^{-2}, with clear geometric patterns in the errors, while the 3-Layers ReLU network achieves a performance closer to 10^{-3} by seemingly "smoothing" the aforementioned error patterns. These results are presented in Figure B.2.

B.3 Network architectures and training hyperparameters

B.3.1 Architectures

The exact architectures used in our implementations are as follows:

- the convolutional networks $\mathcal{V}$ used to obtain the visual representations are:

    1. Input image with 3 channels, size 64×64 (from the retina)

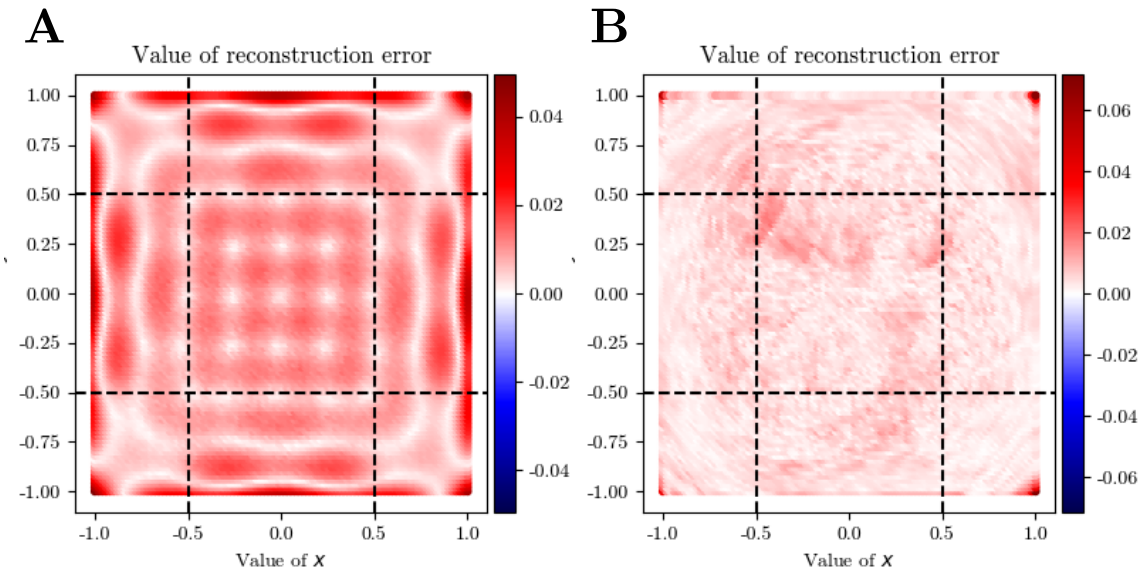


Figure B.2: Reconstruction error on a grid with 100 subdivisions on x and y as a function of position, represented as a heatmap. On the left, the reconstruction error of the optimal linear decoder shows clear geometric patterns related to the geometry of the underlying array of cells. On the right, the reconstruction error for a "deep" 3-layer ReLU network. While the highest observed error is higher for this deep network, the error on all points except the corners is much lower than for the linear network. Additionally, the geometric patterns are not observed in that case.

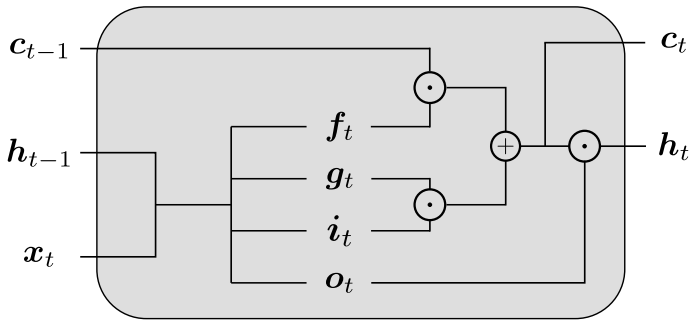


Figure B.3: Computation diagram for a LSTM cell, as implemented in Paszke et al. (2019).

2. Convolution with 16 filters, kernel size of 5, stride of 3, padding of 2
3. Convolution with 32 filters, kernel size of 5, stride of 5, padding of 2
4. Flatten layer
5. Dense layer with 512 outputs
6. Dense layer with 512 outputs

- the action encoding networks $\mathcal{P}$ are:

1. Input layer of size 2
2. ReLU layer with 256 outputs
3. Linear layer with 512 outputs

- the direct model networks $\mathcal{D}$ are:

1. Input layer of size 1024 (concatenation of representation and action encoding)
2. ReLU layer with 512 outputs
3. Linear layer with 512 outputs

- the inverse model networks $\mathcal{D}$ are:

1. Input layer of size 1024 (concatenation of two representations)
2. ReLU layer with 256 outputs
3. Linear layer with 2 outputs

- the gating networks $\mathcal{G}$ are:

1. Input layer of size 512 (visual representation)
2. ReLU layer with 128 outputs
3. ReLU layer with 64 outputs
4. Sigmoid layer with 1 output

