Harnessing symmetries for modern deep learning challenges : a path-lifting perspective

Antoine Gonon

THÈSE

en vue de l'obtention du grade de Docteur, délivré par

l'ÉCOLE NORMALE SUPERIEURE DE LYON

École Doctorale N°512
École Doctorale en Informatique et Mathématiques de Lyon

Discipline : Informatique

Soutenue publiquement le 12/11/2024, par :

Antoine GONON

Harnessing symmetries for modern deep learning challenges: a path-lifting perspective

Exploiter les symétries pour relever les défis modernes de l'apprentissage profond : une perspective par le relèvement dans l'espace des chemins

---

Devant le jury composé de :

BACH, Francis	Directeur de recherche Inria, ENS	Rapporteur
WILLETT, Rebecca	Professeur Université de Chicago	Rapportrice
BLANCHARD, Gilles	Professeur des universités Université Paris-Saclay	Examinateur
BRISEBARRE, Nicolas	Directeur de recherche CNRS, ENS de Lyon	Examinateur
RICCIETTI, Elisa	Maître de conférences ENS de Lyon	Examinatrice
ROSASCO, Lorenzo	Professeur Université de Gênes	Examinateur
GRIBONVAL, Rémi	Directeur de recherche Inria, ENS de Lyon	Directeur de thèse

Remerciements

Mes premiers remerciements vont tout naturellement à Rémi Gribonval, mon directeur de thèse. Travailler avec toi a été un réel plaisir. J'ai tout de suite accroché dès mon stage de L3, et ce n'est pas un hasard si je suis revenu frapper à ta porte deux fois par la suite : d'abord pour obtenir tes recommandations pour un stage de M1 à l'étranger, puis pour le duo stage de M2 et thèse. Merci pour ton accompagnement à chacune de ces étapes, et merci pour la confiance que tu m'as accordée. C'est grâce à ta confiance que j'ai pu pleinement m'épanouir pendant la thèse, me donnant la liberté de suivre mes idées en sachant que j'aurais ton soutien.

J'aimerais aussi te remercier pour ta simplicité, humaine et scientifique. C'est ça qui m'a tout de suite plu en L3 : ta capacité à ramener les choses à l'essentiel, à garder les idées claires, et à faciliter les relations humaines. Tous ces aspects ont rendu nos échanges particulièrement agréables. Merci pour tout.

Cette belle aventure n'aurait pas été la même sans mes deux autres encadrants de thèse, Nicolas Brisebarre et Elisa Riccietti (merci à Rémi de vous avoir inclus dans le processus).

Nicolas, un grand merci pour votre bienveillance et votre disponibilité. Que ce soit pour prendre des nouvelles pendant mes déplacements ou pour m'offrir votre aide avec générosité, même lorsque vous étiez en congés. Votre soutien a été constant. J'ai aussi beaucoup aimé la manière dont vous avez su partager votre expérience, en me donnant de précieux conseils qui m'auraient, sinon, souvent échappés. Je pense aussi à toutes les fois où vous avez eu la présence d'esprit de pointer quelque chose qui était dans l'air mais qui restait encore implicite. Cela nous a souvent permis d'avancer, en vérifiant si Rémi, Elisa, vous et moi étions vraiment sur la même longueur d'onde. Merci pour votre accompagnement et pour votre soutien.

Elisa, grazie un milliardo per l'energia positiva che hai portato durante tutta questa tesi. Non tutti hanno la fortuna di avere un accento italiano che rende magica ogni conversazione! Hai spesso portato una ventata d'aria fresca nelle nostre riunioni, a volte anche involontariamente, con le apparizioni a sorpresa della piccola "ispettore" in videoconferenza, ricordandoci con umorismo l'equilibrio tra la nostra vita professionale e personale. Non posso concludere senza ringraziarti per avermi fatto scoprire la famosa pizza a base bianca (forse dovremmo chiamarla la pizza Riccietti?), che ormai è diventata la mia preferita! Grazie per questi momenti di condivisione e convivialità, che resteranno tra i miei ricordi più cari.

I would like to thank the members of my defense jury, Gilles Blanchard, Lorenzo

Rosasco, Francis Bach, and Rebecca Willett, for dedicating their time and expertise to reviewing my work. I am honored to have had the opportunity to discuss and present my research to such a distinguished group. I am especially grateful to Francis Bach and Rebecca Willett for agreeing to thoroughly review the manuscript.

Merci à Bruno Torresani et Bora Uçar pour leur bienveillance lors de mes comités de suivi individuels de thèse.

J'en viens à remercier toutes les personnes que j'ai côtoyées à l'ENS : les membres de l'équipe, les secrétaires, les enseignants-chercheurs avec qui j'ai été en contact, comme Aurélien, les étudiants en TD. Merci à l'ensemble du personnel de l'ENS, de la personne qui fait l'entretien aux agents de la cantine, toujours sympathiques. Je n'ai eu un que de très bons souvenirs, et c'est grâce à tout l'éco-système de l'école. Voici quelques mots pour ceux dont j'ai été le plus proche, dans un ordre aléatoire.

Pascal, mon seul regret est peut-être que tu ne sois pas arrivé dans l'équipe plus tôt. Merci pour ton investissement dans chacun de nos projets communs. J'ai adoré travailler avec toi et je garde d'excellents souvenirs de tous les moments passés dans ton bureau. À quand une variété de piments chartreux ?

Merci à Mathurin pour tous les bons moments passés ensemble, des plus inoubliables à La Ruisselière aux plus anodins lorsqu'on se croisait dans le couloir en se brossant les dents. Je vais essayer d'être plus assidu aux pièces de théatre pour lesquelles j'ai pris un ticket.

Léon, merci pour ton engagement au travail et ta bonne humeur. Merci également pour tous ces bons moments partagés en dehors du labo. De Lion et Poisson, évidemment, à la pièce en napolitain, le bouillon pimenté force trois – aussi immangeable qu'inoubliable –, et même l'opéra aux accents de déjà-vu.

Clément, merci pour ta joie de vivre, tes avis mesurés et tes raisonnements à la physicienne qui ont su éclairer bien des petits problèmes, où l'enjeu était davantage de les résoudre pour être tranquille d'esprit que pour véritablement avoir la solution.

Merci à Tung, qui complète le trio des anciens doctorants Ockham qui m'ont accueilli, et avec qui j'ai passé de très bons moments. Même si je pense que ce n'est pas loyal d'avoir mis la barre aussi haute pour la nourriture au pot de thèse.

Merci aussi à Samir et Anthony, avec qui j'ai de très bon souvenirs, que ce soit au restaurant, au foot, ou en mangeant les griwechs ramenés par Samir. Je n'ai que pu regretter votre départ pour le troisième étage.

Can, tesekkür ederim pour ta bienveillance et ta gentillesse qui profitent à toute l'équipe au quotidien. J'ai adoré échanger avec toi, et quelque part dans ma liste de tâches reste l'apprentissage du turc – belki bir gün, kısmet!

Merci Etienne pour ton humour qui m'a fait travaillé les muscles du visage plus d'une fois ! Et puis, je ne m'attendais pas à ressortir de la thèse avec une meilleure compréhension des notions de base en météorologie.

Guillaume, merci pour ces moments absolument mémorables, du CIRM au coin café. Il va falloir que je m'habitue à ne plus te voir en entrant dans mon bureau.

Merci Paulo pour toutes ces anecdotes sur Lyon et la nourriture.

Merci à Marion pour m'avoir transmis les codes secrets de plusieurs recettes en patisserie.

Merci à Rémi V pour la bonne humeur et tous nos échanges. On avait plus de chance de se croiser à Vienne qu'à Wien, et pourtant !

Merci Ayoub pour tous ces bons moments. Pour n'en citer qu'un, je pense que tout le monde se souvient de cette fameuse gaffe sur le serveur pirate. Merci aussi pour toutes les fois où tu m'as donné des références précieuses quand je venais te consulter dans ton bureau.

Merci à Badr pour toutes ces explications sur la F1. Même si, je suis désolé, je crois que je fais toujours partie du grand public car j'accroche toujours à la série Netflix. Merci aussi pour ces moments drôles. Est-ce qu'on peut partager l'addition en 8 ?

Merci aux nouveaux, Arthur, Maël et Wassim, pour tous les bons moments passés ensemble. Je sais maintenant que si Arthur prétend être allé quelque part, même un lieu aussi immanquable que la tour Eiffel, il faut lui demander une photo pour vérifier qu'il ne s'est pas trompé de tour radio. Merci pour votre énergie. Vous avez ramené un vent de fraîcheur à l'équipe.

Muchas gracias Gabriel y Clara (lo siento, Clara, pero este mensaje no será en portugués... un idioma a la vez) por ayudarme a aprendar espanol. Gracias también por todos estos buenos momentos, especialmente en los diferentes restaurantes. Solo me queda la pena de que no fuimos a un restaurante brasileño... ¡queda pendiente!

Merci à Sibylle pour les bouffées d'oxygène lorsque tu nous rendais visite, et ces histoires sur ces célébrités qui nous rappelait que Lyon est bien en province. Même si ton arrivée était aussi souvent malheureusement annonciatrice d'une deadline imminente.

Merci à Anne pour ton humour et ta bonne humeur. Il m'était presque autant agréable d'entendre ton retour d'expériences sur des endroits qui me donnent envie (la Corse, Cannes) que sur des endroits qui ne me donnent pas envie (je ne veux pas me mettre de pays à dos alors je ne citerai que le TNP). C'est dire comme je garde un bon souvenir de nos discussions !

Merci à Titouan pour ton authenticité. Je me demande si ta ténacité contre vents et marées n'a pas été acquise en Bretagne. Est-ce que c'est le moment d'avouer que j'étais parfois secrètement d'accord avec tes idées insensées de ne pas chercher à passer à l'échelle ? Je ne sais pas, je vais demander à ChatGPT.

En relisant les lignes ci-dessus, je me rends compte avoir en somme remercié tout le monde pour sa positivité. Ce n'est pas par manque d'originalité : je réalise que j'ai juste eu de la chance d'être dans un groupe avec une telle énergie. Et puis, l'autre thème commun à tous ces remerciements est sans doute la nourriture, et pour ça, merci Lyon!

Enfin, ces remerciements ne seraient pas complets si je ne mentionnais pas les personnes qui constituent ma grande et belle famille.

Mes amis d'enfance dans un ordre aléatoire : Gromle, Chol, Mass, Flo, Nkmk, Toto, Piflou, Rieut.

Ma soeur.

Papa, maman.

Je vous aime.

Merci.

Abstract

Neural networks have demonstrated impressive practical success, but theoretical tools for analyzing them are often limited to simple cases that do not capture the complexity of real-world applications. This thesis seeks to narrow this gap by making some theoretical tools more applicable to practical scenarios.

The first focus of this work is on generalization: can a given network perform well on previously unseen data? This thesis improves generalization guarantees based on the path-norm and extends their applicability to ReLU networks incorporating pooling or skip connections. By reducing the gap between theoretically analyzable networks and those used in practice, this work provides the first empirical evaluation of these guarantees on practical ReLU networks, such as ResNets.

The second focus is on resource optimization (time, energy, memory). This thesis introduces a novel pruning method based on the path-norm, which not only retains the accuracy of traditional magnitude pruning but also exhibits robustness to parameter symmetries. Additionally, this work presents a new GPU matrix multiplication algorithm that enhances the state-of-the-art for sparse matrices with Kronecker-structured support, achieving gains in both time and energy. Finally, this thesis makes approximation guarantees for neural networks more concrete by establishing sufficient bit-precision conditions to ensure that quantized networks maintain the same approximation speed as their unconstrained real-weight counterparts.

Résumé

Les réseaux de neurones connaissent un grand succès pratique, mais les outils théoriques pour les analyser sont encore souvent limités à des situations simples qui ne reflètent pas toute la complexité des cas pratiques d'intérêts. Cette thèse vise à réduire cet écart en rendant certains outils théoriques plus concrets.

Le premier axe de recherche concerne la généralisation : un réseau donné pourra-t-il bien se comporter sur des données jamais vues auparavant? Ce travail améliore les garanties de généralisation basées sur la norme de chemins, les rendant applicables à des réseaux ReLU incluant du pooling ou des connexions résiduelles. En réduisant l'écart entre les réseaux analytiquement étudiables et ceux utilisés en pratique, cette thèse permet la première évaluation empirique de ces garanties sur des réseaux ReLU pratiques tels que les ResNets.

Le second axe porte sur l'optimisation des ressources (temps, énergie, mémoire). Une nouvelle méthode d'élagage des paramètres, fondée sur la norme de chemins, est proposée. Cette approche conserve non seulement la précision de l'élagage par amplitude, tout en étant robuste aux symétries des paramètres. Cette thèse fournit aussi un nouvel algorithme de multiplication de matrices sur GPU qui améliore l'état de l'art pour les matrices creuses à support de Kronecker, offrant des gains en temps et en énergie. Enfin, ce travail rend les garanties d'approximation pour les réseaux de neurones plus concrètes en établissant des conditions suffisantes de précision en bits pour que les réseaux quantifiés conservent la même vitesse d'approximation que les réseaux avec des poids réels non contraints.

Résumé des chapitres en français

Les réseaux de neurones sont abondamment utilisés aujourd'hui. Un réseau de neurones est simplement une manière de décrire un certain type de fonctions en utilisant des paramètres, tout comme un polynôme est une manière de décrire une fonction en utilisant des coefficients. Dans le Chapitre 1, je rappelle deux grands enjeux des réseaux de neurones auxquels je vais m'intéresser dans cette thèse.

Le premier défi est celui d'obtenir des garanties statistiques de généralisation. En pratique, les paramètres d'un réseau de neurones sont ajustés pour que le réseau fasse de bonnes prédictions sur une tâche donnée. Par exemple, il peut s'agir de prédire si une image contient un chien ou non. Dans cette thèse, je considère le cadre de l'apprentissage supervisé, où les paramètres sont ajustés en présentant au réseau des exemples de données et en lui indiquant la réponse attendue pour chaque exemple. Dans le cas de la reconnaissance de chiens, on pourrait lui présenter des images avec des chiens et des images sans chiens, et lui indiquer les réponses attendues. Une fois que le réseau s'est suffisamment amélioré sur les exemples d'entraînement, on veut que le réseau fasse de bonnes prédictions sur des images qu'il n'a jamais vues. C'est ce qu'on appelle la généralisation. En pratique, il est possible de trouver de bonnes valeurs pour les paramètres qui permettent au réseau de bien généraliser, alors que le nombre de paramètres est grand (supérieur à 500 milliards pour Narayanan et al. [2021], par exemple) et excède souvent le nombre de données d'entraînement disponibles, si bien qu'un réseau pourrait simplement faire de bonnes prédictions sur les données d'entraînement en apprenant par cœur les réponses attendues pour ces données [Zhang et al., 2021]. Il y a donc un véritable enjeu à comprendre pourquoi les paramètres appris en pratique arrivent à généraliser, alors qu'il avait aussi des possibilités d'avoir une performance similaire sur les données d'entraînement sans pour autant généraliser [Shalev-Shwartz and Ben-David, 2014, Goodfellow et al., 2016, Bach, 2024].

Concernant ce défi, je commence par introduire en Chapitre 2 le modèle de réseaux que je vais analyser. Ce modèle couvre n'importe quel réseau de neurones ReLU organisé en graphe acyclique dirigé, et contenant des ingrédients désormais classiques, tels que le max-pooling ou les connexions résiduelles. Cela couvre en particulier beaucoup de réseaux de neurones très utilisés en pratique, tels que les architectures ResNets, VGGs, Alexnet, U-nets, ReLU MobileNets, et même les architectures utilisées dans les fameux AlphaGo et AlphaZero développées par DeepMind pour jouer au Go.

Dans le Chapitre 2, j'étends à ce modèle général de réseaux de neurones une gamme d'outils, centrée sur le relèvement dans l'espace des chemins, qui a émergé dans la littérature récente pour l'analyse théorique des réseaux, mais seulement pour des modèles simples qui ne reflètent pas la complexité des réseaux utilisés en pratique. Le défi lors de l'extension de ces outils est de réussir à préserver leurs propriétés intéressantes, tout en les rendant opérationnels pour des réseaux plus généraux. Je réalise une telle extension en préservant les deux propriétés phares de ces outils : la capture des symétries des réseaux ReLU ainsi que leur structure affine par morceaux. Ces deux propriétés ont été utilisées de manière cruciale dans la littérature récente pour établir des garanties théoriques en généralisation [Neyshabur et al., 2015, Barron and Klusowski, 2019], en identifiabilité des réseaux [Bona-Pellissier et al., 2022, Stock and Gribonval, 2023], ou encore sur leur

dynamique d'entraînement [Marcotte et al., 2023], mais seulement pour des modèles *simples*. Grâce au travail du Chapitre 2, je vais pouvoir exploiter dans les autres chapitres ces outils prometteurs pour des modèles bien *plus généraux*.

Dans le Chapitre 3, j'établis de nouvelles garanties sur le caractère Lipschitz de ces réseaux très généraux. Le caractère Lipschitz est une propriété primordiale des réseaux, car il renseigne sur la variation des sorties du réseau lorsque les entrées sont perturbées, ou lorsque les paramètres du réseau sont perturbés. Les nouvelles garanties que j'établis sont fondées sur le relèvement dans l'espace des chemins, introduit en Chapitre 2, et ont plusieurs propriétés particulièrement intéressantes : elles sont facilement calculables, elles sont invariantes par les symétries de remise à l'échelle et de permutation des réseaux, et elles sont plus fines que certaines garanties classiques exprimées directement en fonction des paramètres (avec des produits de normes de couches).

Dans le Chapitre 4, j'utilise crucialement ces nouvelles propriétés Lipschitz pour obtenir les premières garanties de généralisation fondées sur la norme de chemins, pour des réseaux de neurones aussi généraux que ceux mentionnés ci-dessus. La norme de chemins est une mesure de complexité des paramètres qui a éveillé un grand intérêt dans la littérature pour tenter d'expliquer la généralisation des réseaux, car elle a plusieurs propriétés désirables : elle est invariante par les symétries de remise à l'échelle et de permutation des réseaux, elle est plus fine que certaines mesures de complexité classiques basées sur des produits de normes de couches, elle est facilement calculable, et enfin elle est l'une des meilleures mesures de complexité connue basée sur la taille des paramètres, en matière de corrélation avec la généralisation mesurée empiriquement [Jiang et al., 2020, Dziugaite et al., 2020]. L'extension des garanties de généralisation basées sur la norme de chemins aux réseaux de neurones plus généraux que ceux considérés dans la littérature précédente est donc une étape importante, qui va me permettre d'évaluer pour la première fois ce type de garanties sur des réseaux vraiment utilisés en pratique. Cela me permettra en Chapitre 7 d'identifier les défis restants pour obtenir, à partir de la prometteuse norme de chemins, des garanties de généralisation plus précises. Un enjeu majeur est notamment la conception de mesure de complexité qui correspondrait au comportement *moyen* du réseau, plutôt qu'à son comportement *pire cas*, sans quoi le cadre théorique actuel n'est pas assez fin pour expliquer la généralisation dans les cas où il y a plus de données d'entraînement que de paramètres, et où il y a la possibilité d'apprendre par cœur les données d'entraînement.

Le deuxième défi que je vais aborder dans cette thèse est celui de l'efficacité des réseaux de neurones. Les réseaux de neurones sont très coûteux en ressources, que ce soit en terme de temps de calcul, de mémoire ou d'énergie dépensée, et la tendance actuelle est de continuer à augmenter le nombre de paramètres et le nombre de données d'entraînements afin d'obtenir les meilleures performances possibles. Il a par exemple été estimé que le coût de déploiement des nouveaux réseaux de neurones états de l'art double tous les 3 à 6 mois [OpenAI, May 2018, Kaplan et al., 2020, Sastry et al., 2024]. La réduction de ces coûts est donc un enjeu majeur. Cette thèse va aborder ce problème sous différents angles.

D'abord, j'utilise les nouvelles garanties de robustesse des réseaux à la perturbation des paramètres établies en Chapitre 3 pour motiver la conception d'un nouvel algorithme d'élagage des réseaux dans ce même chapitre. L'élagage consiste à mettre à zéro des paramètres du réseau pour diminuer le nombre de paramètres qui rentrent en jeu dans les

calculs, et donc les ressources nécessaires à l'utilisation du réseau. Cet algorithme est basé sur la norme de chemins, et je montre expérimentalement qu'il atteint les mêmes performances que l'élagage par amplitude, une méthode couramment utilisée pour ses bonnes performances [Frankle and Carbin, 2019]. En outre, ce nouvel algorithme a l'avantage d'être robuste aux symétries de remise à l'échelle des paramètres, alors que je montre que ce n'est pas le cas de la technique classique d'élagage par amplitude, dont les performances chutent drastiquement dès lors qu'on commence à remettre à l'échelle les paramètres, et ce, sans même chercher délibérément à réduire ses performances, seulement en remettant à l'échelle de manière aléatoire.

Le deuxième volet de la thèse qui concerne l'efficacité des réseaux de neurones est abordé en Chapitre 5, où je m'intéresse à l'utilisation de matrices creuses à support Kronecker, que j'appelle plus simplement par la suite matrices Kronecker-creuses, dans les réseaux de neurones. Les matrices Kronecker-creuses sont des matrices contenant beaucoup de zéros, et dont l'emplacement de ces zéros obéit à une structure particulière, définie par un produit de Kronecker. Ces matrices ont récemment reçues de l'attention dans la littérature car la multiplication de ces matrices a un faible coût théorique, et en pratique, ces matrices permettent dans certaines configurations d'obtenir les mêmes performances que les matrices denses utilisées traditionnellement, mais avec moins de paramètres. Dans le Chapitre 5, je commence par étudier l'efficacité des implémentations existantes pour la multiplication avec une matrice Kronecker-creuse sur GPU. Ces expériences montrent que les implémentations actuelles passent beaucoup de temps à transférer des données entre les différents niveaux de mémoire du GPU. Je propose alors un nouvel algorithme qui vise à réduire les coûts liés à ces transferts de données, et je montre expérimentalement en précision flottante que cet algorithme est non seulement plus rapide que les implémentations existantes, mais qu'il permet aussi de réduire l'énergie dépensée. Je discuterai en Chapitre 7 des perspectives pour étendre ces résultats à la précision réduite, qui est souvent utilisée en pratique pour réduire les coûts de calculs.