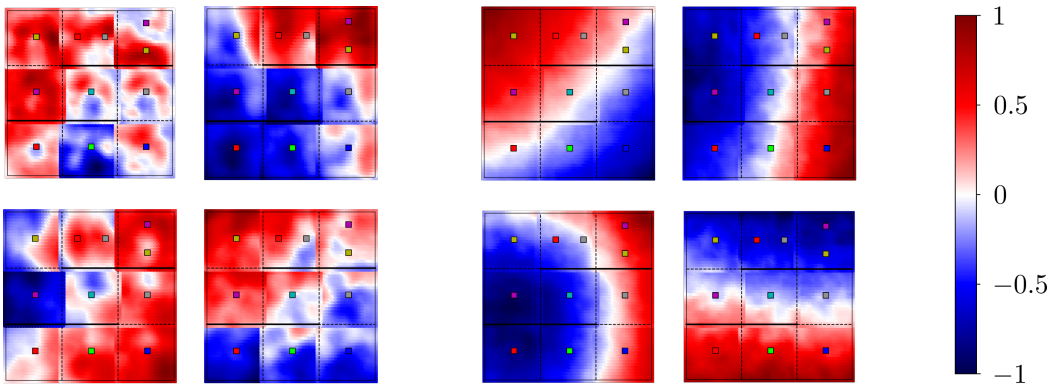
It should be noted that our Recurrent Path Integrator models have a number of parameters of the same order of magnitude as the off-the-shelf implementations of LSTM that we used (see Figure B.3).

B.3.2 Training

In all experiments we present, the PI losses are computed on batches of 32 trajectories of length 40, with actions drawn from a two-dimensional uncorrelated Gaussian of standard deviation 1/2 and starting points chosen randomly at any position in any non-ambiguous room. The direct and inverse losses are computed on batches of 512 transitions, for which starting points and actions are chosen in the same way as for PI.

The relative weights in the total loss are 10 times higher for the direct inverse losses than for the PI loss, as the former are smaller (these hyperparameters have not been optimized)

Training consists of 4000 steps of computing the losses, and performing one step of Adam optimization with uniform learning rates of 10^{-3} except for the forward model which is at 10^{-4} (no hyperparameter optimization was lead on these either). In most cases, losses only evolve marginally after the first hundreds of epochs.



(a) Without forward model (b) With forward model

Figure B.4: Comparison between representations obtained after training the Direct-Inverse Model module in our environment. Each panel represents the normalized activation of a single neuron in the visual representation $\mathcal{V}(\boldsymbol{s})$, obtained at the end of the visual processing module, represented as a function of position within the environment through a color code presented on the right scale. The activities are of the same order of magnitude in all cases. When optimizing only the inverse loss (eq. 8.5), see panel (a), the representations can be spatially irregular; the introduction of the direct loss (eq. 8.4) smooths out the activities, see panel (b). This effect is quantified in Table B.1.

We emphasize that since the environment, starting positions and actions are both continuous and random, no trajectory is ever seen twice by the network, so overfitting is not a concern (rather, we hope that the network meaningfully interpolates between what it has previously seen). However, we train the networks on a single environment, and the question of the capacity–resolution tradeoff (how the precision of PI is modified when several environments are learned at the same time) remains unaddressed in the present study and a meaningful future direction.

B.4 Interactions between direct and inverse losses

As mentioned in the main text, training the inverse model without the direct one is possible, but the other way around is not, as training in that case converges to a trivial solution where the representation module $\mathcal{V}$ and the direct model $\mathcal{D}$ always output $\mathbf{0}$. When training with both losses, we observe a slight but noticeable smoothing of the representations, as illustrated in Figure B.4 and Table B.1. This improvement in regularity does not however translate into any measurable difference in performance of the inverse model: the representations are made simpler by the direct loss (as expected when adding a regularization term), but they do not carry any more positional information. It should be noted that while "generalization" errors (computed on any couple of images, even if they can not be reached in a single transition and hence were never part of a transition tuple used in training) are large, the difference in position predicted remains qualitatively relevant (just with a lower precision).

Table B.1: Comparison of the inverse model performance and the representation regularity between models trained with or without the direct loss. The addition of the direct loss shifts the distribution of R^2 scores between neuron activities and spatial position towards one, meaning it made some neuron activities closer to linear functions of position, which we argue is a desirable property in order to obtain transferable representations. While this shift is noticeable, it does not come with any appreciable change in inverse model performance. Means and deviations computed across 8 realizations.