Enfin, je m'intéresse à la quantification des réseaux de neurones en Chapitre 6. La quantification consiste à remplacer les paramètres d'un réseau par des versions approchées, en utilisant un nombre restreint de bits mémoire pour les stocker. Cette technique est largement utilisée en pratique puisqu'elle permet de réduire les coûts de calculs et de stockage. Le développement de garanties sur les méthodes de quantification est un enjeu majeur puisque pour l'instant, la recherche de bonnes méthodes de quantification se fait par essais-erreurs. Dans le Chapitre 6, j'aborde la question fondamentale de l'approximation de fonction avec des réseaux quantifiés. Peut-on toujours approcher les mêmes fonctions quand on décide de diminuer le nombre de bits utilisé pour représenter les paramètres d'un réseau ? J'apporte des premiers éléments de réponse lorsque les poids sont quantifiés au plus proche voisin sur une grille uniformément espacée, en établissant un nombre de bits suffisant par poids pour que les réseaux puissent approcher les mêmes fonctions que leur version avec des poids sans contraintes, et en gardant les mêmes taux polynomiaux d'approximation dans un espace L^p. Je discute dans le Chapitre 7 de perspectives pour la prise en compte des symétries dans la conception des méthodes de quantification, qui pourrait permettre d'améliorer leur précision à un nombre de bits donné.

Enfin, j'aborde également dans le Chapitre 6 la question de l'existence de limites fondamentales à l'approximation avec des réseaux de neurones, quantifiés ou non. J'identifie une propriété générale des familles d'approximation (réseaux de neurones, ondelettes,

polynômes, etc.) qui permet d'unifier et de généraliser une limite fondamentale bien connue pour certaines familles : il est impossible d'approcher les fonctions d'un ensemble C plus vite qu'une certaine complexité de l'ensemble C du ressort de la théorie de l'information. Ce lien entre la théorie de l'approximation et la théorie de l'information est rendu possible par une propriété qui était jusqu'à présent utilisée de manière implicite et établie au cas par cas dans la littérature, et que j'identifie explicitement : la vitesse polynomiale en ε à laquelle on peut recouvrir la famille considérée par des boules de rayon ε. Ce nouveau cadre unificateur montre que l'existence de ce type de limite fondamentale sur la vitesse d'approximation d'un ensemble C par une certaine famille d'approximants provient d'une propriété intrinsèque à la famille d'approximants, et non pas de son adaptation à la classe C.

Contents

Chapter 1

Introduction

What do the tasks of recognizing cancerous cells in medical images, translating languages, driving autonomous vehicles, composing music, and recommending personalized content on social media have in common? They all involve extracting features from raw data. Remarkably, artificial neural networks, or simply "neural networks", are empirically successful to extract relevant features in all these different cases [LeCun et al., 2015, Goodfellow et al., 2016].

As depicted in Figure 1.1, this success heavily relies on the availability of large datasets, and the ever increasing availability of large-scale computing power, which makes it possible to process such large datasets using large models.

Computing Power Large Datasets Large Models

Figure 1.1: Elements contributing to the success of neural networks.

Today, the main paradigm for improving performance is to scale everything up: more computing power, more data, and more parameters in the models [OpenAI, May 2018, Kaplan et al., 2020, Karpathy, 2022, Sevilla et al., 2022, Hoffmann et al., 2022, Sastry et al., 2024, Aschenbrenner, 2024]. This approach, however, raises several issues, including interpretability challenges and increased energy consumption. The cost of maintaining and developing such large-scale systems is indeed doubling approximately every six months, as shown in Figure 1.2. Moreover, a fine-grained understanding of neural networks is still lacking, as they are predominantly viewed as big black boxes, and the bigger the box, the better it seems to perform.

This thesis aims to better understand neural networks by developing tools that can capture some of their key properties. This is part of a long line of research whose long-

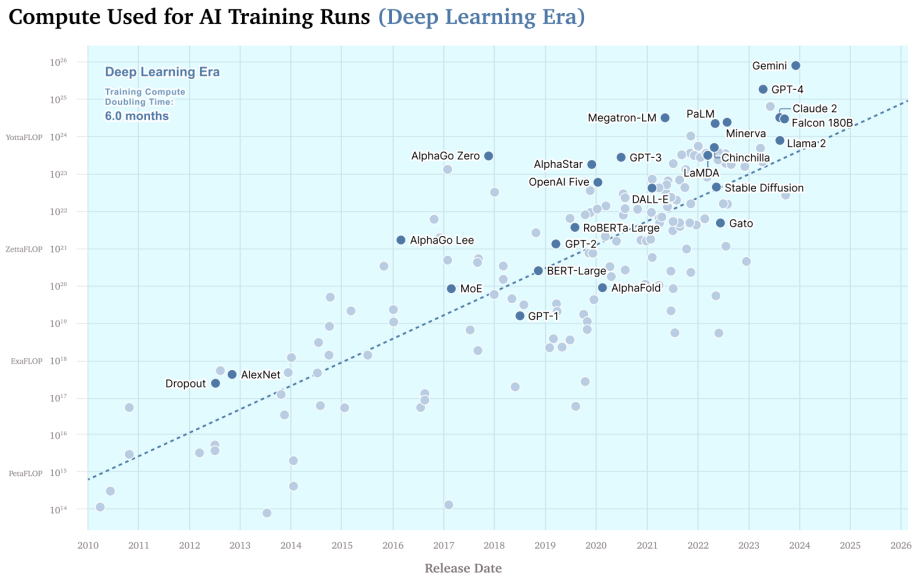
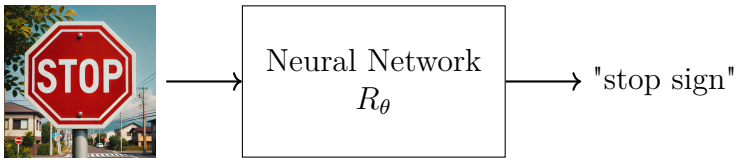


Figure 1.2: Cost of machine learning doubles every 6 months [Sastry et al., 2024]

term goal is to build an understanding that is both theoretically and practically useful, leading to more interpretable, efficient, and sustainable neural network models.



Input Image x

Figure 1.3: Example of a Neural Network

A neural network has parameters θ and these parameters realize a function R_θ that maps an input (an image of a stop sign in Figure 1.3) to an output (the label "stop sign" in Figure 1.3). In this thesis, I will be mainly interested in statistical aspects of neural networks and in their resource efficiency.

Generalization is the main statistical property of neural networks that I will focus on. In practice, the parameters θ of a network are adjusted during a phase called training. The goal of the training phase is to make R_θ perform well on a set of training samples. I will be interested in the so-called supervised learning framework, where pairs of inputs along with their desired outputs are available for training. In the example from Figure 1.3, we could give the network some images of road signs as inputs, and the corresponding labels as target outputs, such as "stop sign", "speed limit", etc. The training phase aims at adjusting the parameters θ so that the network outputs the correct label for each input image. Once the performance on the training data is satisfactory, we may want to use this network in a car to recognize signs on the road. Will it recognize correctly *new* images of road signs that haven't been used for training? This is the key question of generalization: the ability of a model to extend its learned abilities from seen to unseen data.

Ressource efficiency aims at reducing the time, energy, and/or memory required to use neural networks. One way to achieve this is to *prune* (set to zero) coordinates of θ while preserving the prediction accuracy of the function R_θ. More generally, one can explicitly constrain θ to be *sparse* (having only a few nonzero entries). Depending on the type of sparsity, different gains in time, memory or energy can be achieved. For example, *structured* sparsity is generally better suited than *unstructured* sparsity for achieving time gains, since the known *structure* (the location of zeros) can be injected *a priori* into the algorithms to optimize computations. Another way is to *quantize* the parameters θ, which consists in reducing, as much as possible, the number of bits used to store each coordinate of θ.

Informal overview of the contributions. This thesis aims at making more concrete tools that have been identified as promising for the analysis of neural networks and for the improvement of their efficiency. The main contributions are as follows.

- This thesis improves the **generalization** guarantees based on the so-called path-norm, a proxy of the "slope" (Lipschitz constant) of ReLU neural network functions (introduced below), and make these guarantees more widely applicable to cope with ReLU networks that also incorporate now-standard ingredients such as so-called pooling or skip connections. This closes the gap between the range of networks that can be theoretically analyzed with this kind of tools and the range of networks that are used in practice. This results in the first assessment of this kind of guarantees for widely used networks such as so-called ResNets [He et al., 2016].

- The usefulness of the path-norm is further demonstrated by designing a new **pruning** method that sets a given number of parameters to zero in order to **compress** them. This new pruning method not only empirically retains the performance of standard pruning methods, but it is also robust to *key intrinsic symmetries* of the parameters.

- The thesis leverages **structured sparsity** to decrease the time, energy and memory needed to run neural networks. I investigate sparse matrices with supports defined by Kronecker products as they have a small theoretical matrix multiplication complexity and have been shown to match the accuracy of dense matrices when used in neural networks [Dao et al., 2020, Lin et al., 2021, Chen et al., 2022, Dao et al., 2022a]. However, their practical interest in terms of time, energy, and memory had not been comprehensively studied so far. I address this by extensively benchmarking existing GPU implementations for Kronecker-sparse matrix multiplication and by proposing a new GPU multiplication algorithm that outperforms the state-of-the-art.

- The thesis also aims at making the approximation guarantees of ReLU neural networks more concrete. While the universal approximation theorem states that neural networks with arbitrary real weights can theoretically approximate any sufficiently smooth function [Leshno et al., 1993], practical networks operate with specific floating-point constrained weights. This thesis refines and extends concrete sufficient bit-precision conditions under which **uniformly quantized** ReLU networks

maintain the same approximation polynomial rates as those with unconstrained real weights.

The rest of the introduction explains why this thesis focuses on ReLU neural networks, and how their piecewise affine structure and symmetries are crucial to understand their behavior.

1.1 Why ReLU neural networks?

The simplest type of neural networks corresponds to the alternate composition of affine maps with a so-called activation function. In this thesis, I will focus on the ReLU activation function, a popular choice for many reasons that I now discuss.

Definition 1.1.1 (ReLU activation function)**.** *The ReLU activation function is defined as*

$$\rho(x) = \max(x, 0), \quad \text{for } x \in \mathbb{R}$$

and is extended to vectors by applying it element-wise.

Here are some **theoretical** reasons to consider the ReLU activation function.

- Since the ReLU is non-polynomial, this guarantees that ReLU neural networks can approximate any sufficiently smooth function, a result known as the universal approximation theorem, see, e.g., Leshno et al. [1993].
- The ReLU is positively homogeneous, meaning that $\rho(\lambda x) = \lambda\rho(x)$ for $\lambda > 0$. This property implies the existence of *symmetries*. This yields interesting practical behaviors that can be theoretically analyzed. As an example, Emmy Noether's famous theorem states that every symmetry of a physical system leads to a conservation law. Using this in the context of neural networks allows for precise predictions of the behaviors of certain norms of the parameters θ during training [Kunin et al., 2021, Marcotte et al., 2023].
- The functions associated with ReLU neural networks are piecewise affine functions. Therefore, the theoretical analysis of their behavior can be done locally on each affine part of the network. This is a key property used throughout this thesis.

Here are some **practical** reasons to consider the ReLU activation function.

- The ReLU is fast to compute as it only requires a comparison of the input x with zero, while other activation functions such as the sigmoid $x \mapsto 1/(1 + e^{-x})$ or the tanh $x \mapsto \tanh(x)$ require more complex computations.
- The ReLU is differentiable everywhere except for $x = 0$, with derivative 0 or 1, which is great in practice to avoid numerical instability when computing gradients. This is in contrast to other activations such as the sigmoid or the tanh: they have arbitrarily small but non-zero derivatives for large inputs x, which can lead to vanishing gradients and slow gradient-based learning [Krizhevsky et al., 2012, Table 1].

- Over recent years, many state-of-the-art networks have been built around the ReLU activation function. Below is a chronological list of models that have made substantial contributions to the field of computer vision: AlexNet [Krizhevsky et al., 2012], VGGs [Simonyan and Zisserman, 2015], Inception Net [Szegedy et al., 2015], ResNet [He et al., 2016], ReLU MobileNets [Howard et al., 2017] and Transformers [Dosovitskiy et al., 2021]. The results of this thesis are applicable to all these models as all the operations they involve are covered in this thesis, with the exception of Transformers since it uses an operation called attention that is not covered here.

  In addition to computer vision, ReLU networks have also gained attention in natural language processing (NLP). Although many recent NLP models do not predominantly use the ReLU activation function, recent research suggests strong motivations to reintroduce the ReLU in current models to accelerate computations without compromising their accuracy [Mirzadeh et al., 2024].

  ReLU neural networks are also employed in reinforcement learning, such as in the famous AlphaGo [Silver et al., 2016, 2017] and AlphaZero [Silver et al., 2018] models.

1.1.1 ReLU Neural networks and their piecewise affine structure

One of the main characteristics of ReLU neural networks activation function is their piecewise affine structure.

The neural network function realized by a vector of parameters θ is denoted by $R_\theta : \mathbb{R}^{d_{\text{in}}} \to \mathbb{R}^{d_{\text{out}}}$. This function maps an input $x \in \mathbb{R}^{d_{\text{in}}}$ to a representation $R_\theta(x) \in \mathbb{R}^{d_{\text{out}}}$. A typical situation is to train (adjust) the parameters θ on some training data $(x_i, y_i)_{i=1}^n \in (\mathbb{R}^{d_{\text{in}}} \times \mathbb{R}^{d_{\text{out}}})^n$ in order to make the representation $R_\theta(x_i)$ relevant for predicting y_i. Depending on the values of the parameters θ, the function will learn different representations of the data that will be more or less useful for the considered task. Therefore, it is crucial to understand how the properties of the learned parameters θ affect the patterns of the training data $(x_i, y_i)_i$ that are captured by the function R_θ.

Piecewise affine structure. For a given value of the parameters θ, the function $R_\theta : \mathbb{R}^{d_{\text{in}}} \mapsto \mathbb{R}^{d_{\text{out}}}$ is piecewise affine. This means that it partitions the input space $\mathbb{R}^{d_{\text{in}}}$ into (polytope) regions, and on each of these regions, the function R_θ is simply affine [Arora et al., 2017]. This is illustrated[1] in Figure 1.4.

The regions and the coefficients of the affine maps within each region, as determined by the parameters θ, influence the data patterns captured by the function R_θ. We can already imagine that an increase in the number of regions allows the function R_θ to capture more diverse and complex data patterns.

Consider a simple example where the task is to discriminate between two types of objects, such as images of stop and speed limit signs. If it is possible to divide the input space with a hyperplane such that stop signs are on one side and speed signs are on the other one, we can imagine a neural network that successfully identifies this partitioning of the space. For instance, the network could output zero on one side of the hyperplane and

[1] Source of the illustration: https://commons.wikimedia.org/wiki/File:Piecewise_linear_function2D.svg, made by Oleg Alexandrov.

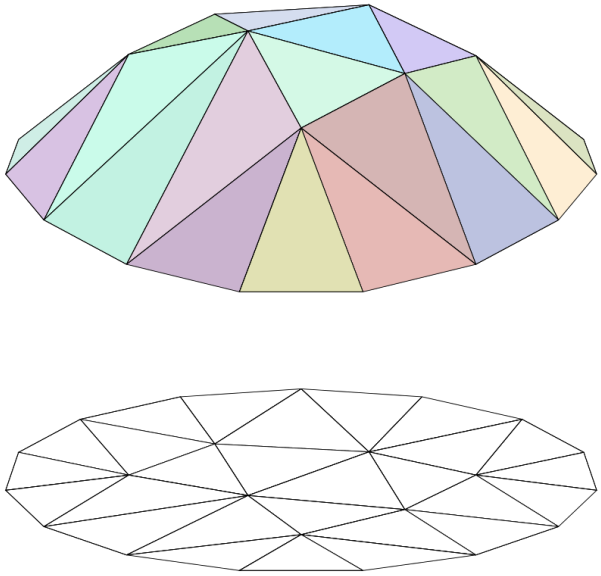


Figure 1.4: Neural network corresponds to piecewise affine functions.

a strictly positive value on the other. The output value of the network would therefore indicate which category the input belongs to.

Relevance of the piecewise affine structure. In a given region, the slope (magnitude of the affine coefficients) indicates how rapidly the network changes its output (and consequently the decision made based on its output, for example, discriminating between stop signs and speed limit ones). The lower the slope, the more *robust* the network is to input perturbations. When the regions and slopes have been learned from training data on which the network now performs well, a key statistical problem is to guarantee the same performance on new similar data. If the network's output varies little for nearby data points because of low slopes, then the output will remain similar. Thus, the magnitude of the slopes can provide insight into how well the network will generalize what it has learned from the training data to new, similar data.

In this thesis, I will define objects that encode the different regions as well as the affine coefficients, and use them to establish robustness and generalization guarantees for neural networks. These objects are respectively named the path-activations A and the path-lifting Φ.

1.1.2 ReLU Neural networks and their symmetries

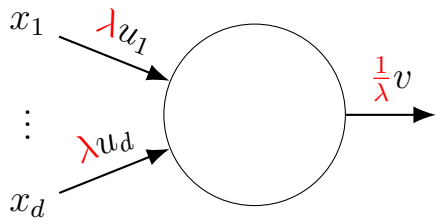
Besides the piecewise affine structure discussed in the previous section, neural networks have *symmetries* that are also crucial to understand their behavior, as explained in this section.

Description of the symmetries. There are transformations of the weights θ that leave the function R_θ unchanged. These symmetries appear at the smallest level of computation in a neural network: the neuron.

Consider the case of a neuron with incoming weights $u_1, \dots, u_d$ and one outgoing weight v, as shown in Figure 1.5. Given scalar inputs $x_1, \dots, x_d$, the output of the neuron is

$$v\rho\left(\sum_{i=1}^{d} u_i x_i\right)$$

where ρ is a so-called activation function. A standard choice for ρ is a positively-homogeneous function such as the ReLU function, max-pooling ($\rho(x_1, \dots, x_n) = \max(x_1, \dots, x_n)$), or average-pooling ($\rho(x_1, \dots, x_n) = (\sum_{i=1}^{n} x_i)/n$). Positive-homogeneity implies the existence of rescaling symmetries, or simply symmetries: scaling up the incoming weights by $\lambda > 0$ and scaling down the outgoing weight by λ does not change the input-output mapping, even though the parameters have changed. This is a transformation of the parameters that leaves the function unchanged.



$$v\rho\left(\sum_{i=1}^{d} u_i x_i\right) = \tfrac{1}{\lambda} v\rho\left(\sum_{i=1}^{d} \lambda u_i x_i\right)$$

Figure 1.5: Symmetries arise at the neuron-level: scaling up incoming weights by $\lambda > 0$ and scaling down the outgoing weight by λ does not change the input-output mapping if ρ is positive-homogeneous.

I denote by $(\theta, \lambda) \mapsto \lambda \diamond \theta$ the mapping that rescales the parameters of a neural network and say that two sets of parameters θ and $\lambda \diamond \theta$ are *rescaling-equivalent*. Note that $\lambda \diamond \theta$ is *not* θ multiplied by λ: some coordinates are multiplied by λ, some others by $1/\lambda$, as shown in Figure 1.5. Rescaling equivalent parameters θ and $\lambda \diamond \theta$ are functionally equivalent: $R_{\lambda \diamond \theta} = R_\theta$, but they can behave very differently in other aspects, such as norms, pruning, and SGD dynamics. *Due to these different behaviors, a guarantee established on ReLU neural networks can significantly vary depending on the scaling of the parameters if this guarantee is not invariant.* Let me show two consequences of the lack of invariance to symmetries: bounds that can be non-informative, and state-of-the-art compression methods that can lead to very poor performance.

Consequence 1: Non-invariant bounds can be vacuous. Imagine you have a bound that looks like

$$\text{Interesting Quantity}(R_\theta) \leqslant C\|\theta\|$$

where C is a constant that can depend on many things but not on θ, and where $\|\cdot\|$ is some kind of norm. Since the interesting quantity only depends on the function R_θ, it is invariant to rescaling $\lambda \mapsto \lambda \diamond \theta$:

$$\text{Interesting Quantity}(R_\theta) = \text{Interesting Quantity}(R_{\lambda \diamond \theta})$$

In general, the bound is not invariant to rescaling:

$$\|\theta\| \neq \|\lambda \diamond \theta\|$$

and, even worse than that, this norm can be arbitrarily large when λ gets to infinity:

$$\|\lambda \diamond \theta\| \to \infty \text{ as } \lambda \to \infty,$$

thus the size of the bound depends on how lucky we are with the scale of the parameters θ at hand. In particular, the bound can be vacuous if θ is not properly scaled.

A generic way to make the bound invariant would be to consider:

$$\text{Interesting Quantity}(R_\theta) \leqslant C \inf_\lambda \|\lambda \diamond \theta\|.$$

However, this may not be practical as it is not obvious whether computing this infimum is easy. One of the objectives of this thesis will be to derive bounds:

- invariant to symmetries to avoid vacuous bounds when not properly scaled,
- easy to compute,
- valid on practical networks, and
- as faithful[2] as possible to the network's behavior.

Consequence 2: the magnitude of the weights fails to characterize their importance. Rescaling symmetries change the scale of the weights, without changing the function realized by the network. This is a problem for methods that directly base their decisions on the magnitude (absolute value) of the weights. An important example is magnitude pruning: it consists of setting to zero the smallest weights of θ to reduce the number of parameters. While this technique is the gold standard in many situations, I will show that because of its lack of invariance under symmetries, it can lead to poor performance when the parameters are not properly scaled.

Conclusion. Taking into account the symmetries is therefore crucial. The central tools that I will use to analyze neural networks are the path-lifting Φ and the path-activations A. Besides capturing the affine regions and the coefficients of the affine functions as mentioned in Section 1.1.1, these objects are also invariant under the rescaling symmetries, which will prove to be very useful in this thesis.

1.2 Outline

In this section, I provide an outline of the thesis, summarizing the main topics tackled in each chapter and the connections between them.