	Error (training)	Error (generalization)	R^2 (visual)
Without direct	0.017 ± 0.01	1.6 ± 0.77	0.95 ± 0.089
With direct	0.015 ± 0.0091	1.6 ± 0.75	0.83 ± 0.19

B.5 LSTM variants

In this section, we present our attempts at improving the performance of the "vanilla" LSTM architecture. Beyond basic hyperparameter tuning (no extensive optimization has been led due to prohibitive computational costs), we mostly considered modifications on initialization, and architecture:

- the standard architecture corresponds to simply using a LSTM (see Figure B.3) as the RNN in the computational graph of Figure 8.2. We considered two training schemes with this architecture:
  - the "vanilla" scheme trains this network from scratch on the PI loss. It yields good integration properties, but fails to learn resetting behaviors (results presented in main paper).
  - the "pretrained" scheme initializes the "encoders" (both for the image and the action) using the ones of a Resetting Path Integrator trained with all losses (since they exhibit the cleanest representations). The results are very similar to the "vanilla" scheme.
- the "hybrid" solutions have the computation diagram of Figure B.5, which corresponds to using the LSTM only to update the internal state, replacing the combination of the direct model $\mathcal{D}$ and gating module $\mathcal{G}$. This architecture has the advantage of explicitly using the initial representation as a form of "anchor", which seems in practice to help training converge to resetting behaviors. We considered several training schemes using this architecture:
  - the "default" scheme trains this network from scratch on the PI loss, yields similar result to vanilla LSTM.
  - the "pretrained" initializes the "encoders" (both for the image and the action) using the ones of a Resetting Path Integrator trained with all losses; it reliably achieves resetting, but also has lower precision on short trajectories.
  - the "scratch" scheme does not do any initialization, but adds a "direct" module (same architecture as the ones of our Resetting Path Integrator), defines the direct and inverse losses using it and the $\mathcal{I}$ module that outputs the displacement, and uses those losses as regularization. This scheme often manages to

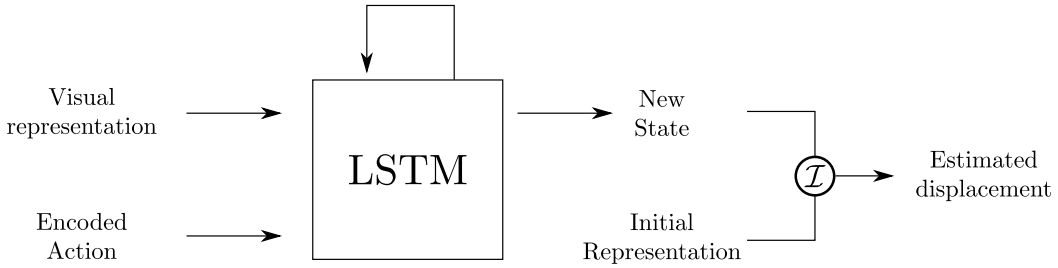


Figure B.5: Computation diagram for the hybrid path integrator structure.

Table B.2: Comparison between the different LSTM variants we considered on the SnakePath environment, in terms of both errors and representation correlation with position, see main text for details. Means and errors computed on 8 realizations.

	Standard	Hybrid		
	Pretrained	Default	Pretrained	Scratch
Error (short)	0.01 ± 0.0064	0.013 ± 0.0086	0.034 ± 0.02	0.022 ± 0.014
Error (long)	0.091 ± 0.088	0.11 ± 0.099	0.043 ± 0.026	0.042 ± 0.03
R^2 (visual)	0.46 ± 0.2	0.63 ± 0.24	0.91 ± 0.14	0.94 ± 0.11
R^2 (PI, absolute)	0.27 ± 0.11	0.29 ± 0.11	0.63 ± 0.19	0.58 ± 0.21
R^2 (PI, relative)	0.69 ± 0.26	0.72 ± 0.24	0.24 ± 0.092	0.34 ± 0.14

find resetting solutions, but requires more care in hyperparameter tuning to converge properly to a resetting solution.

– the "improved" scheme is similar to scratch, but also initializes the encoders. This scheme is included in the main text, and it reliably achieves resetting.

In Table B.2, we present results for the aforementioned architecture that were not included in the main text, Table 8.1. It should be noted that even schemes that yield solutions that exhibit resetting do not necessarily have higher levels of positional tuning, which we argue still makes them less convincing candidates for transferable cognitive maps than our Resetting Path Integrator model.

B.6 Curriculum Learning and catastrophic forgetting

In this section, we consider the case in which the direct-inverse model of the environment is trained first, using transition tuples, without any consideration of Path Integration.

After this initial pretraining, we introduce those weights into the full Resetting Path Integrator network, and consider three different ways of training the PI task:

- **A**: we optimize only the weights of the resetting gate $\mathcal{G}$, and use only the Path Integration Loss.
- **B**: we optimize all weights in the network, including those that were initialized from the pretraining, still using only the Path Integration Loss.
- **C**: we optimize all weights in the network, but do Gradient Descent on a sum of all losses (Path Integration, direct and inverse), as we would in end-to-end training.

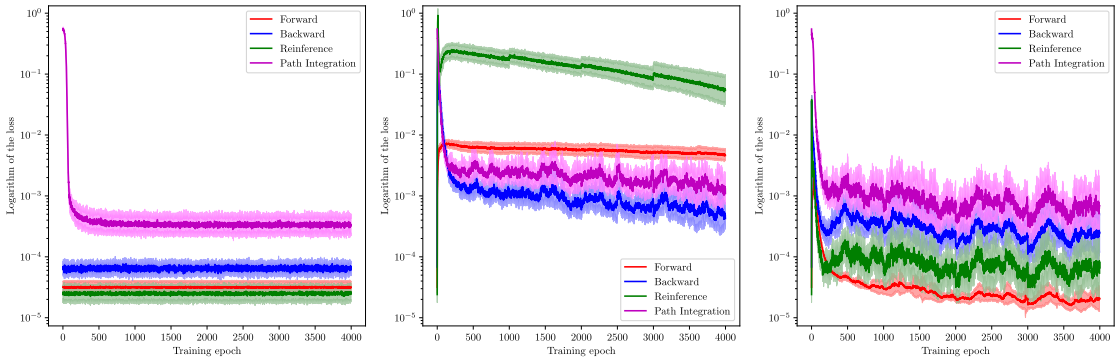


Figure B.6: Evolution of the different losses when training a Resetting Path Integrator, whose visual, direct and inverse modules were pretrained using transition tuples from the environment, in the three different protocols described in the text.

Table B.3: Comparison between our Resetting Path Integrator model and standard LSTM in the DoubleDonut environment. As was the case in the SnakePath environment, models trained without the direct-inverse losses fail to learn how to perform resetting and show lower levels of spatial structure in their representations.

	Resetting Path Integrator		Long Short Term Memory	
	All losses	No model losses	Vanilla	Improved
Error (short)	0.026 ± 0.019	0.035 ± 0.026	0.015 ± 0.0086	0.032 ± 0.019
Error (long)	0.032 ± 0.023	0.46 ± 0.41	0.15 ± 0.14	0.051 ± 0.033
R^2 (visual)	0.97 ± 0.068	0.16 ± 0.12	0.36 ± 0.16	0.91 ± 0.13
R^2 (PI, absolute)	0.96 ± 0.073	0.29 ± 0.063	0.27 ± 0.086	0.68 ± 0.19
R^2 (PI, relative)	0.34 ± 0.058	0.85 ± 0.14	0.74 ± 0.22	0.26 ± 0.083

The results are presented in Figure B.6, and show that while the final level of Path Integration error is close between the different protocols, retraining all parameters of the model on the Path Integration loss only (protocol **B**) produces noticeable deterioration in the quality of the Direct-Inverse model, an example of the catastrophic forgetting phenomenon (Kirkpatrick et al., 2017). Independently of the choice of loss on which Gradient Descent is performed, optimizing on the parameters of the Direct-Inverse model produces much more chaotic evolution of the loss.

B.7 Errors and spatial correlations on the DoubleDonut environment

We present in Table 8.1 the values of errors and spatial correlations measured in networks trained on a second environment layout, slightly different from the one used in the main text, and which we call DoubleDonut. This environment has the general structure of its ambiguous variant (represented in Figure B.11), except that the objects in the left- and right-most rooms of the middle row are different in the case of the non-ambiguous version that we consider here. The conclusions of this study are identical to the ones presented in the main text.

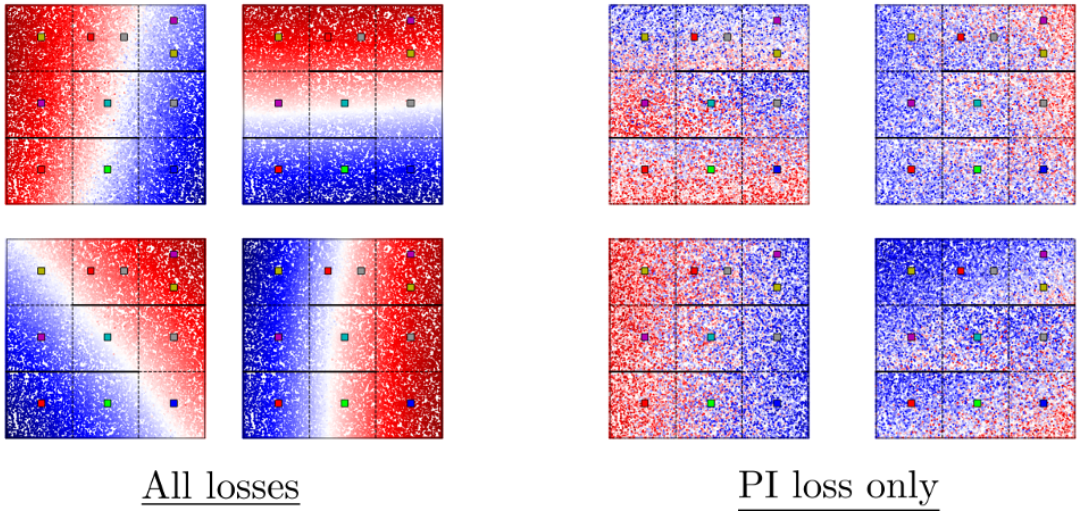


Figure B.7: Activity of four representative neurons in the internal state population of an RPI trained with (left) or without (right) the model losses, as a function of absolute position in the environment. Only the ones trained with those losses (and performing resetting) are close to a linear function of absolute position.