Chapters 2 to 4 form a cohesive unit centered around the path-lifting and path-activations, based on Gonon et al. [2024a,b]. The presentation and insights provided in these chapters have evolved significantly since the original submissions. This refined perspective emerged from taking a step back, engaging in discussions with colleagues, and further reflecting on the results. Therefore, while the core ideas are rooted in these papers, the structure and narrative presented here hopefully offer a clearer perspective on the contributions.

[2]It is desirable to manipulate quantities of θ that enjoy similar behavior as the functions R_θ: same symmetries, correlation with important quantities of R_θ and so on.

Chapter 2 introduces the concepts of path-lifting and path-activations. These objects, previously considered in the literature for simple ReLU neural networks, capture the networks' piecewise affine structure and symmetries. The chapter begins by reviewing the simple model of layered fully-connected networks for which these tools were already known. Then, this chapter extends the path-lifting and path-activations to a more general model of ReLU networks, incorporating operations like max-pooling and skip connections. These extended tools will be shown to preserve their essential properties: : capturing the piecewise affine structure and the symmetries of the networks. This sets the stage for their use in neural network analysis.

Chapter 3 demonstrates that the path-lifting and path-activations allow to establish Lipschitz bounds for ReLU networks both with respect to the input x and the parameters θ. These bounds rely on mixed ℓ^q-norms of the path-lifting, called mixed path-norms, and Chapter 3 show that these norms can be easily computed in one forward-pass of the model. The Lipschitz property in x extends to general ReLU networks possibly with max-pooling and skip connections (e.g., ResNets) previous results that were only known for layered fully-connected networks. The Lipschitz bound in θ is novel and Chapter 3 already shows a first concrete application of this bound: it is used to design a new compression method for neural networks via pruning. This method not only achieves comparable accuracy to standard magnitude pruning but also exhibits robustness to intrinsic parameter symmetries. The utility of the new Lipschitz bound in θ will further be illustrated in the context of generalization guarantees.

Chapter 4 establishes upper bounds on the Rademacher complexity of ReLU neural networks in terms of the path-norm. This bound, which is invariant under the symmetries of the network and easy to compute, extends prior results established for layered fully-connected networks to the case where there can also be max-pooling, skip-connections etc. This bridges the gap between the networks that can be theoretically analyzed with the path-lifting and the ones used in practice.

Chapters 5 and 6 tackle different aspects of resource efficiency, one of the main subjects addressed in this thesis. Chapter 5 addresses the efficiency of matrix multiplication in neural networks [Gonon et al., 2024c], while Chapter 6 focuses on the approximation properties of quantized neural networks [Gonon et al., 2023a]. They are disconnected from Chapters 2 to 4, as they do not involve the path-lifting and path-activations.

Chapter 5 shows that so-called Kronecker-sparse matrices can be used to *accelerate the inference of neural networks, while also reducing the memory and energy needed to run them.* Given the centrality of matrix multiplication in neural network operations, Kronecker-sparse matrices, with their structured support, offer promising potential for efficient implementations and sub-quadratic complexity in matrix-vector multiplications. However, their practical benefits in terms of time and energy efficiency was unclear because of a lack of comprehensive numerical results. To fill this gap, the chapter begins with a comprehensive benchmark for Kronecker-sparse matrix multiplication, revealing that significant processing time is spent on memory operations. Then, it introduces a new CUDA kernel designed to minimize memory transfers, achieving a median speed-up of $\times 1.4$ and energy reduction of $\times 0.85$. The chapter concludes by demonstrating the concrete kernel's efficacy in improving neural network inference efficiency.

Chapter 6 examines the approximation properties of quantized neural networks. Quantizing the weights of a neural network is a crucial step in reducing the network's

memory and computational requirements. However, the relationship between the number of bits used in quantization and the expressivity of neural networks is not well understood. This chapter establishes upper bounds on the bit-precision required by *neareast-neighbour uniform quantization* to preserve the approximation rates of neural networks with unconstrained real weights. *This chapter is based on work I completed before becoming familiar with the concept of path-lifting. It primarily utilizes the Lipschitz property in θ to relate the approximation error in an L^p-space of functions to the error introduced by the quantization process in the parameter space. The Lipschitz property used here is not invariant to symmetries. As demonstrated in Chapter 3, the new Lipschitz property in θ, defined in terms of the path-lifting, is always smaller. Although I did not have the opportunity to revisit the results of Chapter 6 in light of the path-lifting, I believe that the new Lipschitz property in θ of Chapter 3 could improve and extend the findings of Chapter 6, making them applicable to more general ReLU networks with max-pooling, skip-connections, and other modern features. This is evoked in Chapter 7.*

Chapter 7 summarizes the contributions and discusses the research directions opened by this thesis. I will first discuss how my extension to general ReLU networks of the path-lifting and path-activations could be used to extend to such general networks some existing results on the identifiability and the training dynamics. I will also numerically challenge the promises of path-norm-based generalization guarantees in practical situations, building upon Chapter 4 that established not only the finest generalization bound of this type but also the most widely applicable one, including in practical situations such as a ResNet trained on ImageNet. Thanks to that, I will identify the limits of the current theory and propose new research directions to overcome these limitations. Finally, I will also discuss perspectives on Chapters 5 and 6, including the design of efficient Kronecker-sparse neural networks, and the use of symmetries in quantization schemes.

1.3 List of Publications

International Journal

- [Gonon et al., 2023a]: *Approximation speed of quantized vs. unquantized ReLU neural networks and beyond*, A. Gonon, N. Brisebarre, E. Riccietti, R. Gribonval, IEEE Transactions on Information Theory, vol. 69, no. 6, pp. 3960-3977, June 2023, doi: 10.1109/TIT.2023.3240360.

International Conference

- [Gonon et al., 2024a]: *A path-norm toolkit for modern networks: consequences, promises and challenges*, A. Gonon, N. Brisebarre, E. Riccietti, R. Gribonval, ICLR 2024.

National Conference

- [Gonon et al., 2023b]: *Can sparsity improve the privacy of neural networks?*, A. Gonon, L. Zheng, C. Lalanne, Q.T. Le, G. Lauga, C. Pouliquen, Gretsi (French National Conference of Signal Processing), August 2023, Grenoble, France.

Preprint

- [Gonon et al., 2024b]: *Path-metrics, pruning and generalization*, A. Gonon, N. Brisebarre, E. Riccietti, R. Gribonval, Preprint.
- [Gonon et al., 2024c]: *Fast inference with Kronecker-sparse matrices*, A. Gonon, L. Zheng, P. Carrivain, Q.T. Le, Preprint.

Chapter 2

Fundamentals of the path-lifting and the path-activations

This chapter is based on results in Gonon et al. [2024a,b].

In the theory of neural networks, the primary objects of study are the functions realized by neural networks and their properties, such as their generalization error and robustness.

The so-called path-lifting and path-activations are promising concepts to theoretically analyze these functions: they have for example been used to derive generalization guarantees [Neyshabur et al., 2015, Kawaguchi et al., 2017, Barron and Klusowski, 2019], identifiability guarantees [Bona-Pellissier et al., 2022, Stock and Gribonval, 2023] and characterizations of properties of the dynamics of training algorithms [Marcotte et al., 2023]. This analysis is made possible thanks to the two main properties of these tools: capturing the piecewise affine structure of the networks, and capturing their symmetries.

Yet, the definitions of the path-lifting and path-activations known before this thesis are severely limited: they only cover simple models unable to combine in a single framework standard ingredients of neural networks such as pooling layers, skip-connections, biases, or even multi-dimensional output [Neyshabur et al., 2015, Kawaguchi et al., 2017, Bona-Pellissier et al., 2022, Stock and Gribonval, 2023]. Thus, the promises of existing theoretical guarantees based on these tools are currently out of reach: they cannot even be tested on standard practical networks.

This chapter adresses the challenge of making these tools fully compatible with practical networks. First, **it formalizes a definition of the path-lifting and path-activations adapted to very generic ReLU networks**, covering any DAG architecture (in particular with skip connections), including in the presence of max/average-pooling (and even more generally k-max-pooling, which extracts the k-th largest coordinate, recovering max-pooling for $k = 1$) and/or biases. This covers a wide variety of modern networks (notably ResNets, VGGs, U-nets, ReLU MobileNets, Inception nets, Alexnet), and recovers previously known definitions of these tools in simpler settings such as layered fully-connected networks (LFCN). Moreover, **this extension is shown to preserve the key properties of the path-lifting and path-activations, i.e., capturing the piecewise affine structure of the networks and their symmetries**.

The outline is as follows.

- Section 2.1 recalls the definition of the simple LFCNs (Definition 2.1.1) and high-

lights the mismatch with the more complex network architectures used today. This section motivates the need for a more general model to accommodate standard neural network ingredients such as max-pooling and skip-connections.

- Section 2.2 contains the first contribution of this chapter: the introduction of a more general model of ReLU networks given by arbitrary directed acyclic graphs (DAG), formalized in Definition 2.2.2, to account for max-pooling, skip-connections and more, operations that are now standard in practical models. I proposed this model in Gonon et al. [2024a]. It unifies and generalizes several models from the literature, including those from Neyshabur et al. [2015], Kawaguchi et al. [2017], DeVore et al. [2021], Bona-Pellissier et al. [2022], Stock and Gribonval [2023].

- Section 2.3 extends the path-lifting and path-activations to this more general DAG model (Definition 2.3.3) [Gonon et al., 2024a]. Then, the challenge is to show that these tools preserve their key properties in this more general setting.

- Section 2.4 proves that the two key properties of the path-lifting and path-activations are preserved in this extended model: they still capture the network's piecewise affine structure (Theorem 2.4.2) and its symmetries (Theorem 2.4.1) [Gonon et al., 2024a,b]. This is absolutely crucial for the theoretical and practical utility of these tools, and will be used many times in the following chapters.

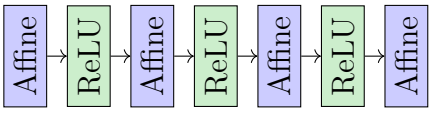
- Section 2.5 concludes this chapter by *a personal interpretation of the path-lifting as defining an intermediate space that takes some of the best of both parameter and function spaces. This perspective, which has implicitly guided my interest in these tools from the beginning, only became explicit to me after completing my works [Gonon et al., 2024a,b] and discussing it with colleagues. While this section is not formal, I still share it with the hope that it will be useful in appreciating the potential of these objects.*

2.1 Layered fully-connected ReLU neural networks (LFCN)

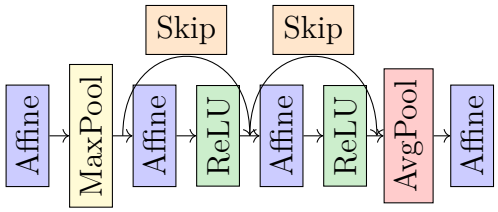
The reader already familiar with LFCNs can skip this section.

This section recalls the definition of layered fully-connected ReLU neural networks (LFCN), one of the simplest types of ReLU networks that corresponds to the alternate composition of affine maps with the ReLU function as depicted on Figure 2.1a. This model is simple enough to allow for theoretical analysis, and it has been the focus of many works in the literature [Neyshabur et al., 2015, Kawaguchi et al., 2017, Barron and Klusowski, 2019, Stock and Gribonval, 2023, Bona-Pellissier et al., 2022, Marcotte et al., 2023]. However, this model does not capture now-standard ingredients of networks widely used in practice such as ResNets [He et al., 2016]. This will be my motivation for the definition of a more general class of neural networks in the next section.

Definition 2.1.1 (Layered fully-connected ReLU neural network (LFCN))**.** *A layered fully-connected ReLU neural network has an architecture, parameters, and these parameters realize a function. These concepts are recalled below.*



(a) Illustration of a layered fully-connected ReLU neural network (LFCN).



(b) Illustration of a network with the same ingredients as a ResNet [He et al., 2016].

Figure 2.1: While LFCN is a simple model well suited for theoretical analysis, it lacks standard ingredients used in practice such as skip connections, max-pooling, and average-pooling.

Architecture. *The architecture of a LFCN is given by an positive integer L and $L+1$ positive integers $d_0, \dots, d_L$.*

Parameters. *The parameters of a LFCN with architecture $(L, d_0, d_1, d_2, \dots, d_L)$ are:*

$$\theta = (M_1, M_2, \dots, M_L, b_1, b_2, \dots, b_L)$$

where $M_\ell \in \mathbb{R}^{d_{\ell+1} \times d_\ell}$ are weight matrices and $b_\ell \in \mathbb{R}^{d_{\ell+1}}$ are bias vectors.

Realization. *The realization of a LFCN with parameters θ is the function $R_\theta : x \in \mathbb{R}^{d_0} \to \mathbb{R}^{d_L}$ defined by:*

$$R_\theta(x) = M_L \,\mathrm{ReLU}(M_{L-1} \,\mathrm{ReLU}(\cdots \mathrm{ReLU}(M_1 x + b_1) \cdots) + b_{L-1}) + b_L \tag{2.1}$$

where I recall (Definition 1.1.1) that $\mathrm{ReLU}(x) = \max(0, x)$ is the ReLU activation function applied coordinate-wise.

Such a neural network is said to have L affine layers and $L+1$ layers of neurons: the input layer of dimension d_0, the output layer of dimension d_L, and $L-1$ hidden layers of dimensions $d_1, \dots, d_{L-1}$ respectively.

I evoked in Chapter 1 that the symmetries and piecewise affine structure of the networks are expected to be useful to better understand their behavior. The path-lifting and path-activations tools have been introduced in the literature to capture these properties and analyze ReLU networks, but only for LFCNs [Neyshabur et al., 2015, Kawaguchi et al., 2017, Barron and Klusowski, 2019, Stock and Gribonval, 2023, Bona-Pellissier et al., 2022, Marcotte et al., 2023]. However, comparing Figure 2.1a and Figure 2.1b, we see that LFCNs do not capture networks like ResNets, widely used in practice. This calls for defining a more general model, which is the subject of the next section.

2.2 Defining a comprehensive neural network model

As illustrated in Figure 2.1b, the compositional structure of LFCNs between successive layers of neurons no longer holds in practical networks. Indeed, skip-connections (orange blocks labeled "Skip") disrupt this layer-wise compositional flow. Figure 2.1b rather involves a compositional structure at the neuron level: each individual neuron is connected to subsequent neurons in a directed manner. This can be captured by a directed acyclic graph (DAG) where the vertices represent neurons and the edges represent connections between neurons. This is the model I propose in this section, that generalizes and unifies several models from the literature, including those from Neyshabur et al. [2015], Kawaguchi et al. [2017], DeVore et al. [2021], Bona-Pellissier et al. [2022], Stock and Gribonval [2023].

Moreover, I aim to cover not only ReLU neurons but also max-pooling and average-pooling neurons as in Figure 2.1b, since these operations are now standard in practice. Thus, I define the model a DAG with an activation function for each neuron: ReLU, max-pooling (and even k-max-pooling, with max-pooling corresponding to $k = 1$), or the identity for average-pooling. This is illustrated in Figure 2.2. *While I now need to formalize this, I invite readers to skip the rest of this section and refer back to the details as needed.*

Let me start by defining k-max-pooling, which captures max-pooling for $k = 1$.

Definition 2.2.1 (k-max-pooling activation function)**.** *The k-max-pooling function k-$\texttt{pool}(x) := x_{(k)}$ returns the k-th largest coordinate of $x \in \mathbb{R}^d$.*

I now define the model of DAG ReLU neural network (simply called DAG network thereafter). The reader can refer to Figure 2.2 and Figure 2.3 for illustrations of the definition.

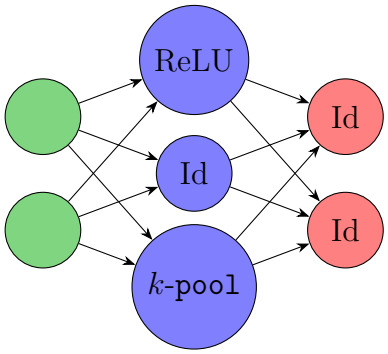
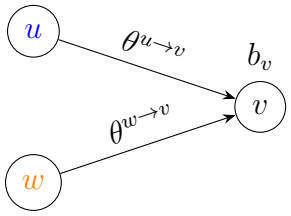


Figure 2.2: Example of a DAG ReLU Neural Network Architecture (Definition 2.2.2). Input neurons are in green, output neurons in red and hidden neurons in blue. Each neuron has an activation function, except for the input neurons. By definition, the output neurons are enforced to be identity neurons, i.e., neurons with the identity as activation function.

Definition 2.2.2 (DAG ReLU neural network [Gonon et al., 2024a])**.** *Consider a Directed Acyclic Graph (DAG) $G = (N, E)$ with edges E, and vertices N called neurons. For a neuron v, the sets* $\mathrm{ant}(v), \mathrm{suc}(v)$ *of antecedents and successors of v are $ant(v) := \{u \in$*



and $v(\theta, x) = \rho_v(b_v + u(\theta, x)\theta^{u\to v} + w(\theta, x)\theta^{w\to v})$

Figure 2.3: Detailed representation of a single neuron v (Definition 2.2.2) with antecedents u and w, a bias b_v, and an activation function $\rho_v \in \{\mathrm{ReLU}, \mathrm{id}\}$.

$N, u \to v \in E\}$, $\mathrm{suc}(v) := \{u \in N, v \to u \in E\}$. *Neurons with no antecedents (resp. no successors) are called input (resp. output) neurons, and their set is denoted* N_{in} *(resp.* N_{out}*). Neurons in* $H := N \setminus (N_{in} \cup N_{out})$ *are called hidden neurons. Input and output dimensions are respectively* $d_{in} := |N_{in}|$ *and* $d_{out} := |N_{out}|$.

• *A* **DAG ReLU neural network architecture***, or simply* ***DAG network****, is a tuple* $(G, (\rho_v)_{v\in N\setminus N_{in}})$ *composed of a DAG* $G = (N, E)$ *with attributes* $\rho_v \in \{\mathrm{id}, \mathrm{ReLU}\} \cup \{k\texttt{-pool}, k \in \mathbb{N}_{>0}\}$ *for* $v \in N \setminus (N_{out} \cup N_{in})$ *and* $\rho_v = \mathrm{id}$ *for* $v \in N_{out}$*. We will again denote the tuple* $(G, (\rho_v)_{v\in N\setminus N_{in}})$ *by* G*, and it will be clear from context whether the results depend only on* $G = (N, E)$ *or also on its attributes. Define* $N_\rho := \{v \in N, \rho_v = \rho\}$ *for an activation* ρ*, and* $N_{*\texttt{-pool}} := \cup_{k\in\mathbb{N}_{>0}} N_{k\texttt{-pool}}$*. A neuron in* $N_{*\texttt{-pool}}$ *is called a* $*$*-max-pooling neuron. For* $v \in N_{*\texttt{-pool}}$*, its kernel size is defined as being* $|\mathrm{ant}(v)|$.

• **Parameters** *associated with this architecture are vectors*[1] $\theta \in \mathbb{R}^G := \mathbb{R}^{E\cup N\setminus N_{in}}$*. We call bias* $b_v := \theta_v$ *the coordinate associated with a neuron* v *(input neurons have no bias), and denote* $\theta^{u\to v}$ *the weight associated with an edge* $u \to v \in E$*. We will often denote* $\theta^{\to v} := (\theta^{u\to v})_{u\in\mathrm{ant}(v)}$ *and* $\theta^{v\to} := (\theta^{u\to v})_{u\in\mathrm{suc}(v)}$.

• *The* **pre-activation** *of a neuron* $v \in N \setminus N_{in}$ *is the function* v_{pre} *defined as*

$$v_{pre}(\theta, x) := \begin{cases} b_v + \sum_{u\in\mathrm{ant}(v)} u(\theta, x)\theta^{u\to v} & \text{if } \rho_v = \mathrm{ReLU} \text{ or } \rho_v = \mathrm{id}, \\ (b_v + u(\theta, x)\theta^{u\to v})_{u\in\mathrm{ant}(v)} & \text{if } \rho_v = k\texttt{-pool}. \end{cases} \tag{2.2}$$

• *The* **realization** *of a neural network with parameters* $\theta \in \mathbb{R}^G$ *is the function* $R^G_\theta : \mathbb{R}^{N_{in}} \to \mathbb{R}^{N_{out}}$ *(simply denoted* R_θ *when* G *is clear from the context) defined for every input* $x \in \mathbb{R}^{N_{in}}$ *as*

$$R_\theta(x) := (v(\theta, x))_{v\in N_{out}},$$

where I use the same symbol v *to denote a neuron* $v \in N$ *and the associated function* $v(\theta, x)$*, defined as* $v(\theta, x) := x_v$ *for an input neuron* v*, and defined by induction otherwise*

$$v(\theta, x) := \rho_v(v_{pre}(\theta, x)). \tag{2.3}$$

As promised, let me now show that the model of Definition 2.2.2 covers many ingredients widely used in practice, with the exception of the attention mechanism. Thanks to that, many practical networks are covered by Definition 2.2.2: ResNets, VGGs, U-nets, ReLU MobileNets, Inception nets, Alexnet etc.

- **Max-pooling:** set $\rho_v = k\texttt{-pool}$ for $k = 1$, $b_v = 0$ and $\theta^{u\to v} = 1$ for every $u \in \mathrm{ant}(v)$.

[1] For an index set I, denote $\mathbb{R}^I = \{(\theta_i)_{i\in I}, \theta_i \in \mathbb{R}\}$.

Motivation. This operation selects the maximum value from a set of scalar inputs, effectively down-sampling the input representation and reducing its dimensionality while retaining important information [LeCun et al., 1998].

- **Average-pooling:** set $\rho_v = \mathrm{id}$, $b_v = 0$ and $\theta^{u\to v} = 1/|\operatorname{ant}(v)|$ for every $u \in \operatorname{ant}(v)$.

  Motivation. This operation computes the average value of a set of scalar inputs, similarly down-sampling the input representation but by averaging, which can provide smoother results [LeCun et al., 1998].

- **Batch normalization:** set $\rho_v = \mathrm{id}$ and weights accordingly. Batch normalization layers only differ from standard affine layers by the way their parameters are updated during training.