B.8 Representations in absolute and relative coordinates

In order to complement the R^2 values presented in main text Table 8.1, we report the value of neuron activations (in the internal state, observed during Path Integration) as a function of absolute position in the environment (Figure B.7) and as a function of position within the trajectory (Figure B.8) for our Resetting Path Integrator model, trained with or without the direct–inverse losses (and consequently, respectively displaying resetting or not). We find that non-resetting representations are linear functions of displacement, as expected from an integrator network (see Fanthomme and Monasson (2021)), while resetting representations are linear functions of absolute position, which makes them much more relevant as cognitive maps. Interestingly, we note that (for the resetting network on the left of Figure B.8), points that correspond to "extreme" displacements seem more correlated with trajectory coordinate than points with small trajectory coordinates; this is to be expected since extreme displacement points necessarily lie on the edge of our environment (to get a displacement of -3 in the x direction, the agent necessarily started on the right side of the environment and finished on the left side), so that for those points trajectory coordinates and absolute coordinates are correlated.

B.9 Gating strength in a Resetting Path Integrator

While all training conditions we investigated lead to resetting behaviors, the mean value of the gating obtained from images of the environment was observed to vary drastically, from 10^{-1} to 10^{-3}, while the level of Path Integration errors remained mostly unchanged. Our understanding for this phenomenon is the following: the minimum level of achievable error is the same for all training conditions, and related to the limitations of the retinal array detailed in Appendix B.2; therefore, the network can accumulate errors (coming either from imperfect reafference or imperfect integration) for a certain number of time-steps without any noticeable effect. In the limit case where the direct model performs

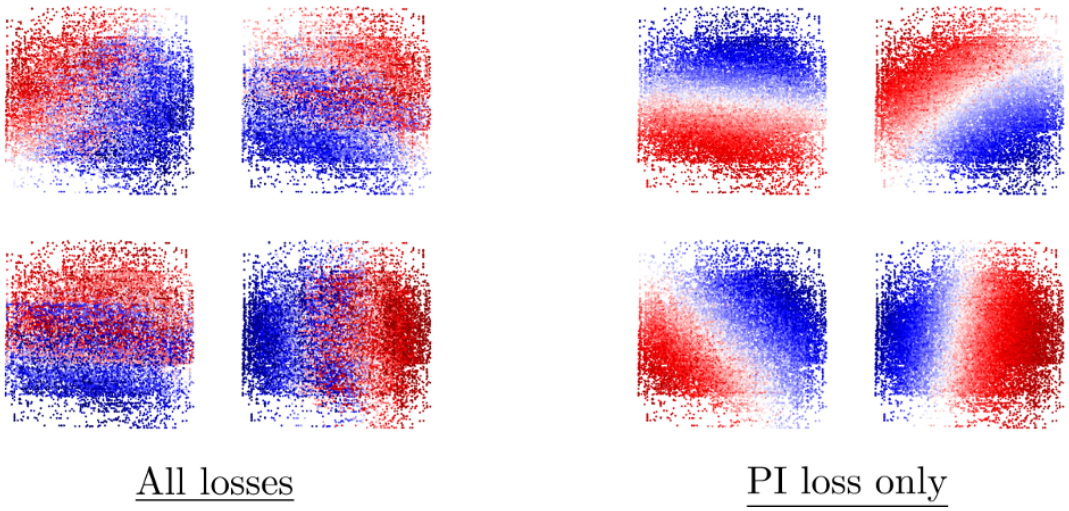


Figure B.8: Activity of four representative neurons in the internal state population of an RPI trained with (left) or without (right) the model losses, as a function of position along the trajectories. Only the ones trained without those losses (and not performing resetting) are close to a linear function of position within the trajectory.

perfectly and reafference is exact, no resetting is ever necessary. A direct way to limit the accuracy of the direct model is to add noise to the reafferent action during training, and our hypothesis is that as this noise increases the value of the resetting gate will get closer to 0, meaning that the resettings will be stronger and "keep less memory" of the state before resetting. This hypothesis is confirmed by Figure B.9**A**. Additionally, we expect that if, at test time, we present the gating module $\mathcal{G}$ with increasingly perturbed representations, the value of the reset gate will increase until no resetting happens ($g = 1$) if the image is completely shuffled.[2] This situation is represented in Figure B.9**B**.

B.10 Internal gates in an LSTM network

Averaged across a large number of trajectories, the values of the input and reset internal gates at each neuron show, to a varying extent, the behavior that was to be expected from the gate names: the reset gate neurons are inhibited when an image is presented (meaning that the previous internal state is suppressed), while the input gate neurons are activated (meaning that the current input contributes more to the internal state update). We present in Figure B.10 a few representative neurons in both populations. Given the high variability that is observed in the reset and input gates, we expect that those two subnetworks contribute not only to the resetting, but also to the computation of the direct model.