  Motivation. A batch is a subset of the training data processed together to update at once the parameters θ. Regrouping the training data in batches allows for parallel computing, hence faster training. Batch normalization is a technique used to improve the training of deep neural networks by stabilizing the distributions of the outputs of some operations performed in batch [Ioffe and Szegedy, 2015]. During training, it normalizes over a batch the output of a previous operation by subtracting the batch mean and dividing by the batch standard deviation. This normalization is followed by a scaling and shifting operation controlled by learnable parameters. During inference, the mean and standard deviation are fixed (learned during training) and this simply becomes a linear layer with only parameters the scaling α and the bias β:

  $$\mathrm{BatchNorm}(x) = \alpha\frac{x-\mu}{\sigma} + \beta$$

  where μ and σ are the mean and standard deviation learned during training, and x is the input.

- **Fully-connected layer:** via the DAG structure, two sets of neurons can be fully connected by adding connections between all neurons of the first set to all neurons of the second set.

  Motivation. This ensures that LFCNs are a special case of this model.

- **Convolutional layer:** consider them as (doubly) circulant/Toeplitz fully connected layers.

  Motivation. This corresponds to fully-connected layers but with additional constraint on the affine map to ensure invariance to translation of the input. This is particularly interesting to process spatial data such as images since relevant features are often invariant to translation. Moreover, this constraint reduces the number of parameters to learn as many coefficients of the weight matrix are the same.

- **GroupSort/Top-k operator:** use the DAG structure and $*$-max-pooling neurons [Anil et al., 2019, Sander et al., 2023].

  Motivation. Finding the top-k larger components of a vector, or sorting the components can be relevant to select the most significant features from a set.

- **Skip connections:** via the DAG structure, the outputs of any past layers can be added to the pre-activation of any neuron by adding connections from these layers to the given neuron.

  Motivation. These connections allow the output of a layer to be added to a subsequent layer, facilitating the training of deeper networks by mitigating the so-called vanishing gradient problem He et al. [2016].

We now got a model that captures ReLU networks widely used in practice and the challenge is now to have tools to theoretically analyze it. This is the object of the next section.

2.3 Path-lifting and path-activations: formal definitions

As discussed in Chapter 1, the symmetries and piecewise affine structure of ReLU networks are key to understanding their behavior.

- **Symmetries and non-invariant bounds.** I evoked in Chapter 1 that a non-invariant bound

  $$\text{Interesting Quantity}(R_\theta) \leqslant C\|\theta\|$$

  can be vacuous depending on the luck we have with the scaling of the parameters θ. This is because the left-hand side is invariant under rescaling ($R_\theta = R_{\lambda \diamond \theta}$ for any rescaling vector λ, the symmetries are formally defined later in Definition 2.4.1), while the right-hand side can be arbitrarily large ($\|\lambda \diamond \theta\| \to \infty$ as $\|\lambda\| \to \infty$). This motivates looking for representations of the parameters $\Phi(\theta)$ that are invariant to symmetries, and to establish bounds that depend on $\|\Phi(\theta)\|$ rather than $\|\theta\|$.

- **Symmetries and compression.** I also evoked in Chapter 1 that a gold-standard compression method is to prune (set to zero) some weights directly based on their magnitude. However, I will show that this method is not robust to parameters rescaling: the accuracy of this compression method can significantly drop depending on the scaling of the parameters. This again motivates looking for representations of the parameters $\Phi(\theta)$ that are invariant to symmetries, and to establish compression methods that depend on $\Phi(\theta)$ rather than directly on θ.

- **Piecewise affine structure.** Finally, Chapter 1 also discusses how the piecewise affine structure of ReLU networks is expected to influence their expressivity (e.g., via the number of affine regions) or their robustness (e.g., via the Lipschitz constant on each affine region).

This motivates the study of ReLU networks through the lens of their symmetries and piecewise affine structure. Tools to do so have been developed in the literature, *but only for LFCNs*. I now extend these tools to the *general DAG network model* defined in Section 2.2. In the next sections, I will show that these extended tools preserve their fundamental properties:

- $\Phi(\theta)$ is a vector, whose entries are monomial functions of the coordinates of θ;

- $A(\theta, x)$ is a binary matrix, and is a piecewise constant function of (θ, x),
- both $\Phi(\theta)$ and $A(\theta, x)$ are invariant under neuron-wise rescalings of θ that leave invariant the function R_θ,
- the network output is a simple function of these two objects:

$$R_\theta(x) = \left\langle \Phi(\theta), A(\theta, x) \begin{pmatrix} x \\ 1 \end{pmatrix} \right\rangle \tag{2.4}$$

in the scalar-valued case, and with a similar simple formula in the vector-valued case (Equation (2.6) to come).

I will now describe in words the various objects at play before giving the formal definitions. Hopefully, these informal descriptions are enough and I invite the readers to solely focus on them, skip the formal definitions for now, and refer back to them as needed.

In the terms "path-lifting" and "path-activations", a path refers to a sequence of connected neurons in the DAG that ends at an output neuron and starts at either an input or a hidden neuron.

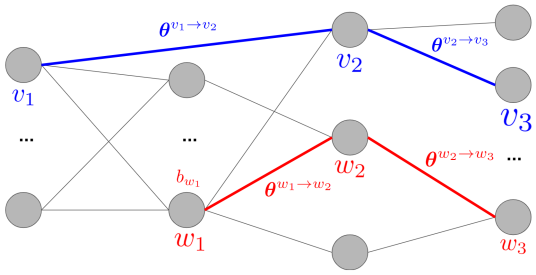
For parameters θ, its path-lifting $\Phi(\theta)$ is a vector indexed by all the paths. Note that even if its ambient dimension is combinatorial (number of paths), I will show that norms of the path-lifting can be computed very efficiently. The coordinate $\Phi_p(\theta)$ along a given path p is the product of the weights along that path, including the bias when starting from a hidden neuron. See Figure 2.4 for an illustration.

The path-activations capture which neurons and edges transmit information taken into account in the network's output or not. A ReLU neuron is said active if its output is non-zero. An edge $u \to v$ going to a max-pooling neuron v is said active only if the maximum comes from the neuron u. Other neurons and edges are declared always active.

Formal definitions. Let me now turn to the formal definitions of the path-lifting and path-activations. They both rely on the definitions of the paths of a DAG G.

Definition 2.3.1 (Paths and depth in a DAG [Gonon et al., 2024a])**.** *Consider a DAG $G = (N, E)$ as in Definition 2.2.2. A path of G is any sequence of neurons $v_0, \dots, v_d$ such that each $v_i \to v_{i+1}$ is an edge in G. Such a path is denoted $p = v_0 \to \dots \to v_d$. This includes paths reduced to a single neuron $v \in N$, denoted $p = v$. The* length *of a path $p = v_0 \to \dots \to v_d$ is $\texttt{length}(p) = d$ (the number of edges). We will denote $p_\ell := v_\ell$ the ℓ-th neuron for a general $\ell \in \{0, \dots, \texttt{length}(p)\}$ and use the shorthand $p_{\texttt{end}} = v_{\texttt{length}(p)}$ for the last neuron. The* depth of the graph *G is the maximum length over all of its paths. If $v_{d+1} \in \mathrm{suc}(p_{\texttt{end}})$ then $p \to v_{d+1}$ denotes the path $v_0 \to \dots \to v_d \to v_{d+1}$. In the sequel, we will often restrict to the set of paths* ending at an output neuron*, a set denoted by $\mathcal{P}^G$ (or simply $\mathcal{P}$), and will often say simply path to refer to a path $p \in \mathcal{P}$ when the context is clear.*

Definition 2.3.2 (Sub-graph ending at a given neuron)**.** *Given a neuron v of a DAG G, denote by $G^{\to v}$ the graph deduced from G by keeping only the largest subgraph with the same inputs as G and with v as a single output. This consists of removing all the neurons u that cannot reach v by following the edges of G, as well as all the incoming and outcoming edges of such neurons u. We will use the shorthand $\mathcal{P}^{\to v} := \mathcal{P}^{G^{\to v}}$ to denote the set of paths of $G^{\to v}$, which correspond to the paths in G ending at v.*



$$A(\theta, x) = \begin{array}{c} \\ \mathcal{P}_I \left\{ \begin{array}{c} p \\ \\ \\ \\ p' \end{array} \right. \end{array} \begin{array}{c} \overbrace{\qquad\qquad\qquad\qquad}^{N_{\text{in}}} \\ v_1 \\ \cdots \\ 0 \ \ldots \ 0 \ \ a_p(\theta, x) \ \ 0 \ \ldots \ 0 \\ \cdots \\ \\ 0 \end{array} \begin{array}{c} b \\ \\ \\ 0 \\ \\ \vdots \\ a_{p'}(\theta, x) \\ \vdots \end{array}$$

Figure 2.4: The coordinate of the path-lifting Φ associated with the path $p = v_1 \to v_2 \to v_3$ is $\Phi_p(\theta) = \theta^{v_1 \to v_2} \theta^{v_2 \to v_3}$ since it starts from an input neuron (Definition 2.3.3). While the path $p' = w_1 \to w_2 \to w_3$ starts from a hidden neuron (in $N \setminus (N_{\text{in}} \cup N_{\text{out}})$), so there is also the bias of w_1 to take into account: $\Phi_{p'}(\theta) = b_{w_1} \theta^{w_1 \to w_2} \theta^{w_2 \to w_3}$. As specified in Definition 2.3.3, the columns of the path-activation matrix A are indexed by $N_{\text{in}} \cup \{b\}$ and its rows are indexed by $\mathcal{P} = \mathcal{P}_I \cup \mathcal{P}_H$, with $\mathcal{P}_I$ the set of paths in $\mathcal{P}$ starting from an input neuron, and $\mathcal{P}_H$ the set of paths starting from a hidden neuron.

It is now time to define the path-lifting and path-activations. The reader can refer to Figure 2.4 for an illustration.

Definition 2.3.3 (Path-lifting and path-activations)**.** *Consider a ReLU neural network architecture G as in Definition 2.2.2 and parameters $\theta \in \mathbb{R}^G$ associated with G. For $p \in \mathcal{P}$, define*[2]

$$\Phi_p(\theta) := \begin{cases} \prod_{\ell=1}^{\texttt{length}(p)} \theta^{v_{\ell-1}\to v_\ell} & \text{if } p_0 \in N_{in}, \\ b_{p_0} \prod_{\ell=1}^{\texttt{length}(p)} \theta^{v_{\ell-1}\to v_\ell} & \text{otherwise,} \end{cases}$$

where an empty product is equal to 1 by convention. The path-lifting $\Phi^G(\theta)$ of θ is

$$\Phi^G(\theta) := (\Phi_p(\theta))_{p\in\mathcal{P}^G}.$$

This is often denoted Φ when the graph G is clear from the context. We will use the shorthand $\Phi^{\to v} := \Phi^{G^{\to v}}$ to denote the path-lifting associated with $G^{\to v}$ (Definition 2.3.2). We will denote by $\Phi^I(\theta)$ (resp. $\Phi^H(\theta)$) the sub-vector of $\Phi(\theta)$ corresponding to the coordinates associated with paths starting from an input (resp. hidden) neuron. Therefore, $\Phi(\theta)$ is the concatenation of $\Phi^I(\theta)$ and $\Phi^H(\theta)$.

Consider an input x of G. The activation of an edge $u \to v$ *on (θ, x) is defined to be $a_{u\to v}(\theta, x) := 1$ when v is an identity neuron; $a_{u\to v}(\theta, x) := \mathbb{1}_{v(\theta,x)>0}$ when v is a ReLU neuron; and when v is a k-max-pooling neuron, define $a_{u\to v}(\theta, x) := 1$ if the neuron u is the first in* $\mathrm{ant}(v)$ *in lexicographic order to satisfy $u(\theta, x) := k\text{-}\texttt{pool}\left((b_v + w(\theta, x)\theta^{w\to v})_{w\in\mathrm{ant}(v)}\right)$ and $a_{u\to v}(\theta, x) := 0$ otherwise. The activation of a* neuron v *on (θ, x) is defined to be $a_v(\theta, x) := 1$ if v is an input neuron, an identity neuron, or a k-max-pooling neuron, and $a_v(\theta, x) := \mathbb{1}_{v(\theta,x)>0}$ if v is a ReLU neuron. We then define the activation of a* path $p \in \mathcal{P}$ *with respect to input x and parameters θ as: $a_p(\theta, x) := a_{p_0}(\theta, x) \prod_{\ell=1}^{\texttt{length}(p)} a_{v_{\ell-1}\to v_\ell}(\theta, x)$ (with an empty product set to one by convention). Consider a new symbol $v_{\texttt{bias}}$ that is not used for denoting neurons. The path-activations matrix $A(\theta, x)$ is defined as the matrix in $\mathbb{R}^{\mathcal{P}\times(N_{in}\cup\{v_{\texttt{bias}}\})}$ such that for any path $p \in \mathcal{P}$ and neuron $u \in N_{in} \cup \{v_{\texttt{bias}}\}$*

$$(A(\theta, x))_{p,u} := \begin{cases} a_p(\theta, x)\mathbb{1}_{p_0=u} & \text{if } u \in N_{in}, \\ a_p(\theta, x) & \text{otherwise when } u = v_{\texttt{bias}}. \end{cases}$$

Positioning. Previous definitions in the literature were given for simpler models (no k-max-pooling even for a single given k, no skip connections, no biases, one-dimensional output and/or layered network) [Kawaguchi et al., 2017, Bona-Pellissier et al., 2022, Stock and Gribonval, 2023]. Definition 2.3.3 extends these definitions to the general model of Definition 2.2.2, as it recovers the previous ones in simpler settings. The main novelty is essentially to properly define the path-activations $A(\theta, x)$ in the presence of $*$-max-pooling neurons: when going through a k-max-pooling neuron, a path stays active only if the previous neuron of the path is the first in lexicographic order to be the k-th largest input of this pooling neuron.

[2] We do not only consider paths starting at input neurons, but also the ones starting at *hidden* neurons, which will reveal to be crucial to take into account the biases.

On the name "path-lifting" and "path-activations". The mapping Φ was initially called path-embedding in Stock and Gribonval [2023] but since it is not injective [Stock and Gribonval, 2023, Remark 2], I choose to call it path-lifting just as in Bona-Pellissier et al. [2022]. Note however that this is not a lifting in the sense of category theory as it does not factorize the mapping $\theta \mapsto R_\theta$ (for that, we should extend the mapping with the signs $\theta \to (\mathrm{sgn}(\theta), \Phi(\theta))$ [Stock and Gribonval, 2023, Theorem 1]). I still choose the name lifting for the sake of brevity. The path-activations A were called activations in Stock and Gribonval [2023]. I choose to call them path-activations to avoid confusion with the usual activations of the neurons.

2.4 Main properties of the path-lifting and path-activations

The challenge is now to show that the path-lifting and the path-activations extended to the general model of Definition 2.2.2 have the same fundamental properties as in the case of simple LFCNs: they capture in some sense the neuron-wise rescaling symmetries, as well as the piecewise affine structure of the model. This is tackled respectively in Section 2.4.1 and Section 2.4.2.

2.4.1 Capturing the rescaling symmetries

The symmetries are transformations of the parameters θ that leave the function R_θ unchanged. Let me introduce these transformations for a general DAG network.

Definition 2.4.1 (Neuron-wise rescaling symmetry)**.** *Consider a DAG network G as in Definition 2.2.2. Recall that H is the set of hidden neurons (Definition 2.2.2), and R^G is the set of parameters. Consider the group $\mathbb{R}^H_{>0}$ with pointwise multiplication and neutral element $\mathbf{1}$ the vector full of ones. The rescaling symmetries can almost be seen as a group action of $\mathbb{R}^H_{>0}$ on $\mathbb{R}^G$ but with a twist: the action is not commutative. We will denote*[3] *by $\lambda \diamond \theta$ the action of $\lambda \in \mathbb{R}^H_{>0}$ on $\theta \in \mathbb{R}^G$ and define it as the composition of several* elementary operations *$\lambda_v \diamond_v \theta$ for $v \in H$:*

$$(\lambda_v \diamond_v \theta)^{\to v} := \lambda_v \theta^{\to v}, \quad (\lambda_v \diamond_v \theta)_v := \lambda_v \theta_v, \quad (\lambda_v \diamond_v \theta)^{v\to} := \frac{1}{\lambda_v} \theta^{v\to} \tag{2.5}$$

Since the elementary operations $\diamond_v$ do not commute for different v's, we have to fix a convention to compose them. We choose the convention that the composition of the elementary operations is done in the order of the indices of the neurons in H:

$$\lambda \diamond \theta := \lambda_{v_{|H|}} \diamond_{v_{|H|}} (\dots \lambda_{v_1} \diamond_{v_1} \theta)\,.$$

This convention has no impact in this thesis: the results would be the same with any other convention.

We denote by $\theta \sim_R \theta'$ the fact that there exists a $\lambda \in \mathbb{R}^H_{>0}$ such that $\theta' = \lambda \diamond \theta$. This is an equivalence relation on $\mathbb{R}^G$ and we say that θ and θ' are rescaling-equivalent*.*

[3] To denote this action, I choose the symbol $\diamond$ rather than the more conventional one $\cdot$ to avoid it to be mistaken with the scalar multiplication.

I will prove that within a rescaling-equivalence class, all the parameters θ define the same function R_θ, path-lifting $\Phi(\theta)$ and path-activations $A(\theta, x)$. In the special case where θ has a so-called *input-dead neuron* v, the notion of equivalence can be relaxed to include more parameters that still correspond to the same function R_θ, path-lifting $\Phi(\theta)$, and path-activations $A(\theta, x)$.

Definition 2.4.2 (Dead neuron)**.** *Consider a DAG network G as in Definition 2.2.2 and parameters $\theta \in \mathbb{R}^G$. A neuron v is said to be* input-dead *for θ if $\theta^{\to v} = b_v = 0$. We denote by $\texttt{In} - \texttt{Dead}(\theta)$ the set of input-dead* hidden *neurons for θ.*

Definition 2.4.3 (Weak neuron-wise rescaling symmetry)**.** *Consider a DAG network G as in Definition 2.2.2 and parameters $\theta \in \mathbb{R}^G$. Consider all the parameters θ' that can be reached such a θ by finite composition of the following operations in any given order:*

- *neuron-wise rescaling $\lambda \diamond \theta$ for some $\lambda \in \mathbb{R}^H_{>0}$,*
- *and setting the outgoing weights $\theta^{v\to}$ arbitrarily for some $v \in \texttt{In} - \texttt{Dead}(\theta)$.*

We say that θ and θ' are weakly rescaling-equivalent *and we denote $\theta \sim_{WR} \theta'$.*

I could also haved introduced the notion of output-dead neurons v, satisfying $\theta^{v\to} = 0$, and relax even more the equivalence class by allowing to set the incoming weights $\theta^{\to v}$ arbitrarily for all output-dead neurons v. While this operation would preserve the function R_θ and the path-lifting $\Phi(\theta)$, this would *not* preserve the path-activations $A(\theta, x)$. Indeed, we could have the pre-activation of an output-dead ReLU neuron strictly positive, hence active, and then by setting to zero the incoming weights, it becomes inactive. Moreover, we will only encounter input-dead neurons in the sequel. For these reasons, I will not further discuss the notion of output-dead neurons.

Rescaling transformations are indeed *symmetries* as shown in the next lemma. The reason for this is that the ReLU and $*$-max-pooling activation functions ρ are positively homogeneous ($\rho(\lambda x) = \lambda\rho(x)$ for $\lambda > 0$).

Lemma 2.4.1. *Consider a DAG network as in Definition 2.2.2. If parameters θ, θ' are such that $\theta \sim_{WR} \theta'$ (Definition 2.4.3) then $R_\theta = R_{\theta'}$.*

Proof. This is already known for *rescaling-equivalent parameters in the special case of LFCNs*, see, e.g., [Stock and Gribonval, 2023]. For *weakly* rescaling-equivalent parameters *and general DAG networks* (Definition 2.2.2), this is a direct consequence of two theorems proved below: Theorem 2.4.1 and Theorem 2.4.2. □

In the special case of *LFCNs*, it is also known that rescaling-equivalent parameters have the same path-lifting and path-activations, see, e.g., [Stock and Gribonval, 2023]. *As promised, the next theorem shows that our extended path-lifting and path-activations preserve this property*, even for *weakly* rescaling-equivalent parameters.

Theorem 2.4.1. *Consider parameters θ, θ' of a DAG network (Definition 2.2.2) such that $\theta \sim_{WR} \theta'$ (Definition 2.4.3). They have the same path-lifting and path-activations: $\Phi(\theta) = \Phi(\theta')$ and $A(\theta, x) = A(\theta', x)$ for every x.*

To avoid breaking the reading flow, the proof is postponed to Section 2.4.3.

2.4.2 Capturing the piecewise affine structure

The second equally fundamental property of the path-lifting and path-activations is the reparameterization of the model in terms of Φ and A.

Theorem 2.4.2 ([Gonon et al., 2024a])**.** *Consider a DAG ReLU network as in Definition 2.2.2. For every neuron v, every input x and every parameters θ, it holds:*

$$v(\theta, x) = \left\langle \Phi^{\to v}(\theta), A^{\to v}(\theta, x) \begin{pmatrix} x \\ 1 \end{pmatrix} \right\rangle. \tag{2.6}$$

The proof is an induction on the size of the graph, and is deferred to Section 2.4.3.

In the scalar-valued case with single output neuron v, corresponding to $R_\theta(x) = v(\theta, x)$, Theorem 2.4.2 shows that we have the simple formula:

$$R_\theta(x) = \left\langle \Phi(\theta), A(\theta, x) \begin{pmatrix} x \\ 1 \end{pmatrix} \right\rangle.$$

In the vector-valued case, Theorem 2.4.2 says that the same formula holds for every output neuron v by considering the sub-vector of Φ and the sub-matrix of A associated with the maximal subgraph $G^{\to v}$ of G ending at v. This equation is not only fundamental to understand the structure of the function R_θ in terms of Φ and A, but also of practical interest as it will allow us to compute (mixed) ℓ^q-norms of $\Phi(\theta)$, called path-norms, in only one forward-pass of the model. *This is a blessing at it makes all the theoretical bounds of this thesis easily computable in practice, despite the combinatorial dimensions of Φ and A.*

Let me now explain why Equation (2.6), reproduced below for convenience, implies that Φ and A capture **the piecewise affine structure of the function $(x, \theta) \mapsto R_\theta(x)$ in x and the piecewise polynomial structure in θ.**

$$v(\theta, x) = \left\langle \Phi^{\to v}(\theta), A^{\to v}(\theta, x) \begin{pmatrix} x \\ 1 \end{pmatrix} \right\rangle.$$

If there were no path-activations A in this equation, this would say that the output of neuron v is an affine function of x, with the affine coefficients stored in $\Phi^{\to v}(\theta)$.