B.11 Resetting in the case of Ambiguous DoubleDonut

In this section, we consider the same end-to-end training procedure as in the main article, but apply it to a more complex environment comprised of 16 rooms, two of which (the

[2]We did not present the networks with partial shufflings of the representations at any stage in the training, only unperturbed or fully shuffled.

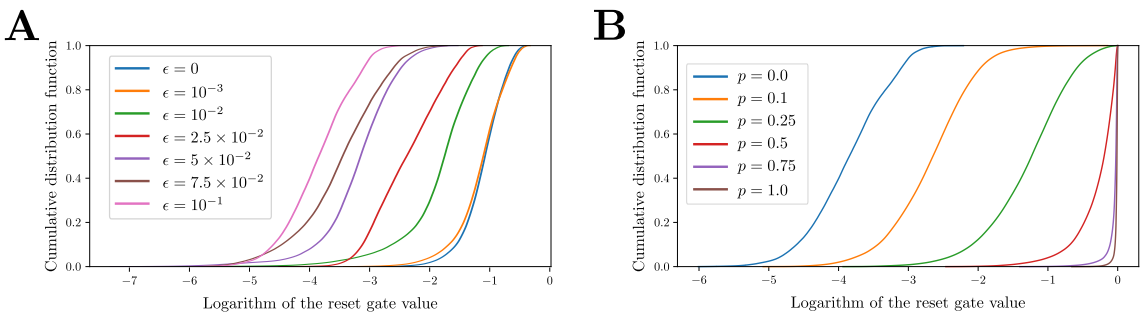


Figure B.9: Cumulative distribution function of the natural logarithm of the reset gate g across the environment in different conditions. **A**: Varying the level of noise ϵ in the reafferent action during training. As expected, high levels of noise favor strong resettings, hence lower values of g. **B**: Varying the level of perturbation p, defined as the fraction of neurons in the representation that were randomly reshuffled, at test time. As representations are increasingly perturbed, they become less similar to ones that come from the environment, and we expect the network to reset its state less strongly as the expected benefit from such a resetting decreases. Both panels present results aggregated across 16 different realizations of the PI training.

left-most and right-most of the middle row) provide the agent with identical visual cues, creating an ambiguity where two different positions in the global environment correspond to the same images. This ambiguity is still such that the inverse model is unambiguously defined (since there are no positions in the environment that could be reached in a single transition from both ambiguous rooms), and the forward model too as long as we choose the start position in any non-ambiguous rooms (because otherwise, the same initial state and the same action could lead to two different new states, one for each room). As shown in Figure B.11, the Resetting Path Integrator models still manage to perform reasonably well despite this ambiguity by creating new representations, distinct from the visual ones, as shown in Figure 8.7, and not performing resetting when the image comes from one of the ambiguous rooms, see Figure B.12.