Of course, ReLU neurons are not affine in x, and the matrix A restores the non-linearity in x. It is binary and piecewise constant, meaning that we can divide the set of (θ, x) into a finite number of regions where A is constant [Arora et al., 2017].

For a fixed θ, the set of x such that $A(\theta, x)$ is constant has boundaries defined by zeros of affine equations, so that it is a polytope [Arora et al., 2017]. For a fixed x, the set of θ such that $A(\theta, x)$ is constant has boundaries defined by zeros of polynomial sets as a change of a coefficient in A is either due to a change in the sign of a ReLU neuron output or a change in the ordering of the inputs of a k-max-pooling neuron and the result follows from the fact that all the neuron functions are locally polynomial in the coordinates of θ (Equation (2.6)).

On each of the region in (θ, x) where $A(\theta, x)$ is constant, Equation (2.6) says that the output of every neuron v is affine in x, with the affine coefficients given by $A(\theta, x)^T \Phi^{\to v}(\theta)$: the matrix A filters the coordinates of $\Phi^{\to v}(\theta)$ to select the ones relevant on the current region.

	Key properties
Parameters θ	• parameters of the neural network (weights of the edges and the biases of the neurons) • directly updated when training the function R_θ
Path-lifting $\Phi(\theta)$	• affine coefficients of the function $R_\theta(x)$ on the active region • invariant under neuron-wise rescaling symmetries of R_θ • polynomial in the coordinates of θ
Path-activations $A(\theta, x)$	• capture the regions where $x \mapsto R_\theta(x)$ is affine and where $\theta \mapsto R_\theta(x)$ is polynomial • invariant under neuron-wise rescaling symmetries of R_θ • binary • piecewise constant in (θ, x)
Function R_θ	• piecewise affine function of x • piecewise polynomial in the coordinates of θ • invariant under neuron-wise rescaling symmetries of θ

Table 2.1: Properties of the objects $\theta, \Phi(\theta)$, $A(\theta, x)$, R_θ.

This shows that Φ and A encode the piecewise affine structure of the function R_θ: Φ holds all the affine coefficients, and A acts as a region-aware filter that knows which coefficients to select in each region. This will be key to analyze ReLU neural network functions. I summarize the properties of the path-lifting and the path-activations in Table 2.1.

Related works. *[Kawaguchi et al., 2017]:* Equation (2.6) is stated in the specific case of Kawaguchi et al. [2017, Section 5.1] (as an explicit sum rather than an inner product), without proof since the objects are not explicitly defined in Kawaguchi et al. [2017].

[Stock and Gribonval, 2023, Bona-Pellissier et al., 2022]: Corollary 3 of Stock and Gribonval [2023] proves that Equation (2.6) holds *in this specific case* of fully-connected neural networks. Definition 2.3.3 and Equation (2.6) generalize the latter to an arbitrary DAG with $*$-max-pooling or identity neurons (allowing in particular for skip connections, max-pooling and average-pooling).

Rest of the literature: The rest of the works I am aware of only define and consider the *norm* of the path-lifting, but not the lifting *itself*. The most general setting is the one of Neyshabur et al. [2015] with a general DAG, *but without max or identity neurons, nor biases*. Our formal definition of the path-lifting and the path-activations makes notations arguably lighter since Equation (2.6) is no longer written with an explicit sum over all paths, with explicit product of weights along each path etc.

2.4.3 Proofs of the main properties

Let me start with the proof that the path-lifting and path-activations are the same for weakly rescaling-equivalent parameters.

Proof of Theorem 2.4.1. By definition of weakly-rescaling equivalent parameters (Definition 2.4.3), it is enough to check that the path-lifting and path-activations of given parameters θ are invariant under the two basic operations:

- neuron-wise rescaling $\lambda \diamond \theta$ for some $\lambda \in \mathbb{R}^H_{>0}$,
- and setting $\theta^{v\to}$ arbitrarily for a given $v \in \texttt{In} - \texttt{Dead}(\theta)$.

Invariance to neuron-wise rescaling. Define $\theta' = \lambda \diamond \theta$ for some $\lambda \in \mathbb{R}^H_{>0}$. We show that $\Phi(\theta) = \Phi(\theta')$ and $A(\theta, x) = A(\theta', x)$ for every x. We can restrict to the case where a single hidden neuron v is rescaled by $\lambda_v > 0$, while all the other hidden neurons are unchanged, i.e., $\lambda_u = 1$ for $u \neq v$. Indeed, rescaling by a general vector $\lambda \in \mathbb{R}^H_{>0}$ is defined as iteratively rescaling one neuron at a time (Definition 2.4.1).

Consider a path $p = v_0 \to \ldots \to v_d \in \mathcal{P}$ and let me prove that the coordinate $\Phi_p(\theta)$ of $\Phi(\theta)$ is preserved. Since v_d must be an output neuron (Definition 2.3.1), $v \neq v_d$ and we have three cases to consider:

- **Case 1: v is not in the path p.** In this case, $\Phi_p(\theta)$ depends only on the coordinates of θ that are unaffected by the rescaling, so $\Phi_p(\theta) = \Phi_p(\lambda \diamond \theta)$ trivially.
- **Case 2: $v = v_0$.** Since $v_0 = v \notin N_{\text{in}}$, we have

  $$\Phi_p(\theta) = b_{v_0} \prod_{\ell=1}^{d} \theta^{v_{\ell-1}\to v_\ell}.$$

  Rescaling affects b_{v_0} and $\theta^{v_0\to v_1}$ as follows:

  $$\Phi_p(\lambda \diamond \theta) = (\lambda_v b_v) \left(\frac{1}{\lambda_v}\theta^{v\to v_1}\right) \prod_{\ell=2}^{d} \theta^{v_{\ell-1}\to v_\ell} = \Phi_p(\theta).$$

  Here, b_v is multiplied by λ_v and $\theta^{v\to v_1}$ is divided by λ_v, while other weights remain unchanged.
- **Case 3: $v = v_\ell$ with $0 < \ell < d$.** Among the weights appearing in $\Phi_p(\theta)$, only the weights $\theta^{v_{\ell-1}\to v_\ell}$ and $\theta^{v_\ell\to v_{\ell+1}}$ are affected:

  $$\theta^{v_{\ell-1}\to v_\ell}\theta^{v_\ell\to v_{\ell+1}} = (\lambda_v \theta^{v_{\ell-1}\to v_\ell}) \left(\frac{1}{\lambda_v}\theta^{v_\ell\to v_{\ell+1}}\right).$$

  The product is invariant under the rescaling, hence $\Phi_p(\theta) = \Phi_p(\lambda \diamond \theta)$.

Next, we show that the path-activations are preserved. Since the map $(\theta, x) \mapsto u(\theta, x)$ is not impacted by the considered rescaling for $u \neq v$, it is enough to check that the activation of v is preserved, as well as the activations of the edges $u \to v$ if v is a k-max-pooling neuron, and $v \to u$ otherwise.

- **If v is an identity neuron**, $a_v(\theta, x) = 1$ for every θ and x, so $a_v(\theta, x) = a_v(\lambda \diamond \theta, x)$. The same applies to edge activations $a_{v\to u}(\theta, x)$.
- **If v is a ReLU neuron**, $a_v(\theta, x) = \mathbb{1}_{v(\theta,x)>0}$. Using the positive homogeneity of the ReLU function, rescaling gives:

  $$v(\lambda \diamond \theta, x) = \lambda_v v(\theta, x).$$

  Thus the activation of v is preserved:

  $$a_v(\lambda \diamond \theta, x) = \mathbb{1}_{v(\lambda\diamond\theta,x)>0} = \mathbb{1}_{\lambda_v v(\theta,x)>0} = \mathbb{1}_{v(\theta,x)>0} = a_v(\theta, x).$$

  The same applies to the edge activations $a_{v\to u}(\theta, x)$.

- If v is a k-max-pooling neuron then the result is clear for the activation of v, as it is defined to be one irrespectively of θ and x. For the edge activation $u \to v$, we have by definition $a_{u\to v}(\theta, x) := 1$ if the neuron u is the first in $\mathrm{ant}(v)$ in lexicographic order to satisfy $u(\theta, x) := k\text{-}\mathtt{pool}\left((b_v + w(\theta, x)\theta^{w\to v})_{w\in\mathrm{ant}(v)}\right)$ and $a_{u\to v}(\theta, x) := 0$ otherwise. The rescaling multiplies each argument of the k-max-pooling by λ_v, preserving the ordering and hence the edge activations $a_{u\to v}(\theta, x)$.

This completes the proof that both path-lifting and path-activations are preserved by the operation $\lambda \in \mathbb{R}^H_{>0} \mapsto \lambda \diamond \theta$. We now turn to the second operation of setting $\theta^{v\to}$ to an arbitrary value, for a given $v \in \mathtt{In} - \mathtt{Dead}(\theta)$.

Invariance to setting $\theta^{v\to}$ arbitrarily, $v \in \mathtt{In} - \mathtt{Dead}(\theta)$. Consider $v \in \mathtt{In} - \mathtt{Dead}(\theta)$ and $\theta' = \theta$ except that $(\theta')^{v\to}$ is set arbitrarily. We have $\theta^{v\to} = (\theta')^{v\to} = 0$ and $b_v = b'_v = 0$ so $v(\theta, x) = v(\theta', x) = 0$ for any x. Therefore, the activations of v and the edges $u \to v$ are unchanged. The same holds for the edges $v \to w$ since v contributes the same to w in both cases. The rest of the network is kept identical, so the other activations remain unchanged, hence the invariance of the path-activations. For the path-lifting: $\Phi_p(\theta) = 0$ is unchanged for all paths p going through the input-dead neuron v; $\Phi_p(\theta)$ is also unchanged for other paths since they only involve entries of θ that are kept identical. As a result, $\Phi(\theta)$ is also preserved.

This completes the proof that both path-lifting and path-activations are the same for weakly rescaling-equivalent parameters. □

I now prove the formula that relates the output of a neuron to the path-lifting and path-activations.

Proof of Theorem 2.4.2. For any neuron v, recall that $\mathcal{P}^{\to v}$ is the set of paths ending at neuron v (Definition 2.3.2). We want to prove

$$\begin{aligned} v(\theta, x) &= \left\langle \Phi^{\to v}(\theta), A^{\to v}(\theta, x) \begin{pmatrix} x \\ 1 \end{pmatrix} \right\rangle \\ &= \sum_{p\in\mathcal{P}^{\to v}} \Phi_p(\theta) a_p(\theta, x) x_{p_0}, \end{aligned} \tag{2.7}$$

where we recall that p_0 denotes the first neuron of a path p (Definition 2.3.1). For paths starting at an input neuron, x_{p_0} is simply the corresponding coordinate of x. For other paths, we use the convention $x_u := 1$ for any neuron $u \in N \setminus N_{\mathrm{in}}$.

The proof of Equation (2.7) goes by induction on a topological sorting [Cormen et al., 2009] of the neurons. We start with input neurons since by Definition 2.2.2, these are the ones without antecedents so they are the first to appear in a topological sorting.

Consider an input neuron v. The only path in $\mathcal{P}^{\to v}$ is $p = v$. By Definition 2.3.3, it holds $\Phi_p(\theta) = 1$ (empty product) and $a_p(\theta, x) = a_v(\theta, x) = 1$. Moreover, we have $v(\theta, x) = x_v$ (Definition 2.2.2). This proves Equation (2.7) for input neurons.

Now, consider $v \notin N_{in}$ and assume that Equation (2.7) holds true for every neuron $u \in \mathrm{ant}(v)$. We prove by cases that

$$v(\theta, x) = a_v(\theta, x) b_v + \sum_{u\in\mathrm{ant}(v)} u(\theta, x) a_{u\to v}(\theta, x) \theta^{u\to v}. \tag{2.8}$$

There is no particular difficulty to prove Equation (2.8) as it just consists of carefully unrolling the definition of $v(\theta, x)$ (Definition 2.2.2) depending on the activation function of v. The proof is given in Appendix A for completeness.

Using the induction hypothesis on the antecedents of v, Equation (2.8) implies

$$v(\theta, x) = a_v(\theta, x)b_v + \sum_{u \in \mathrm{ant}(v)} \left(\sum_{p \in \mathcal{P}^{\to u}} \Phi_p(\theta) a_p(\theta, x) x_{p_0} \right) a_{u \to v}(\theta, x) \theta^{u \to v}.$$

We want to prove that this is equal to

$$\sum_{p \in \mathcal{P}^{\to v}} \Phi_p(\theta) a_p(\theta, x) x_{p_0}.$$

A path $\tilde{p} \in \mathcal{P}^{\to v}$ is either the path $\tilde{p} = v$ starting and ending at v, or it can be written in a unique way as $\tilde{p} = p \to v$ where $p \in \mathcal{P}^{\to u}$ is a path ending at an antecedent u of v. For the simple path $\tilde{p} = v$, it holds by definition (Definition 2.3.3): $a_{\tilde{p}}(\theta, x) = a_v(\theta, x)$, $\Phi_{\tilde{p}}(\theta) = b_v$ and by the convention used in this proof we have $x_{\tilde{p}_0} = x_v = 1$ since v is not an input neuron. This shows:

$$a_v(\theta, x)b_v = \Phi_{\tilde{p}}(\theta) a_{\tilde{p}}(\theta, x) x_{\tilde{p}_0}.$$

When $\tilde{p} = p \to v$ as above, it holds by definition: $x_{p_0} = x_{\tilde{p}_0}$, $\Phi_{\tilde{p}}(\theta) = \Phi_p(\theta)\theta^{u \to v}$ and $a_{\tilde{p}}(\theta, x) = a_p(\theta, x) a_{u \to v}(\theta, x)$. It then holds:

$$\Phi_p(\theta) a_p(\theta, x) x_{p_0} a_{u \to v}(\theta, x) \theta^{u \to v} = \Phi_{\tilde{p}}(\theta) a_{\tilde{p}}(\theta, x) x_{\tilde{p}_0}.$$

This concludes the induction and proves the result. □

2.5 Conclusion

When analyzing neural networks, the primary object of interest is often the function R_θ realized by the network. Indeed, key challenges are to control the robustness of the function to input perturbations, to ensure that it generalizes well, etc. This is done by controlling quantities that only make sense on the function R_θ rather than its raw parameters θ, such as Lipschitz constants, integrals of the function (for the generalization error), etc.

However, in practice, we often end up analyzing the parameters θ, as this is the concrete representation that we fully observe and manipulate in practice. Therefore, we often end up controlling quantities of interest on the functions by quantities on the parameters. The advantage is that the parameter space is finite-dimensional, so many quantities such as norms are easy to compute in this space, while it is not the case in the function space (e.g., the Lipschitz constant of neural networks is NP-hard to compute [Virmaux and Scaman, 2018]).

While controlling function properties in terms of parameter properties can be insightful to some extent, it can also be problematic: I explained for example that the existence of rescaling symmetries can lead to vacuous bounds in Section 1.1.2.

Therefore, it is desirable to have objects that are easier to manipulate than the functions R_θ, but more faithful to the functions R_θ than the raw parameters θ. The path-lifting $\Phi(\theta)$ may offer such a possibility: it lies in an intermediate space between the parameter space and the function space and it takes some of the best properties of both worlds:

1. it is finite-dimensional, so it is probably easier to compute quantities (e.g., norms) on $\Phi(\theta)$ than on R_θ;

2. it is also invariant under symmetries, so this space is more faithful than the parameter space;

3. and its relation to the function space is much nicer compared to the parameter space: the function R_θ is locally linear in $\Phi(\theta)$ against locally polynomial in θ, as can be seen from Equation (2.6). Therefore, the relation of R_θ with $\Phi(\theta)$ is expected to be easier to analyze than the relation with θ.

This is summarized in Figure 2.5.

	θ parameter space	$\Phi(\theta)$ path-lifting space	R_θ function space
	what we end up analyzing	**what we should analyze?**	what we want to analyze
dim$< \infty$	✓	✓	✗
invariance	✗	✓	✓
relation to R_θ	locally polynomial	locally linear	

Figure 2.5: The function Φ lifts the parameter space in an intermediate space that seems promising for the analysis of ReLU networks.

This promising perspective on the path-lifting will be made concrete in the next chapters, where I will show that the path-lifting can be used to derive Lipschitz and generalization bounds that are easy to compute, and at least as fine as the bounds obtained with the same proof techniques but applied directly to the raw parameters θ.

Chapter 3

Lipschitz properties and consequence for pruning

This chapter is based on results in Gonon et al. [2024a,b].

A critical aspect of understanding the behavior of neural networks is to study their Lipschitz properties, which provide bounds on how much the network's output $R_\theta(x)$ can change in response to changes in its input x or parameters θ. These properties are essential for robustness of neural networks, pruning, quantization or generalization.

This chapter establishes Lipschitz properties for general DAG networks in terms of the path-lifting $\Phi(\theta)$. While I will crucially use these properties for generalization in Chapter 4, **this chapter shows already a first concrete application of these properties: designing a new pruning method that matches so-called magnitude pruning in terms of accuracy, while being robust to rescaling symmetries**. Pruning is a type of technique used to reduce the size and complexity of networks, while still preserving their accuracy in many cases of interests. Pruning is crucial for deploying neural networks in resource-constrained environments, such as mobile devices or embedded systems, where computational resources and memory are limited.

The outline is as follows.

- Section 3.1 defines so-called *mixed path-norms* (Definition 3.1.1), norms of the path-lifting, that I will use to establish Lipschitz properties of neural networks [Gonon et al., 2024a]. *Despite the combinatorial ambient dimension of the path-lifting, this section shows that these norms can be computed very easily (one forward-pass) in Theorem 3.1.1.* The latter is vital for the practical application of the theoretical bounds derived in this thesis.

- Section 3.2 introduces the notion of *normalized parameters* (Definition 3.2.1), which are roughly parameters for which mixed path-norms and standard parameter norms coincide [Gonon et al., 2024a]. Moreover, it shows that any parameters have a weakly rescaling-equivalent version of them that are normalized (Lemma 3.2.1). This will be crucial to show that bounds established in terms of mixed path-norms are at least as fine as, or finer than, the ones derived with the same techniques but applied directly to the raw parameters [Gonon et al., 2024a,b].

- Section 3.3 extends to arbitrary DAG networks a Lipschitz bound for the function

$x \mapsto R_\theta(x)$ in terms of mixed path-norms of $\Phi(\theta)$ that was previously known only for scalar-valued LFCN without biases [Neyshabur et al., 2015]: for every input x, x' and parameters θ (Theorem 3.3.1) [Gonon et al., 2024a]

$$\|R_\theta(x) - R_\theta(x')\|_r \leqslant \|\Phi(\theta)\|_{1,r}\|x - x'\|_\infty.$$

These extended bounds are shown to preserve their key properties: invariance to permutation and rescaling symmetries, ease of computation, and tightness compared to the ones obtained with similar techniques but applied directly to the raw parameters [Gonon et al., 2024a].

- Section 3.4 establishes a new Lipschitz bound for the function $\theta \mapsto R_\theta(x)$ in terms of the path-lifting: for every input x and parameters θ, θ' with $\theta_i\theta'_i \geqslant 0$ for all i (Theorem 3.4.1) [Gonon et al., 2024b]

$$\|R_\theta(x) - R_{\theta'}(x)\|_1 \leqslant \|\Phi(\theta) - \Phi(\theta')\|_1 \max(\|x\|_\infty, 1).$$

This assumption ($\theta_i\theta'_i \geqslant 0, \forall i$) is in particular true in practical cases of interests, such as when θ' has been obtained from θ by pruning or by quantization with rounding towards zero. Once again, this bound is shown to be invariant to permutation and rescaling symmetries, easy to compute, and finer than the one obtained with a similar technique but applied directly to the raw parameters [Gonon et al., 2024b]. *And the proof of Theorem 3.4.1 is my favorite one of this thesis, don't miss it.*

- Section 3.5 uses the new Lipschitz bound in θ to design a compression method based on pruning [Gonon et al., 2024b]. The new pruning method not only matches the accuracy of standard magnitude pruning, but is also robust to rescaling symmetries.

3.1 Mixed path-norms and their efficient computation

This section introduces the norms that will appear in our Lipschitz bounds. Despite the high ambient dimension of the path-lifting $\Phi(\theta)$, these norms of $\Phi(\theta)$ can be computed in only one forward-pass of the model. This is a blessing as it allows all theoretical bounds of this thesis to be easily computed.

Motivation. Consider a LFCN (Definition 2.1.1) with L affine layers corresponding to functions

$$R_\theta(x) = M_L \,\mathrm{ReLU}(M_{L-1} \ldots \mathrm{ReLU}(M_1 x + b_1) \ldots + b_{L-1}) + b_L,$$

with matrices $M_\ell \in \mathbb{R}^{d_{\ell+1} \times d_\ell}$, biases $b_\ell \in \mathbb{R}^{d_{\ell+1}}$ for some number of neurons d_ℓ. The number of parameters is

$$\sum_{\ell=1}^{L} d_{\ell+1}d_\ell + \sum_{\ell=1}^{L} d_{\ell+1} = \sum_{\ell=1}^{L} d_{\ell+1}(d_\ell + 1),$$

while the number of paths (Definition 2.3.1) is

$$\sum_{\ell=1}^{L}\prod_{k=\ell}^{L+1} d_k.$$

In the case where $d_\ell = d$ for all ℓ, the number of parameters is of order Ld^2 while the number of paths is of order d^L. This shows that the ambient dimension of $\Phi(\theta)$ grows exponentially in the number of layers, and in general, in the depth of the network, where I recall that the depth is the maximum length of a path (Definition 2.3.1), as opposed to the number of parameters that is linear in the depth. When analyzing the function R_θ through the lens of the path-lifting $\Phi(\theta)$, it is therefore important to ensure that at least some quantities of $\Phi(\theta)$ can be computed without ever having to explicitly compute $\Phi(\theta)$ or enumerate the paths. This is the content of the following theorem: mixed ℓ^q-norms of the vector $\Phi(\theta)$ can be computed in a single forward-pass, up to replacing $*$-max-pooling activations by the identity.