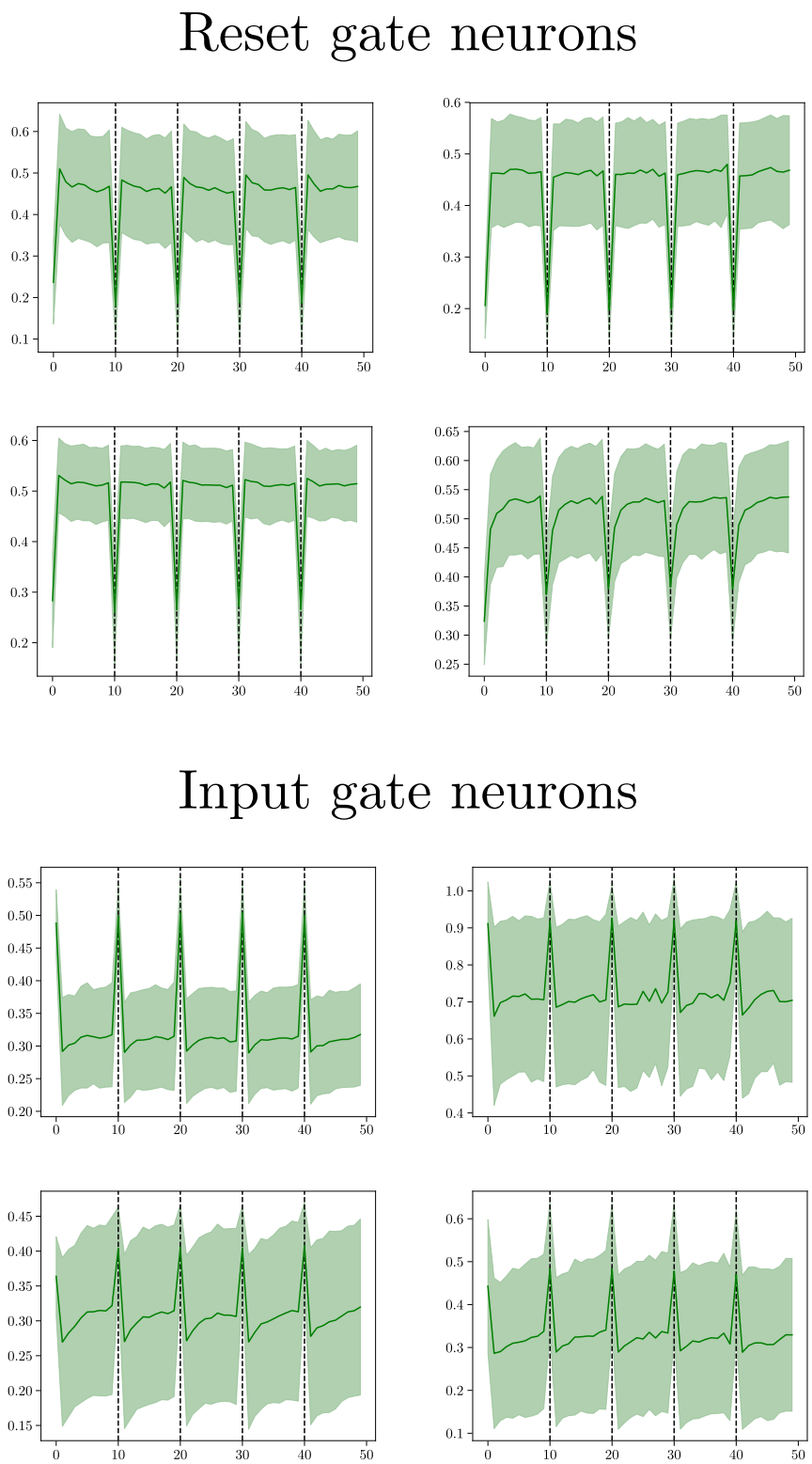


Figure B.10: Activity of four representative neurons in the "reset" and "input" gates, as a function of time, aggregated across 128 trajectories in which images are always presented at the same time-steps.

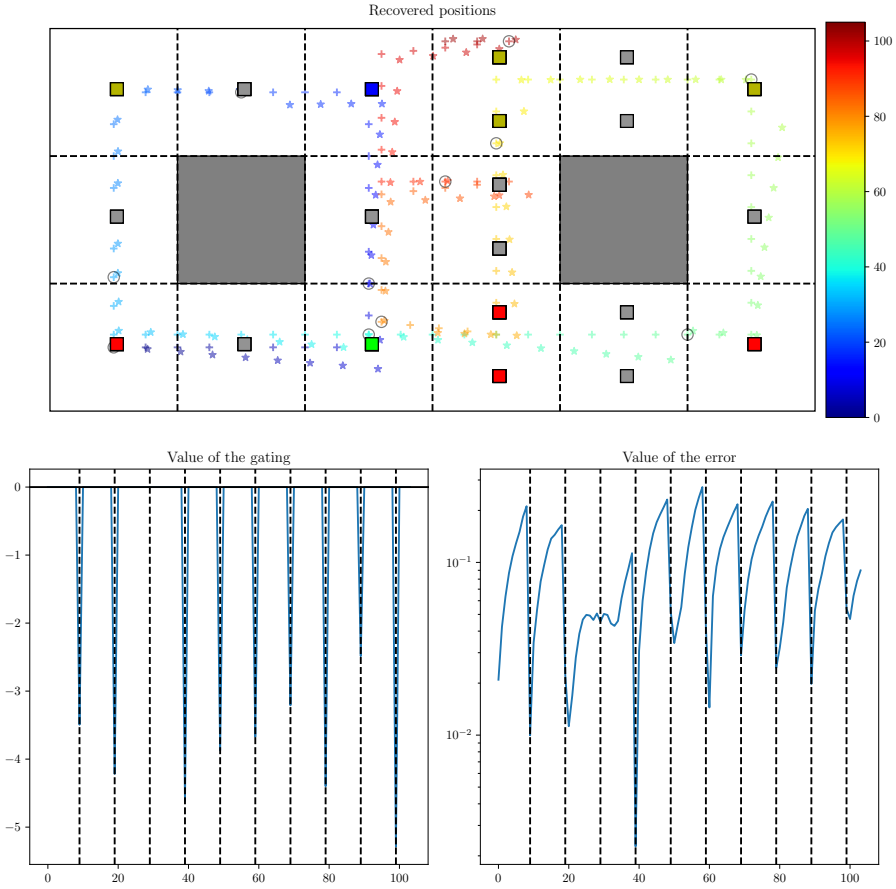


Figure B.11: Example of Path Integration trajectory on the Ambiguous DoubleDonut environment. The network does not exhibit any particular drop in performance upon entering either of the ambiguous rooms, suggesting that the internal state it constructed during Path Integration lifted the ambiguity that is present in the visual cues. It still remains notable that no resetting is performed if the visual cue, even unperturbed, comes from an ambiguous room, a phenomenon further illustrated in Figure B.12.