Definition 3.1.1. *(Mixed path-norm) For $0 < q, r \leqslant \infty$, define the mixed (q,r)-path-norm of the path-lifting $\Phi(\theta)$ as*

$$\|\Phi(\theta)\|_{q,r} := \left\|\left(\|\Phi^{\to v}(\theta)\|_q\right)_{v\in N_{out}}\right\|_r. \tag{3.1}$$

Definition 3.1.1 captures in particular all classical ℓ^q-norms as for $r = q$ we have $\|\Phi(\theta)\|_{q,q} = \|\Phi(\theta)\|_q$. Moreover, for scalar-valued networks, all mixed path-norms are classical ℓ^q-norms since $N_{\text{out}} = \{v\}$, $\|\Phi(\theta)\|_{q,r} = \|\Phi^{\to v}(\theta)\|_q = \|\Phi(\theta)\|_q$.

Theorem 3.1.1 ([Gonon et al., 2024a])**.** *Consider an architecture $G = (N, E, (\rho_v)_{v\in N\setminus N_{in}})$ as in Definition 2.2.2. Consider the architecture $\tilde{G} := (N, E, (\tilde{\rho}_v)_{v\in N\setminus N_{in}})$ where $*$-max-pooling neurons are replaced by ones with the identity as an activation function: $\tilde{\rho}_v := \mathrm{id}$ if $v \in N_{\texttt{*-pool}}$ and $\tilde{\rho}_v := \rho_v$ otherwise. Consider $q \in (0,\infty)$, $r \in (0,\infty]$ and arbitrary parameters $\theta \in \mathbb{R}^G = \mathbb{R}^{\tilde{G}}$. For a vector α, denote $|\alpha|^q$ the vector deduced from α by applying $x \mapsto |x|^q$ coordinate-wise. Denote by $\mathbf{1}$ the input full of ones. It holds:*

$$\|\Phi(\theta)\|_{q,r} = \| \ |R^{\tilde{G}}_{|\theta|^q}(\mathbf{1})|^{1/q} \ \|_r. \tag{3.2}$$

Moreover, the formula is false *in general if the $*$-max-pooling neurons have not been replaced with ones having the identity as an activation function (that is if the forward-pass is done on G rather than $\tilde{G}$).*

I made available at `github.com/agonon/pathnorm_toolkit` a code to compute path-norms in one forward-pass, for a wide variety of models, including ResNets, VGGs etc. The code is also easy to extend to new architectures: the user only needs to specify the max-pooling layers (which the code will replace by layers with the identity as activation function, as specified by Theorem 3.1.1), and the kernel size K of average-pooling layers, in order to explicitly consider average-pooling neurons as ones with the identity as activation function, and with incoming weights all equal to $1/K$ (as specified below Definition 2.2.2). In particular, **this code allows anyone to efficiently compute all the bounds of this thesis**.

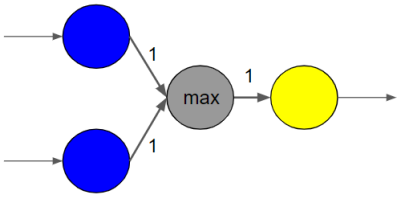


Figure 3.1: Example of a network where one must replace the max-pooling neuron to compute the path-norm with a single forward-pass as in Equation (3.2).

Proof of Theorem 3.1.1. Figure 3.1 shows that Equation (3.2) is false if the $*$-max-pooling neurons have not been replaced with ones having the identity as activation function, since the forward-pass with input $\mathbf{1}$ yields output 1 while the ℓ^1-path-norm is 2.

Let me now show that Equation (3.2) is a simple consequence of Theorem 2.4.2. By continuity in θ of both sides of Equation (3.2), it is sufficient to prove it when every coordinate of θ in E is nonzero. Note that by Definition 2.3.3, the path-lifting Φ^G and $\Phi^{\tilde{G}}$ associated respectively with G and $\tilde{G}$ are the same since G and $\tilde{G}$ have the same underlying DAG. Denote by $\Phi = \Phi^G = \Phi^{\tilde{G}}$ the common path-lifting, and by $a^{\tilde{G}}$ the path-activations associated with $\tilde{G}$. Denote by $\mathcal{P}^{\to v}$ the set of paths ending at a given neuron v of $\tilde{G}$. According to Theorem 2.4.2, it holds for every output neuron v of $\tilde{G}$ with $x = \mathbf{1}$, using the convention $x_u := 1$ if $u \notin N_{\text{in}}$

$$(R^{\tilde{G}}_{|\theta|^q}(\mathbf{1}))_v = \sum_{p \in \mathcal{P}^{\to v}} \Phi_p(|\theta|^q) a_p^{\tilde{G}}(|\theta|^q, x) x_{p_0} = \sum_{p \in \mathcal{P}^{\to v}} \Phi_p(|\theta|^q) a_p^{\tilde{G}}(|\theta|^q, \mathbf{1}).$$

Since we restricted to θ_E with nonzero coordinates, $|\theta_E|^q$ has positive coordinates. As $\tilde{G}$ has only neurons with the ReLU or the identity as activations, it follows by a simple induction on the neurons that $u(|\theta|^q, \mathbf{1}) > 0$ for every neuron u of $\tilde{G}$. Thus, every path $p \in \mathcal{P}^{\tilde{G}}$ is active, that is, $a_p^{\tilde{G}}(|\theta|^q, \mathbf{1}) = 1$. Since by Definition 2.3.3, $\Phi_p(|\theta|^q) = |\Phi_p(\theta)|^q$, we obtain:

$$(R^{\tilde{G}}_{|\theta|^q}(\mathbf{1}))_v = \sum_{p \in \mathcal{P}^{\to v}} |\Phi_p(\theta)|^q = \|\Phi^{\to v}(\theta)\|_q^q.$$

The claim follows by the definition of the mixed path-norm (Definition 3.1.1). □

Positioning. The formula given in Theorem 3.1.1 is the first of its kind to fully encompass ReLU networks with biases, average/$*$-max-pooling, and skip connections, such as ResNets. An equivalent formula is stated in the specific case of *layered fully-connected* ReLU networks *without biases* (and *no pooling/skip connections*) in Dziugaite et al. [2020, Appendix C.6.5] and Jiang et al. [2020, Equations (43) and (44)] but without proof (and only for ℓ^2-path-norm instead of general mixed $\ell^{q,r}$ path-norm). Actually, *this equivalent formula turns out to be false when there are $*$-max-pooling neurons* as one must replace $*$-max-pooling neurons with ones having the identity as activation function, see Theorem 3.1.1. Care must also be taken with average-pooling neurons that must be considered as neurons with the identity as activation function, and the incoming weights must also be transformed with $x \mapsto |x|^q$ when computing $|\theta|^q$.

At this point, we have a promising object $\Phi(\theta)$ that is well suited for theoretical analysis of the function R_θ and more faithful to the function R_θ than the raw parameters θ. I have

shown that some norms of $\Phi(\theta)$ can be computed in a single forward-pass. Moreover, there is a nice relationship between the function R_θ and $\Phi(\theta)$ as shown in Theorem 2.4.2: the function R_θ is locally linear in $\Phi(\theta)$. The next challenge is to concretely use $\Phi(\theta)$ to analyze the function R_θ. I will do this in Section 3.3 by showing that mixed path-norms can be used to bound Lipschitz constants of R_θ. For now, let me discuss how mixed path-norms can be related to classical norms of the raw parameters θ for certain θ.

3.2 Normalized parameters: when mixed path-norms coincide with parameter norms

To better understand how mixed path-norms $\|\Phi(\theta)\|_{q,r}$ relate to standard mixed parameter norms $\|\theta\|_{q,r}$, I introduce the concept of *normalized* parameters. These are parameters that are such that the ℓ^q-norms of some subvectors of $\Phi(\theta)$ coincide with the ℓ^q-norms of some subvectors of θ. I also show that given any parameters θ, there exists a *weakly rescaling-equivalent* version of θ that is *normalized*. Thanks to this property, I will be able to show that the Lipschitz bounds established in the next section in terms of mixed path-norms are at least as fine as the standard ones based on parameter norms. The former will correspond to infimum over all *weakly* rescaling-equivalent versions of the latter.

Definition 3.2.1. *Consider $0 < q \leqslant \infty$. Parameters θ are said* q-normalized *if*

1. *for every $v \in N \setminus (N_{out} \cup N_{in})$:*

$$\|\Phi^{\to v}(\theta)\|_q = \| \begin{pmatrix} \theta^{\to v} \\ b_v \end{pmatrix} \|_q \in \{0, 1\},$$

2. *in addition if this is 0 (input-dead neuron, Definition 2.4.3), it also holds $\theta^{v\to} = 0$ (beware, this is a condition on the outgoing edges of v).*

Let me now introduce Algorithm 3.2.1 that allows to find normalized parameters in each *weak* (but in general not strong) rescaling-equivalence class of parameters. This algorithm proceeds iteratively over the neurons, and for each neuron, it normalizes the incoming weights by applying one of the two operations allowed for weakly equivalent parameters (Definition 2.4.3). The neurons are considered in the order given by a topological sorting [Cormen et al., 2009, Section 22.4], that is an order on the neurons such that if $u \to v$ is an edge then u comes before v in this ordering.

Lemma 3.2.1. *Consider $q \in (0, \infty]$. If θ is the output of Algorithm 3.2.1 for the input $\tilde{\theta}$ then θ is q-normalized (Definition 3.2.1) and θ is weakly rescaling-equivalent to $\tilde{\theta}$ (Definition 2.4.3).*

There is no particular technical obstacle to the proof of Lemma 3.2.1, it is just a matter of analyzing correctly the algorithm. I provide the proof in Appendix B.1.

Algorithm 3.2.1 Normalization of parameters for norm $q \in (0, \infty]$

Consider a topological sorting $v_1, \dots, v_k$ of the neurons
for $v = v_1, \dots, v_k$ **do**
  if $v \notin N_{\text{in}} \cup N_{\text{out}}$ **then**
    $\lambda_v \leftarrow \left\| \begin{pmatrix} \theta^{\to v} \\ b_v \end{pmatrix} \right\|_q$
    if $\lambda_v = 0$ **then**
      $\theta^{v\to} \leftarrow 0$
    else
      $\begin{pmatrix} \theta^{\to v} \\ b_v \end{pmatrix} \leftarrow \frac{1}{\lambda_v} \begin{pmatrix} \theta^{\to v} \\ b_v \end{pmatrix}$ ▷ normalize incoming weights and bias
      $\theta^{v\to} \leftarrow \lambda_v \times \theta^{v\to}$ ▷ rescale outgoing weights to preserve the function R_θ

Positioning. In the special case of *scalar-valued LFCNs*, Algorithm 3.2.1 has been used in the literature to make the ℓ^1-path-norm coincide with a product of layers's norms [Neyshabur et al., 2015, Lemma 19][Barron and Klusowski, 2019, Theorem 1]. It turns out that this algorithm is more general and applies to any DAG ReLU network. The first formal description of this algorithm is given in Algorithm 3.2.1. Moreover, I identified and isolated new concepts, *normalized parameters* and *weak rescaling-equivalence*, that will hopefully help at better understanding the roles of the object at play, and that will lead to more convenient notations later on when comparing mixed path-norms with norms of the parameters.

3.3 Lipschitzness in x

An important property of neural networks is their robustness to input change: does a small perturbation of the input x imply a large perturbation of the output $R_\theta(x)$? For instance, imagine that a car has learned to recognize images x of stop signs using a neural network R_θ. If this network is robust enough to input perturbation, the car will still recognize stop signs even if they are a little dirty or have a sticker on them.

Unfortunately, it has been widely reported that networks lack of robustness: small magnitude input perturbations that are imperceptible to the human eye, such as adding noise, can make the neural network make prediction errors Szegedy et al. [2014]. Even real-world attacks have been found: a stop sign recognized by a neural network can be misclassified as a speed limit sign by adding a small sticker on it [Eykholt et al., 2018]. The desire for robustness motivates the study of the Lipschitz constant of $x \mapsto R_\theta(x)$.

Section 3.3.1 contains the main contribution of this section: a bound on the Lipschitz constant of $x \mapsto R_\theta(x)$ in terms of the mixed-norms of $\Phi(\theta)$. Section 3.3.2 shows that these bounds correspond to the infimum over all weakly rescaling-equivalent parameters of bounds obtained with similar techniques but applied directly to the raw parameters θ, with the infimum met at normalized parameters (Definition 3.2.1). This shows that the bounds based on mixed path-norms are at least as fine as the ones based on norms expressed in terms of the raw parameters, essentially thanks to the fact that the path-lifting Φ captures the rescaling symmetries.

3.3.1 Main result

The next theorem is the main result: mixed path-norms of $\Phi(\theta)$ can be used to bound the Lipschitz constant of $x \mapsto R_\theta(x)$.

Theorem 3.3.1. *Consider $0 < r \leqslant \infty$. For every parameters θ and inputs x, x':*

$$\|R_\theta(x) - R_\theta(x')\|_r \leqslant \|\Phi(\theta)\|_{1,r} \|x - x'\|_\infty.$$

Note that the ℓ^∞-norm is the smallest of the ℓ^q-norms, so the bound of Theorem 3.3.1 also implies bounds for $\|x - x'\|_q$ on the right-hand side.

Numerical evaluations of path-norm-based bounds are deferred to Chapter 7, after other bounds have been introduced.

To prove Theorem 3.3.1, I will use the next lemma.

Lemma 3.3.1. *Consider a matrix $A \in \{0,1\}^{d \times p}$ and a vector $\varphi \in \mathbb{R}^p$. Assume that each column of A has at most one non-zero entry. For every $q \in [1, \infty]$, it holds*

$$\|A\varphi\|_q \leqslant p^{\frac{q-1}{q}} \|\varphi\|_q$$

with the convention $\frac{q-1}{q} = 1$ for $q = \infty$.

Proof of Lemma 3.3.1. I only treat $q \in [1, \infty)$, the case $q = \infty$ is similar. This is just a simple worst-case analysis. The matrix A is binary with at most one non-zero entry per column $j \in \{1, \dots, p\}$ so we have

$$\sum_{i=1}^{d} |A_{ij}\varphi_j| \leqslant |\varphi_j|.$$

By convexity of $x \mapsto x^q$, we have for scalars $x_j \geqslant 0$, $(\sum_{j=1}^p x_j)^q \leqslant p^{q-1} \sum_{j=1}^p x_j^q$, so

$$\begin{aligned}
\|A\varphi\|_q^q = \sum_{i=1}^{d} |(A\varphi)_i|^q &= \sum_{i=1}^{d} \left(\sum_{j=1}^{p} |A_{ij}\varphi_j| \right)^q \\
&\leqslant p^{q-1} \sum_{i=1}^{d} \sum_{j=1}^{p} |A_{ij}\varphi_j|^q = p^{q-1} \sum_{j=1}^{p} \sum_{i=1}^{d} |A_{ij}\varphi_j|^q \\
&\leqslant p^{q-1} \sum_{j=1}^{p} |\varphi_j|^q = p^{q-1} \|\varphi\|_q^q.
\end{aligned}$$

This proves the claim. □

Applying Lemma 3.3.1 to $A = A(\theta, x)^T$ and $\varphi = \Phi(\theta)$, this yields:

$$\|A(\theta, x)^T \Phi(\theta)\|_q \leqslant |\mathcal{P}|^{\frac{q-1}{q}} \|\Phi(\theta)\|_q$$

where $|\mathcal{P}|$ is the number of paths. *This worst-case analysis is the primary reason for the exclusive presence of the ℓ^1-path-norm ($q = 1$) in bounds established in the literature and in this thesis: it is the only choice of q that avoids a combinatorial factor $|\mathcal{P}|$ in the bound, at the price of settling for the largest norm $q = 1$. I haven't find this discussed in the literature. I will discuss it in Chapter 7.*

Proof of Theorem 3.3.1. Consider parameters θ. Consider inputs x, x' with the same path-activations with respect to θ: $A(\theta, x) = A(\theta, x')$. For $0 < r < \infty$:

$$\begin{aligned}
\|R_\theta(x) - R_\theta(x')\|_r^r &\underset{\text{Theorem 2.4.2}}{=} \sum_{v \in N_{\text{out}}} \left| \left\langle \Phi^{\to v}(\theta), A^{\to v}(\theta, x) \begin{pmatrix} x \\ 1 \end{pmatrix} - A^{\to v}(\theta, x') \begin{pmatrix} x' \\ 1 \end{pmatrix} \right\rangle \right|^r \\
&\underset{A(\theta,x)=A(\theta,x')}{=} \sum_{v \in N_{\text{out}}} \left| \left\langle \left(A^{\to v}(\theta, x')\right)^T \Phi^{\to v}(\theta), \begin{pmatrix} x \\ 1 \end{pmatrix} - \begin{pmatrix} x' \\ 1 \end{pmatrix} \right\rangle \right|^r \\
&\underset{\text{Hölder}}{\leqslant} \sum_{v \in N_{\text{out}}} \| \left(A^{\to v}(\theta, x')\right)^T \Phi^{\to v}(\theta)\|_1^r \, \|x - x'\|_\infty^r \\
&\underset{\textit{Lemma 3.3.1}}{\leqslant} \sum_{v \in N_{\text{out}}} \|\Phi^{\to v}(\theta)\|_1^r \|x - x'\|_\infty^r \\
&= \|\Phi(\theta)\|_{1,r}^r \|x - x'\|_\infty^r.
\end{aligned}$$

When $0 < r < \infty$, I just proved the claim locally on each region where the path-activations $A(\theta, \cdot)$ are constant. This remains true on the boundary of these regions by continuity. It then expands to the whole domain by triangular inequality because on any segment joining any pair of inputs, there is a finite number of times when the region changes. The proof can be easily adapted to $r = \infty$. □

Related works. For $r = 1$, we simply have $\|\Phi(\theta)\|_{1,r} = \|\Phi(\theta)\|_1$. For this *specific value of* r, and in the *specific case* of *scalar-valued LFCN without biases*, the conclusion of Theorem 3.3.1 is already mentioned in Neyshabur [2017, before Section 3.4], and proved in Furusho [2020, Theorem 5]. Theorem 3.3.1 extends it to arbitrary DAG ReLU networks, and to arbitrary $r \in (0, \infty]$. This requires the introduction of *mixed* path-norms, which, to the best of my knowledge, have not been considered before in the literature.

The Lipschitz bound of Theorem 3.3.1 is invariant to permutation and rescaling symmetries. For rescalings, this follows from the fact that the path-lifting is invariant under rescaling symmetries (Theorem 2.4.1). Let me now address permutation symmetries. Given a network architecture, there are usually permutations of the neurons and corresponding parameters of θ that leave the function R_θ unchanged [Stock and Gribonval, 2023]. This invariance arises because computing $R_\theta(x)$ involves computing sums over the set of antecedents of different neurons v, and these sums remain invariant under permutations of $\text{ant}(v)$. For example, in LFCNs, any pair of neurons in the same layer can be permuted without altering the network's function.

The path-lifting $\Phi(\theta)$ is covariant under such permutations because permuting neurons results in a corresponding permutation of paths. Since the ℓ^q-path-norm in Theorem 3.3.1 involves sums over sets of paths, it is invariant under permutation symmetries.

Note that permutations and rescalings are provably the *only* symmetries of LFCNs with $L = 2$ affine layers under non-degeneracy conditions[1] [Stock and Gribonval, 2023, Theorem 3].

[1]This assumes no "dead" neurons (neurons that never transmit anything because of zero incoming or outgoing weights) nor "twin" neurons (neurons that perform identical functions and could be merged).

3.3.2 Comparison with bounds directly expressed in terms of θ

This section shows that the Lipschitz bound of Theorem 3.3.1 is at least as fine as the standard one based on a product of layers' norms in the specific case of LFCNs, and even extend this comparison to general DAGs. This illustrates the interest of considering the path-lifting $\Phi(\theta)$ rather than the raw parameters θ.

For LFCN functions $R_\theta(x) = M_L \,\mathrm{ReLU}(M_{L-1} \,\mathrm{ReLU}(M_1 x + b_1) + b_{L-1}) + b_L$ without biases, the standard Lipschitz bound is the product of the layers' norms of the layers.

Lemma 3.3.2 (see e.g., Combettes and Pesquet [2020])**.** *For a layered fully-connected ReLU network with L affine layers, of the form*

$$R_\theta(x) = M_L \,\mathrm{ReLU}(M_{L-1} \ldots \mathrm{ReLU}(M_1 x + b_1) + b_{L-1}) + b_L,$$

it holds for every x, x'

$$\|R_\theta(x) - R_\theta(x')\|_\infty \leqslant \left(\prod_{\ell=1}^{L} \|M_\ell\|_{1,\infty}\right) \|x - x'\|_\infty,$$

where $\|M\|_{1,\infty} := \max_{\text{row of } M} \|\text{row}\|_1$ is the operator norm of M induced by the ℓ^∞-norm on the input and output spaces.

Proof. This is well-known. I still provide a proof for $L = 2$, and the general case easily follows by an induction: (highlighting in orange x and x'):

$$\begin{aligned}
\|R_\theta(x) - R_\theta(x')\|_\infty &= \|M_2 \,\mathrm{ReLU}(M_1 x + b_1) + b_2 - M_2 \,\mathrm{ReLU}(M_1 x' + b_1) + b_2\|_\infty \\
&= \|M_2 \left(\mathrm{ReLU}(M_1 x + b_1) - \mathrm{ReLU}(M_1 x' + b_1)\right)\|_\infty \\
&\leqslant \|M_2\|_{1,\infty} \|\mathrm{ReLU}(M_1 x + b_1) - \mathrm{ReLU}(M_1 x' + b_1)\|_\infty \\
&\leqslant \|M_2\|_{1,\infty} \|M_1 x + b_1 - M_1 x' + b_1\|_\infty \qquad \text{ReLU is 1-Lipschit} \\
&\leqslant \|M_2\|_{1,\infty} \|M_1 \left(x - x'\right)\|_\infty \\
&\leqslant \|M_2\|_{1,\infty} \|M_1\|_{1,\infty} \|x - x'\|_\infty .
\end{aligned}$$

□

This bound is not invariant under rescaling symmetries of R_θ. Because of that, this bound can be vacuous depending on the scaling of the parameters at hand: Figure 3.2 shows an example where this product is arbitrarily large while the path-norm is equal to zero.