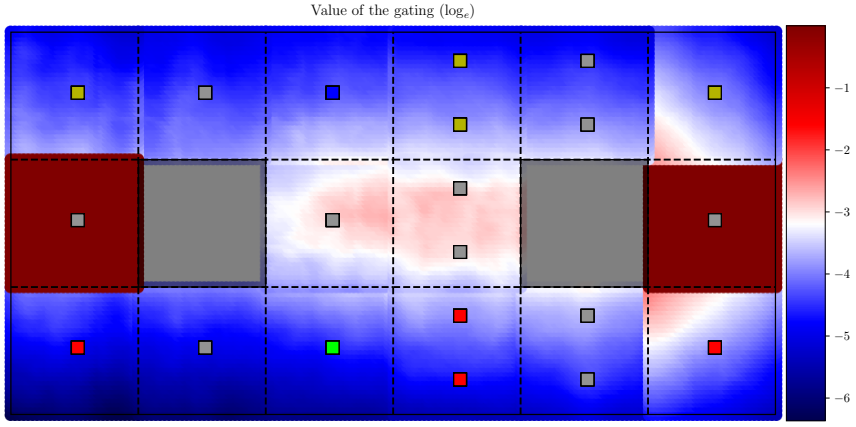


Figure B.12: Value of the natural logarithm of the resetting gate g as a function of position, averaged across 8 realizations of the training. As expected, resetting happens at least partially at every position in the environment, except within the two rooms that have ambiguous visual cues.

RÉSUMÉ

Au cours de la dernière décennie, les méthodes d'apprentissage par réseaux de neurones profonds ont connu un essor sans précédent, fournissant de nouveaux états de l'art dans de nombreux domaines de l'Intelligence Artificielle (vision, analyse de séries temporelles, contrôle continu, etc...). Malgré leur succés empirique évident, beaucoup reste à faire pour comprendre comment ces systèmes se comportent, de leur entraînement aux représentations qui émergent lorsqu'ils performent la tâche qui leur est confiée.
Au cours de cette thèse, nous avons abordé ces questions en étudiant des réseaux récurrents de $n \gg 1$ neurones, entraînés sur le problème de l'intégration en parallèle de $D \simeq 1$ signaux scalaires. Nous observons que, dans le cas de réseaux linéaires comme non-linéarires, l'état interne de la population récurrente évolue dans une variété de dimension D, faible devant la dimension de l'espace des états internes possibles n, et établissons un lien entre la forme de cette variété et la fonction d'activation des neurones. Ces observations nous permettent de proposer une fonction de coût qui, en imposant un ensemble continu de conditions sur la dynamique de l'état interne, permet d'entraîner des réseaux sur une tâche d'intégration arbitraire sans utiliser de données.
Nous étudions également le problème de intégration multimodale du déplacement d'un agent dans un environnement à partir d'images et de signaux proprioceptifs. En particulier, nous cherchons à étudier comment un réseau récurrent parvient à combiner ces deux sources d'information imparfaites (les images étant souvent indisponibles, le signal de vitesse étant bruité) en une représentation commune, qui peut ensuite être transférée vers d'autres tâches impliquant une compréhension de la structure spatiale de l'environnement (par exemple, de la navigation vers un objectif).
Tout au long de ce manuscrit, nous établissons des analogies entre nos résultats et les concepts développés en neurosciences théoriques pour expliquer des comportements similaires observés sur des organismes vivants.

MOTS CLÉS

Réseaux de Neurones Récurrents, Apprentissage Machine, Inférence Statistique, Neurosciences Théoriques.

ABSTRACT

In the last decade, Deep Neural Networks–based methods for Machine Learning have enjoyed un precedented growth, yielding new state-of-the-art results in many domains of Artificial Intelligence (vision, time-series analysis, continuous control, etc...). Despite their undeniable empirical success, a lot of work remains to be done to understand how these systems behave, from their training to the internal representations that emerge while they perform a given task.
During this thesis, we studied these questions through the lens of recurrent neural networks of $n \gg 1$ neurons, trained to perform integration of $D \simeq 1$ scalar signals in parallel. We observe that, for linear as well as non-linear networks, the internal state of the recurrent population evolves within a manifold of dimension D, small compared to the dimension n of the space of possible internal states, and we make a connection between the shape of this manifold and the activation function of the neurons. These observations allow us to propose a cost function which, by imposing a continuous set of conditions on the internal state dynamics, allows training of networks on arbitrary integration tasks, without requiring any data.
We also study the problem of multimodal integration of agent movement within its environment, through visual and proprioceptive signals. In particular, we study how a recurrent neural network manages to combine those two imperfect sources of information (images being often unavailable, and speed signal being noisy) into a joint representation which can then be transfered towards other tasks which require an understanding of the environment's spatial structure (for example, navigation towards a given objective).
All along this manuscript, we establish analogies between our results and concepts developed in theoretical neuroscience to explain similar behaviors observed in living organisms.

KEYWORDS

Recurrent Neural Networks, Machine Learning, Statistical Inference, Computational Neuroscience.