It is known for *scalar-valued LFCNs without biases* that the ℓ^1-path-norm is a smaller Lipschitz bound as we have [Neyshabur et al., 2015, Theorem 5]

$$\|\Phi(\theta)\|_1 \leqslant \prod_{\ell=1}^{L} \|M_\ell\|_{1,\infty}.$$

with equality if the parameters are properly rescaled.

With the new concepts of *normalized parameters* and *weak rescaling-equivalence*, we can already provide a new perspective in this specific case of scalar-valued LFCNs without biases: there is equality for normalized parameters, inequality in general, and since there

are normalized parameters in each weak rescaling-equivalence class (Lemma 3.2.1), the result of [Neyshabur et al., 2015] can be rewritten as:

$$\|\Phi(\theta)\|_1 = \min_{\substack{\theta'=(M'_1,\dots,M'_L),\\ \theta'\sim_{WR}\theta}} \prod_{\ell=1}^{L} \|M'_\ell\|_{1,\infty}.$$

In what follows, I extend this comparison to vector-valued LFCNs with biases, and even to general DAGs as in Definition 2.2.2 by introducing a quantity that plays the role of the product of layers' norms in this situation. Moreover, I also extend the comparison to other mixed path-norms and products of layers' norms.

From now on, I always consider $q \in (0,\infty)$ and $r \in (0,\infty]$ if not specified otherwise.

From scalar-valued to vector-valued networks: from standard to mixed ℓ^q-norms. Theorem 3.3.1 shows that going from a *scalar-valued* network (LFCN or, more generally a DAG) to a *vector-valued* network requires going from standard ℓ^q-norms $\|\Phi(\theta)\|_q$, as considered in Neyshabur et al. [2015], to mixed path-norms $\|\Phi(\theta)\|_{q,r}$, which is, to the best of our knowledge, first introduced in Definition 3.1.1. I will compare such mixed path-norms to products of mixed quasi-norms:

$$\|M\|_{q,\infty} = \max_{\text{row of } M} \|\text{row}\|_q.$$

From LFCN to DAG: extending the product of layers' norms. There is no notion of a "layer" M_ℓ in a general DAG, hence the product $\prod_{\ell=1}^{L} \|M_\ell\|_{q,\infty}$ makes no sense anymore in this general context as a general ReLU network function R_θ as in Definition 2.2.2 cannot be decomposed as a composition of affine maps and ReLU. Therefore, I introduce a new quantity that plays the role of the product of layers' norms for general DAGs.

Definition 3.3.1. *Consider $q \in (0,\infty)$. For practical purposes, we denote $\underline{b}_v := b_v$ when $v \in N \setminus N_{in}$, and $\underline{b}_v := 1$ if $v \in N_{in}$. For every path $p \in \mathcal{P}$, consider*

$$\Pi(\theta,p,q) := \left(\sum_{\ell=0}^{\texttt{length}(p)} \left|\underline{b}_{p_\ell}\right|^q \prod_{k=\ell+1}^{\texttt{length}(p)} \|\theta^{\to p_k}\|_q^q \right)^{1/q}, \tag{3.3}$$

with the convention that an empty product is equal to one. For $q \in (0,\infty)$ and $r \in (0,\infty]$, define:

$$\Pi_{q,r}(\theta) := \left\| \left(\max_{p\in\mathcal{P}^{\to v}} \Pi(\theta,p,q) \right)_{v\in N_{out}} \right\|_r. \tag{3.4}$$

Let me show that $\Pi_{q,\infty}$ generalizes the product of layers' norms to an arbitrary DAG.

Lemma 3.3.3. *Consider $q \in (0,\infty)$. In a LFCN $R_\theta(x) = M_L\,\text{ReLU}(M_{L-1}\dots\text{ReLU}(M_1 x))$ with L layers and without biases, it holds:*

$$\Pi_{q,\infty}(\theta) = \prod_{\ell=1}^{L} \|M_\ell\|_{q,\infty}.$$

Proof. In a LFCN, all the neurons of two consecutive layers are connected, so $\prod_{\ell=1}^{L} \|M_\ell\|_{q,\infty}$ is also the maximum over all paths starting from an input neuron (and ending at an output neuron, by definition of the set of paths $\mathcal{P}$) of the product of ℓ^q-norms:

$$\prod_{\ell=1}^{L} \|M_\ell\|_{q,\infty} = \max_{p\in\mathcal{P}, p_0\in N_{\text{in}}} \prod_{\ell=1}^{\texttt{length}(p)} \|\theta^{\to p_\ell}\|_q.$$

Since there are no biases (corresponding to $\gamma_u = b_u = 0$ for every $u \notin N_{\text{in}}$) we have

$$\Pi(\theta, p, q) = \begin{cases} \prod_{\ell=1}^{\texttt{length}(p)} \|\theta^{\to p_\ell}\|_q & \text{if } p_0 \in N_{\text{in}}, \\ 0 & \text{otherwise.} \end{cases}$$

This shows the claim:

$$\begin{aligned} \Pi_{q,\infty}(\theta) &= \max_{v\in N_{\text{out}}} \max_{p\in\mathcal{P}^{\to v}} \Pi(\theta, p, q) = \max_{p\in\mathcal{P}} \Pi(\theta, p, q) \\ &= \max_{p\in\mathcal{P}, p_0\in N_{\text{in}}} \prod_{\ell=1}^{\texttt{length}(p)} \|\theta^{\to p_\ell}\|_q = \prod_{\ell=1}^{\texttt{length}(p)} \|M_\ell\|_{q,\infty}. \end{aligned}$$

□

Let me now prove that the mixed path-norm is at most equal to this extended product of layers' norms, and that there is equality if the parameters are normalized. Since we also know from Section 3.2 that within any weak rescaling-equivalence class, there are normalized parameters, we will deduce that the mixed path-norm is equal to the minimum value of the product over weakly rescaling-equivalent parameters.

Theorem 3.3.2 ([Gonon et al., 2024a])**.** *Consider $q \in (0, \infty)$ and $r \in (0, \infty]$. For every DAG ReLU network (Definition 2.2.2) and every parameters θ:*

$$\|\Phi(\theta)\|_{q,r} \leqslant \Pi_{q,r}(\theta).$$

If θ is q-normalized (Definition 3.2.1) then:

$$\|\Phi(\theta)\|_{q,r} = \Pi_{q,r}(\theta) = \left\| \left(\left\| \begin{pmatrix} \theta^{\to v} \\ b_v \end{pmatrix} \right\|_q \right)_{v\in N_{out}} \right\|_r.$$

In particular, we have

$$\|\Phi(\theta)\|_{q,r} = \min_{\theta' \sim_{WR} \theta} \Pi_{q,r}(\theta').$$

The proof is not particularly interesting and is deferred to Appendix B.2.

Conversely, even in the simple case of a LFCN, there is in general no constant $C \geqslant 1$ such that for every parameters θ:

$$\Pi_{q,r}(\theta) \leqslant C\|\Phi(\theta)\|_{q,r}.$$

Indeed, the product of layers' norms can be arbitrarily large while the path-norm is zero. This is illustrated in Figure 3.2.

This section has shown that the path-lifting $\Phi(\theta)$ provides a Lipschitz bound that is often strictly smaller than the one based on the product of layers' norms of the layers. This is because the path-lifting is invariant under rescaling symmetries of the network, while the product of layers' norms is not. This will again occur in the rest of the thesis, so I illustrated this phenomenon in Figure 3.3.

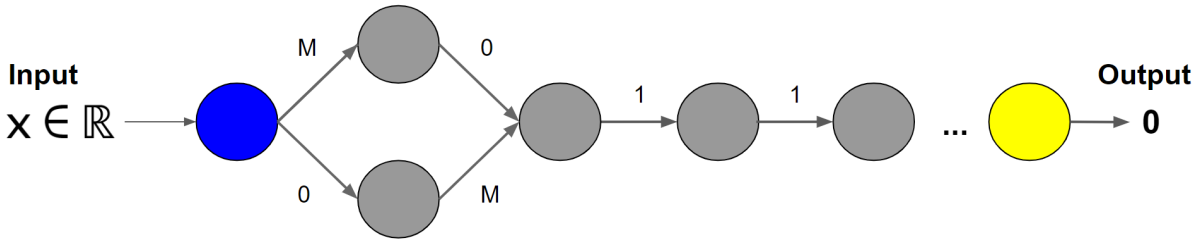


Figure 3.2: A network which path-norm is zero while the product of layers' norms scales as M^2.

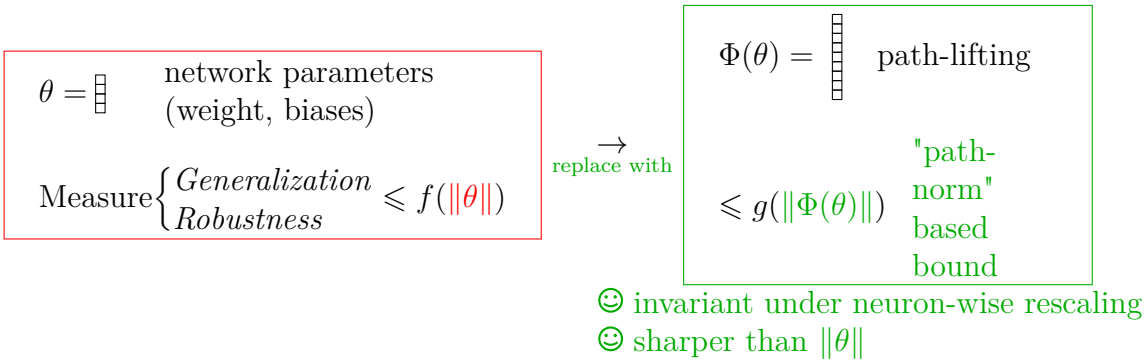


Figure 3.3: Quite often, it is better to consider things in terms of $\Phi(\theta)$ rather than in terms of θ because $\Phi(\theta)$ is invariant to neuron-wise rescaling symmetries of R_θ, while θ is not.

3.4 Lipschitzness in θ

Another important challenge about neural networks is to upper bound as tightly as possible the distances between the realizations $R_\theta, R_{\theta'}$ with parameters θ, θ' when evaluated at x, in terms of a (pseudo-)distance $d(\theta, \theta')$ and a constant C_x:

$$\|R_\theta(x) - R_{\theta'}(x)\|_1 \leqslant C_x d(\theta, \theta').$$

Such an inequality could indeed be crucially leveraged to derive generalization bounds [Neyshabur et al., 2018] or theoretical guarantees about compression [Gonon et al., 2023a]. Indeed, a typical way to achieve compression is to find a pruned or quantized version θ' of θ such that $R_{\theta'}$ makes similar predictions to R_θ. Since θ' has fewer nonzero parameters (if pruned) or fewer bits (if quantized), it can reduce the needed memory and computational resources to run the network. Whether or not the prediction $R_{\theta'}(x)$ made by the compressed parameters is close to the original prediction $R_\theta(x)$ motivates the study of the Lipschitz constant of $\theta \mapsto R_\theta(x)$.

Lipschitzness in θ as above is for example known with

$$d(\theta, \theta') := \|\theta - \theta'\|_\infty, \qquad C_x := (W\|x\|_\infty + 1)WL^2R^{L-1}, \tag{3.5}$$

in the case of a layered fully-connected neural network $R_\theta(x) = M_L\,\mathrm{ReLU}(M_{L-1}\ldots\mathrm{ReLU}(M_1 x))$ with L layers, maximal width W, and with weight matrices M_ℓ having some operator norm bounded by R [Gonon et al., 2023a, Theorem III.1 with $p = \infty$ and $q = 1$][Neyshabur

et al., 2018, Berner et al., 2020]. This known bound is however not satisfying at least for two reasons:

- it is **not invariant under neuron-wise rescalings** of the parameters θ that leave unchanged its realization R_θ, leading to crude dependencies in R and L and to cases where this bound is automatically vacuous (Section 1.1.2); and
- it **only holds for simple LFCNs**, but not for more practical networks that would include pooling, skip connections, etc.

To circumvent these issues, Section 3.4.1 introduces a natural (rescaling-invariant) metric based on the path-lifting that provides a Lipschitz property in θ. Section 3.4.2 then compares this new Lipschitz bound to the standard one based on the product of layers' norms. Section 3.4.3 shows how to compute this new Lipschitz bound in a single forward-pass in practical cases of interest such as quantization or pruning. Section 3.4.4 contains the proof of the Lipschitz bound. In the next section, I will show a first concrete application of this Lipschitz bound to compression of neural networks.

3.4.1 Main Result

For parameters θ, recall that (Definition 2.3.3) $\Phi^I(\theta)$ and $\Phi^H(\theta)$ denote the sub-vectors of $\Phi(\theta)$ corresponding to the coordinates associated with paths starting respectively from input and hidden neurons. In particular, $\Phi(\theta)$ is the concatenation of $\Phi^I(\theta)$ and $\Phi^H(\theta)$ (Definition 2.3.3).

Theorem 3.4.1 ([Gonon et al., 2024b])**.** *Consider a ReLU neural network as in Definition 2.2.2, with output dimension equal to one. Consider associated parameters θ, θ'. If $\theta_i \theta'_i \geqslant 0$ for every coordinate i, we have for every input x:*

$$|R_\theta(x) - R_{\theta'}(x)| \leqslant \|x\|_\infty \|\Phi^I(\theta) - \Phi^I(\theta')\|_1 + \|\Phi^H(\theta) - \Phi^H(\theta')\|_1. \tag{3.6}$$

Moreover, for every neural network architecture, there are parameters $\theta \neq \theta'$ and an input x such that Equation (3.6) is an equality.

I refer the reader to the discussion after Lemma 3.3.1 to understand why this is the ℓ^1-path-norm that appears in the bound. Note that since $\|x\|_\infty \leqslant \|x\|_q$ for any $q \in [1, \infty)$, it also implies a bound for any ℓ^q-norm on the input x.

In the specific case of *LFCNs*, there is also another Lipschitz bound based on the product of layers' norms. I will show in Section 3.4.2 Theorem 3.4.1 is strictly finer. This is because Theorem 3.4.1 is invariant under rescaling symmetries of the network, while the product of layers' norms is not.

The proof of Theorem 3.4.1 is deferred to Section 3.4.4. *Don't miss it, this is my favorite one.*

The assumption of parameters with the same sign cannot be simply removed: see Figure 3.4 for a counterexample.

Theorem 3.4.1 is intentionally stated with scalar output to allow the reader to deduce the result with multidimensional output using their favorite norm. The next corollary consider classical ℓ^q-norms on the output, and states the bound with the simpler pseudo-metric $d(\theta, \theta') := \|\Phi(\theta) - \Phi(\theta')\|_1$, called the *$\ell^1$-path-metric*, by analogy with the *ℓ^1-path-norm* $\|\Phi(\theta)\|_1$.

Figure 3.4: Counter-example to Theorem 3.4.1 when the parameters have opposite signs. If the hidden neurons are ReLU neurons, the left network implements $R_\theta(x) = \mathrm{ReLU}(x)$ (with $\theta = (1 \quad 1)^T$) and the right network implements $R_{\theta'}(x) = -\mathrm{ReLU}(-x)$ (with $\theta' = (-1 \quad -1)^T$). Equation (3.6) does not hold since there is a single path and the product of the weights along this path is equal to one in both cases, so that $\Phi(\theta) = \Phi(\theta') = 1$ while these two functions are nonzero and have disjoint supports.

Corollary 3.4.1 ([Gonon et al., 2024b])**.** *Consider an exponent $r \in [1, \infty)$ and a ReLU neural network as in Definition 2.2.2. Consider associated parameters θ, θ'. If for every coordinate i, it holds $\theta_i \theta'_i \geqslant 0$, then for every input $x \in \mathbb{R}^{d_{in}}$:*

$$\|R_\theta(x) - R_{\theta'}(x)\|_r \leqslant \max(\|x\|_\infty, 1)\|\Phi(\theta) - \Phi(\theta')\|_1.$$

I will show in Section 3.4.3 that this bound can be easily computed in a single forward-pass.

Proof of Corollary 3.4.1. By definition of the model, it holds:

$$\|R_\theta(x) - R_{\theta'}(x)\|_q^q = \sum_{v \in N_{\text{out}}} |v(\theta, x) - v(\theta', x)|^q.$$

Recall that $\Phi^{\to v}$ is the path-lifting associated with the sub-graph $G^{\to v}$ (Definition 2.3.3). By Theorem 3.4.1, it holds:

$$|v(\theta, x) - v(\theta', x)|^q \leqslant \max(\|x\|_\infty^q, 1)\|\Phi^{\to v}(\theta) - \Phi^{\to v}(\theta')\|_1^q.$$

Since $\Phi(\theta) = (\Phi^{\to v}(\theta))_{v \in N_{\text{out}}}$, this implies:

$$\|R_\theta(x) - R_{\theta'}(x)\|_q^q \leqslant \max(\|x\|_\infty^q, 1)\|\Phi(\theta) - \Phi(\theta')\|_1^q. \qquad \square$$

3.4.2 Comparison with bounds directly expressed in terms of θ.

In the special case of a LFCN, it is also known in the literature that for every input x and every parameters θ, θ' (even with different signs) of a LFCN with L affine layers and $L+1$ layers of neurons, $N_0 = N_{\text{in}}, \dots, N_L = N_{\text{out}}$, width $W := \max_{0 \leqslant \ell \leqslant L} |N_\ell|$, and each matrix having some operator norm bounded by $R \geqslant 1$, it holds [Gonon et al., 2023a, Theorem III.1 with $p = q = \infty$ and $D = \|x\|_\infty$][Neyshabur et al., 2018, Berner et al., 2020]:

$$\|R_\theta(x) - R_{\theta'}(x)\|_1 \leqslant (W\|x\|_\infty + 1)WL^2R^{L-1}\|\theta - \theta'\|_\infty. \tag{3.7}$$

I prove in Appendix B that the ℓ^1-path-metric is smaller than the following non-invariant metric defined in terms of the raw parameters:

$$\|\Phi(\theta) - \Phi(\theta')\|_1 \leqslant LW^2R^{L-1}\|\theta - \theta'\|_\infty.$$

This time, there is no equality for normalized parameters, as it was the case when comparing mixed path-norms to product of layers' norms. In particular, the left-hand side is not equal to the minimum of the right-hand side over all possible weakly rescaling-equivalent

parameters. Indeed, the right-hand side is already a worst case upper bound of a finer quantity defined in terms of the raw parameters θ. This finer quantity is not as simple as $\|\theta - \theta'\|_\infty$ so I leave it to the supplemental material of this chapter (Lemma B.3.2).

Thanks to the latter inequality, I show in Appendix B.3 that Theorem 3.4.1 implies for any θ, θ' (without restrictions on the signs) a slightly better bound than Equation (3.7) as soon as the LFCN has at least $L \geqslant 2$ affine layers. This is the second example of the principle illustrated in Figure 3.3: bounds based on the path-lifting $\Phi(\theta)$ are often sharper than bounds directly based on the raw parameters θ.

3.4.3 Computation of the ℓ^1-path-metric in two forward-passes.

Besides its theoretical interest, the proposed pseudo-distance $d(\theta, \theta') = \|\Phi(\theta) - \Phi(\theta')\|_1$ also has the desirable property of being easily computable in many practical cases of interest: when $\theta_i \theta'_i \geqslant 0$ and $|\theta'_i| \leqslant |\theta_i|$ for every coordinate i. This is satisfied if θ' is obtained by pruning θ since we either have $\theta'_i = \theta_i$ for unpruned coordinates or $\theta'_i = 0$ for pruned coordinates. This is also satisfied by quantizing θ by rounding towards zero.

Lemma 3.4.1. *Consider a ReLU neural network with DAG G. Consider the graph $\tilde{G}$ deduced from G but with max-pooling activations replaced by the identity. For a vector α, denote by $|\alpha|$ the vector deduced from α by applying $x \mapsto |x|$ coordinate-wise, and by $\mathbf{1}$ the input full of ones. We have for every θ, θ' such that $\theta_i \theta'_i \geqslant 0$ and $|\theta'_i| \leqslant |\theta_i|$ for every coordinate i:*

$$\|\Phi(\theta) - \Phi(\theta')\|_1 = \|\Phi(\theta)\|_1 - \|\Phi(\theta')\|_1 = \|R^{\tilde{G}}_{|\theta|}(\mathbf{1})\|_1 - \|R^{\tilde{G}}_{|\theta'|}(\mathbf{1})\|_1 = \|R^{\tilde{G}}_{|\theta|}(\mathbf{1}) - R^{\tilde{G}}_{|\theta'|}(\mathbf{1})\|_1 \tag{3.8}$$

where $R^{\tilde{G}}$ denotes the forward-pass in the network with graph $\tilde{G}$.

For simplicity, I stated Lemma 3.4.1 under the assumption $\theta_i \theta'_i \geqslant 0$ and $|\theta'_i| \leqslant |\theta_i$ for every coordinate i. Given the rescaling symmetries, an immediate corollary is that the path-metrics remains computable as soon as θ and θ' are only *rescaling-equivalent* to parameters satisfying these conditions. In general, computing the path-metrics without having to explicit the combinatorial $\Phi(\theta)$ remains open. But note that it can be bounded from below by the right-hand side of Equation (3.8), which is computable in two forward-passes:

$$\|\Phi(\theta) - \Phi(\theta')\|_1 \geqslant \|\Phi(\theta)\|_1 - \|\Phi(\theta')\|_1.$$

We will not use that, but let me mention that it can also be bounded from above by first normalizing θ, θ' and using mixed norms on the parameters, using results[2] such as Lemma C.2.2.

Proof of Lemma 3.4.1. It is heavily relies on Theorem 3.1.1. First, because $\theta_i \theta'_i \geqslant 0$ for every coordinate i, we have for every path p: $\Phi_p(\theta)\Phi_p(\theta') \geqslant 0$ so that $|\Phi_p(\theta) - \Phi_p(\theta')| = |\Phi_p(|\theta|) - \Phi_p(|\theta'|)|$ and:

$$\|\Phi(\theta) - \Phi(\theta')\|_1 = \|\Phi(|\theta|) - \Phi(|\theta'|)\|_1 = \sum_{p \in \mathcal{P}} |\Phi_p(|\theta|) - \Phi_p(|\theta'|)|.$$

[2] In Lemma C.2.2, the upper-bound contains a maximum of some sum, over *all* paths, which is combinatorial. However, this maximum sum can be further bounded by the depth times the maximum over all neurons of the term in the sum. This reduces to classical mixed (∞, q)-norms on the parameters

Because $|\theta_i| \geqslant |\theta'_i|$ for every coordinate i, we have $\Phi_p(|\theta|) \geqslant \Phi_p(|\theta'|)$ for every path p. Therefore, we have:

$$\begin{aligned}\|\Phi(\theta) - \Phi(\theta')\|_1 &= \sum_{p\in\mathcal{P}} \Phi_p(|\theta|) - \Phi_p(|\theta'|) \\ &= \left(\sum_{p\in\mathcal{P}} \Phi_p(|\theta|)\right) - \left(\sum_{p\in\mathcal{P}} \Phi_p(|\theta'|)\right) \\ &= \left(\sum_{p\in\mathcal{P}} |\Phi_p(\theta)|\right) - \left(\sum_{p\in\mathcal{P}} |\Phi_p(\theta')|\right) \\ &= \|\Phi(\theta)\|_1 - \|\Phi(\theta')\|_1. \end{aligned} \tag{3.9}$$

According to Theorem 3.1.1, it holds for every parameters θ:

$$\|\Phi(\theta)\|_1 = \|R^{\tilde{G}}_{|\theta|}(\mathbf{1})\|_1.$$

Therefore, we have

$$\|\Phi(\theta) - \Phi(\theta')\|_1 = \|R^{\tilde{G}}_{|\theta|}(\mathbf{1})\|_1 - \|R^{\tilde{G}}_{|\theta'|}(\mathbf{1})\|_1.$$

The latter is also equal to $\|R^{\tilde{G}}_{|\theta|}(\mathbf{1}) - R^{\tilde{G}}_{|\theta'|}(\mathbf{1})\|_1$ because $|\theta_i| \geqslant |\theta'_i|$ for every coordinate i implies that for every neuron v:

$$v^{\tilde{G}}(|\theta|, \mathbf{1}) \geqslant v^{\tilde{G}}(|\theta'|, \mathbf{1}) \geqslant 0$$

so that

$$\begin{aligned}\|R^{\tilde{G}}_{|\theta|}(\mathbf{1})\|_1 - \|R^{\tilde{G}}_{|\theta'|}(\mathbf{1})\|_1 &= \sum_{v\in N_{\text{out}}} |v^{\tilde{G}}(|\theta|, \mathbf{1})| - |v^{\tilde{G}}(|\theta'|, \mathbf{1})| \\ &= \sum_{v\in N_{\text{out}}} v^{\tilde{G}}(|\theta|, \mathbf{1}) - v^{\tilde{G}}(|\theta'|, \mathbf{1}) \\ &= \sum_{v\in N_{\text{out}}} \left|v^{\tilde{G}}(|\theta|, \mathbf{1}) - v^{\tilde{G}}(|\theta'|, \mathbf{1})\right| \\ &= \|R^{\tilde{G}}_{|\theta|}(\mathbf{1}) - R^{\tilde{G}}_{|\theta'|}(\mathbf{1})\|_1.\end{aligned}$$

This proves the result. □

3.4.4 Proof of Theorem 3.4.1

Let me now turn to the proof of Theorem 3.4.1. *This is my favorite proof of this thesis.* The goal is to control the distance between two functions R_θ and $R_{\theta'}$ in terms of the ℓ^1-distance between their path-liftings $\Phi(\theta)$ and $\Phi(\theta')$. What I like is that this consists of relating two geometries in very different spaces (the functions live in an infinite-dimensional vector space, while the path-liftings live in a finite-dimensional one), and this is made possible by carefully navigating in the path-lifting space by following a trajectory that connects $\Phi(\theta)$ to $\Phi(\theta')$ and whose geometry respects the geometry in the function space.

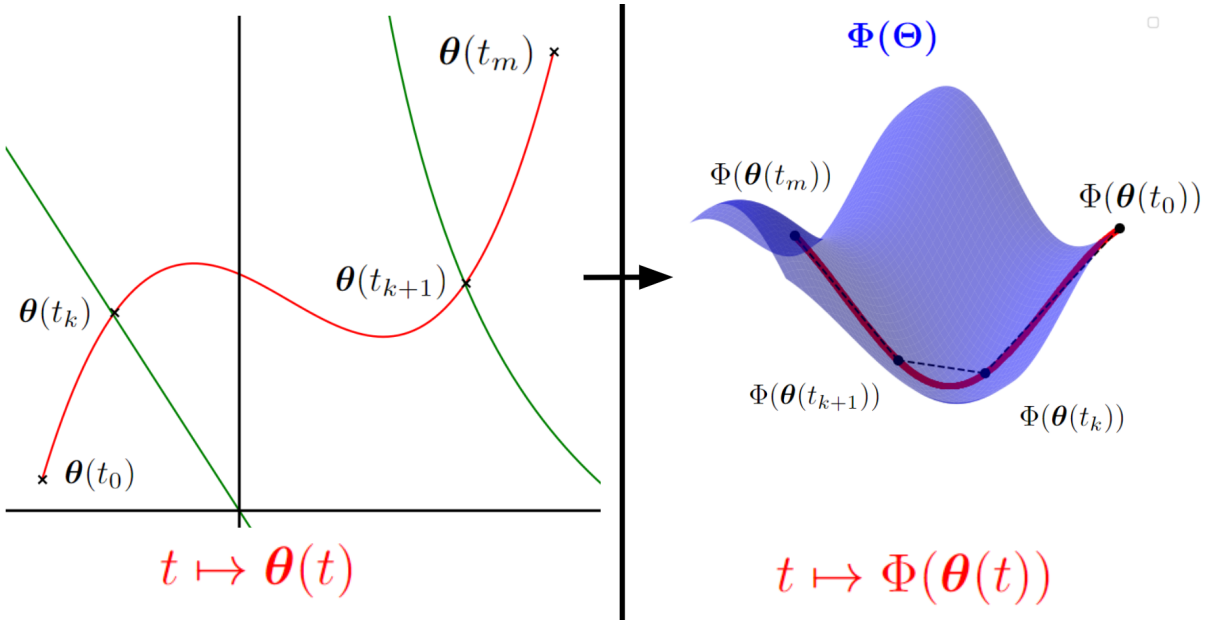


Figure 3.5: Illustration of the proof of Theorem 3.4.1, see the end of Section 3.4 for an explanation.

Proof sketch of Theorem 3.4.1 The output of a ReLU neuron in the d-th affine layer d of a LFCN is a piecewise polynomial function of the parameters θ of degree at most d (consequence of Equation (2.6)) [Bona-Pellissier et al., 2022, consequence of Propositions 1 and 2].

Given an input x, the proof of Theorem 3.4.1 consists in defining a trajectory $t \in [0,1] \to \theta(t) \in \Theta$ (**red** curve in Figure 3.5) that starts at θ, ends at θ', and with finitely many breakpoints $0 = t_0 < t_1 < \cdots < t_m = 1$ such that the path-activations $A(\theta(t), x)$ are constant on the open intervals $t \in (t_k, t_{k+1})$. Each breakpoint corresponds to a value where at least one activation changes in the neighborhood of $\theta(t)$. For instance, in the left part of Figure 3.5, the straight **green** line and quadratic **green** curve correspond to a change of activation of a ReLU neuron (for a given input x to the network) respectively in the first and second layer.

With such a trajectory, given the key property (2.4), each quantity $|R_{\theta(t_k)}(x) - R_{\theta(t_{k+1})}(x)|$ can be controlled in terms of $\|\Phi(\theta(t_k)) - \Phi(\theta(t_{k+1}))\|_1$, and if the path is "nice enough", then this control can be extended globally from t_0 to t_m.

There are two obstacles: 1) proving that there are finitely many breakpoints t_k as above (think of $t \mapsto t^{n+2}\sin(1/t)$ that is n-times continuously differentiable but still crosses $t = 0$ an infinite number of times around zero), and 2) proving that the length $\sum_{k=1}^{m} \|\Phi(\theta(t_k)) - \Phi(\theta(t_{k+1}))\|_1$ of the broken line with vertices $\Phi(\theta(t_k))$ (dashed line on the right part of Figure 3.5) is bounded from above by $\|\Phi(\theta) - \Phi(\theta')\|_1$ times a reasonable factor. Trajectories satisfying these two properties are called "admissible" trajectories. The first property is true as soon as the trajectory $t \mapsto \theta(t)$ is smooth enough (analytic, say). The second is true *with factor one* thanks to a monotonicity property of the chosen trajectory. The core of the proof consists in exhibiting a trajectory with these properties.

Admissible trajectory: definition. Given any input vector x and two parameters θ, θ', I define an *x-admissible trajectory*[3] betweeen θ and θ' as any continuous map $t \in [0,1] \mapsto \theta(t)$ such that for every $t \in [0,1]$, the vector $\theta(t)$ corresponds to parameters associated with the considered network architecture, with the boundary conditions $\theta(0) = \theta$ and $\theta(1) = \theta'$, and with the additional "x-admissibility property" corresponding to the existence of *finitely many* breakpoints $0 = t_0 < t_1 < \cdots < t_m = 1$ such that the path-activations matrix (see Definition 2.3.3) $t \in [0,1] \mapsto A(\theta(t), x)$ is constant on each interval (t_k, t_{k+1}) and such that for every path p of the graph, using the shorthand $\theta_k := \theta(t_k)$, the "reverse triangle inequality" holds (which is then, of course, an equality):

$$\sum_{k=1}^{m} |\Phi_p(\theta_k) - \Phi_p(\theta_{k-1})| \leqslant |\Phi_p(\theta_m) - \Phi_p(\theta_0)|. \tag{3.10}$$

Finding an admissible trajectory is enough.

Lemma 3.4.2. *Consider an input vector x and two parameters θ, θ'. If $t \in [0,1] \mapsto \theta(t)$ is an x-admissible trajectory between θ and θ' then*

$$|R_\theta(x) - R_{\theta'}(x)| \leqslant \|x\|_\infty \|\Phi^I(\theta) - \Phi^I(\theta')\|_1 + \|\Phi^H(\theta) - \Phi^H(\theta')\|_1. \tag{3.11}$$

Proof. In this proof, I denote by convention $x_u := 1$ for any u that is not an input neuron. Recall that p_0 denotes the first neuron of a path p, and x_{p_0} is the coordinate of x for neuron p_0. Theorem 2.4.2 shows that for every parameters θ and every input x, we have

$$R_\theta(x) = \sum_{p \in \mathcal{P}} x_{p_0} a_p(\theta, x) \Phi_p(\theta),$$

so for every $k \in \{1, \dots, m\}$ and every $t_{k-1} < t' < t < t_k$, we get:

$$R_{\theta(t)}(x) - R_{\theta(t')}(x) = \sum_{p \in \mathcal{P}} x_{p_0} \left(a_p(\theta(t), x) \Phi_p(\theta(t)) - a_p(\theta(t'), x) \Phi_p(\theta(t')) \right).$$

Since both t and t' belong to the same interval (t_{k-1}, t_k) and since $t \mapsto \theta(t)$ is an admissible trajectory, the path-activations $a_p(\theta(t), x) = a_p(\theta(t'), x)$ are the same for every path p and we get:

$$R_{\theta(t)}(x) - R_{\theta(t')}(x) = \sum_{p \in \mathcal{P}} x_{p_0} a_p(\theta(t), x) \left(\Phi_p(\theta(t)) - \Phi_p(\theta(t')) \right).$$

Recall that the set of paths $\mathcal{P}$ is partitioned into the sets $\mathcal{P}_I$ and $\mathcal{P}_H$ of paths starting respectively from input and hidden neurons. By the convention taken in this proof, for $p \in \mathcal{P}_H$, it holds $x_{p_0} = 1$. Thus:

$$\begin{aligned} R_{\theta(t)}(x) - R_{\theta(t')}(x) = & \sum_{p \in \mathcal{P}_I} x_{p_0} a_p(\theta(t), x) \left(\Phi_p(\theta(t)) - \Phi_p(\theta(t')) \right) \\ & + \sum_{p \in \mathcal{P}_H} a_p(\theta(t), x) \left(\Phi_p(\theta(t)) - \Phi_p(\theta(t')) \right). \end{aligned}$$

[3] While the standard terminology for such a map $t \mapsto \theta(t)$ is rather "path" than "trajectory", we choose "trajectory" to avoid possible confusions with the notion of "path" of a DAG associated with a neural network.

Recall that a path-activation is always equal to 0 or 1 by definition, so that:

$$\begin{aligned}\left|R_{\theta(t)}(x) - R_{\theta(t')}(x)\right| &\leqslant \sum_{p\in\mathcal{P}_I} |x_{p_0}|\,|\Phi_p(\theta(t)) - \Phi_p(\theta(t'))| + \sum_{p\in\mathcal{P}_H} |\Phi_p(\theta(t)) - \Phi_p(\theta(t'))| \\ &\leqslant \|x\|_\infty \|\Phi^I(\theta(t)) - \Phi^I(\theta(t'))\|_1 + \|\Phi^H(\theta(t)) - \Phi^H(\theta(t'))\|_1.\end{aligned}$$

Considering the limits $t \to t_k$ and $t' \to t_{k-1}$ gives by continuity of both $\theta \mapsto R_\theta(x)$ and $\theta \mapsto \Phi(\theta)$:

$$\left|R_{\theta_k}(x) - R_{\theta_{k-1}}(x)\right| \leqslant \|x\|_\infty \|\Phi^I(\theta_k) - \Phi^I(\theta_{k-1})\|_1 + \|\Phi^H(\theta_k) - \Phi^H(\theta_{k-1})\|_1.$$

Since $\theta = \theta_0$ and $\theta' = \theta_m$, using the triangle inequality yields:

$$|R_\theta(x) - R_{\theta'}(x)| \leqslant \|x\|_\infty \sum_{k=1}^m \|\Phi^I(\theta_k) - \Phi^I(\theta_{k-1})\|_1 + \sum_{k=1}^m \|\Phi^H(\theta_k) - \Phi^H(\theta_{k-1})\|_1. \quad (3.12)$$

See Figure 3.5 for an illustration of what is happening here. By definition, since the trajectory is x-admissible, we have by Equation (3.10)

$$\sum_{k=1}^m \|\Phi^I(\theta_k) - \Phi^I(\theta_{k-1})\|_1 \leqslant \|\Phi^I(\theta_m) - \Phi^I(\theta_0)\|_1 = \|\Phi^I(\theta') - \Phi^I(\theta)\|_1$$

and

$$\sum_{k=1}^m \|\Phi^H(\theta_k) - \Phi^H(\theta_{k-1})\|_1 \leqslant \|\Phi^H(\theta_m) - \Phi^H(\theta_0)\|_1 = \|\Phi^H(\theta') - \Phi^H(\theta)\|_1.$$

With Equation (3.12), this proves Equation (3.11). □

Construction of an admissible trajectory. In the formal proof of Theorem 3.4.1 I will show that it is enough to establish the result when all the coordinates of θ, θ' are nonzero.

Definition 3.4.1. *Consider two parameters θ, θ' with only nonzero coordinates. For every $t \in [0,1]$ and every i, define the following trajectory*[4] *$t \mapsto \theta(t)$ between θ and θ':*

$$(\theta(t))_i = \operatorname{sgn}(\theta_i)|\theta_i|^{1-t}|\theta'_i|^t, \quad (3.13)$$

where $\operatorname{sgn}(y) := \mathbb{1}_{y>0} - \mathbb{1}_{y<0} \in \{-1, 0, +1\}$ *for any $y \in \mathbb{R}$.*

Observe that the trajectory in Equation (3.13) is well-defined since the coordinates of θ and θ' are nonzero by assumption. As proved in the next lemma, this trajectory has indeed finitely many breakpoints where the path-activations change. This is basically because for every coordinate i, the trajectory $t \in [0,1] \to (\theta(t))_i$ is analytic[5]. As a consequence, the set of t's where a coordinate of the path-activations matrix $A(\theta(t), x)$ does change can be realized as a set of zeroes of an analytic function on $\mathbb{C}$, and since these zeroes must be isolated, there could only be finitely of them in the compact $[0,1]$, except when the set of zeroes is the whole $[0,1]$.

[4] This trajectory is linear in log-parameterization: for every i, the map $t \mapsto \ln(|(\theta(t))_i|)$ is linear.

[5] A function $f : C \mapsto \mathbb{R}$ is analytic on a *closed* subset $C \subset \mathbb{R}$ if there exists an open set $C \subset O \subset \mathbb{R}$ such that f is the restriction to C of a function that is analytic on O.

Lemma 3.4.3. *Consider $n \in \mathbb{N}_{>0}$ inputs $X = (x_1, \dots, x_n) \in (\mathbb{R}^{d_{in}})^n$. For parameters θ, θ' with only nonzero coordinates, consider the trajectory $t \in [0,1] \mapsto \theta(t)$ defined in Equation (3.13). There exists finitely many breakpoints $0 = t_0 < t_1 < \cdots < t_m = 1$ such that for every $i = 1, \dots, n$, the path-activations matrix $t \in [0,1] \mapsto A(\theta(t), x_i)$ is constant on each interval (t_k, t_{k+1}).*

Proof of Lemma 3.4.3. After showing that the result for arbitrary n follows from the result for $n = 1$, I establish the latter by an induction on a topological sorting of the graph G.

Reduction to $n = 1$. If for every $i = 1, \dots, n$, we have a finite family of breakpoints $(t_k^i)_k$, then the union of these families gives a finite family of breakpoints that works for every i. It is then sufficient to prove that for a *single* arbitrary input x, there are finitely many breakpoints $0 = t_0 < t_1 < \cdots < t_m = 1$ such that the path-activations matrix $t \in [0,1] \mapsto A(\theta(t), x)$ remains constant on each interval (t_k, t_{k+1}).

For the rest of the proof, consider a single input x, and define for any neuron v the property

$$\begin{aligned} &\text{there are finitely many breakpoints } 0 = t_0 < t_1 < \cdots < t_m = 1 \text{ such that for every } k: \\ &\qquad \text{the map } t \in [t_k, t_{k+1}] \mapsto v(\theta(t), x) \text{ is analytic,} \\ &\text{and the functions } t \mapsto a_v(\theta(t), x), t \mapsto a_{u \to v}(\theta(t), x), \text{for each } u \in \mathrm{ant}(v), \text{are constant on } (t_k, t_{k+1}) \end{aligned} \tag{3.14}$$

Reduction to proving Property (3.14) for every neuron v. I will soon prove that Property (3.14) holds for every neuron v. Let me explain why this is enough to reach the desired conclusion. By the same argument as in the reduction to $n = 1$, the union of the breakpoints associated to all neurons yields finitely many intervals such that, on each interval, *all functions* $t \mapsto a_v(\theta(t), x)$, $v \in N$, and $a_{u \to v}(\theta(t), x)$, $u \in \mathrm{ant}(v)$, are constant. By Definition 2.3.3 this implies that $t \mapsto A(\theta(t), x)$ is constant on each corresponding open interval.

Proof of Property (3.14) for every neuron v by induction on a topological sorting [Cormen et al., 2009, Section 22.4] of the graph. We start with input neurons v since by Definition 2.2.2, these are the ones without antecedents so they are the first to appear in a topological sorting.

Initialization: Property (3.14) for input neurons. For any input neuron v, it holds by Definition 2.2.2 $v(\theta, x) = x_v$ that is constant in θ. Thus $t \in [0,1] \mapsto v(\theta(t), x)$ is trivially analytic. Since v is an input neuron, it has no antecedent, and by Definition 2.3.3 we have $a_v(\theta, x) := 1$. This shows that Property (3.14) holds for input neurons.

Induction: Now, consider a non-input neuron v and assume Property (3.14) to hold for every neuron coming before v in the considered topological sorting. Since every antecedent of v must come before v in the topological sorting, there are finitely many breakpoints $0 = t_0 < t_1 < \cdots < t_m = 1$ such that for every $u \in \mathrm{ant}(v)$ and every k, the map $t \in [t_k, t_{k+1}] \mapsto u(\theta(t), x)$ is analytic. We distinguish three cases depending on the activation function of neuron v.

- **Case of neurons with the identity as activation function.** By Definition 2.2.2 $v(\theta(t), x) = b_v + \sum_{u \in \mathrm{ant}(v)} u(\theta(t), x)\theta(t)^{u \to v}$ and for every k it is clear that it is analytic as it is the case for each $t \in [t_k, t_{k+1}] \mapsto u(\theta(t), x)$ by induction, and it is also the case for $t \in [t_k, t_{k+1}] \mapsto \theta(t)^{u \to v}$ by definition (Equation (3.13)). Since v is a neuron with the identity as activation function, Definition 2.3.3 implies that $a_{u \to v}(\theta(t), x) = a_v(\theta(t), x) = 1$ for every t. This establishes Property (3.14) for v.

• **Case of a ReLU neuron.** By Definition 2.2.2: $v(\theta,x) = \mathrm{ReLU}(\mathrm{pre}_v(\theta,x))$ where the pre-activation of v is $\mathrm{pre}_v(\theta,x) := b_v + \sum_{u\in\mathrm{ant}(v)} u(\theta,x)\theta^{u\to v}$. Reasoning as in the previous case, the induction hypothesis implies that for every k the function $t \in [t_k, t_{k+1}] \mapsto \mathrm{pre}_v(\theta(t),x)$ is analytic. We distinguish two sub-cases:

– If this function is identically zero then $t \in [t_k,t_{k+1}] \mapsto v(\theta(t),x)$ is null, so it is analytic, and by Definition 2.3.3 $a_{u\to v}(\theta(t),x) = a_v(\theta(t),x) = \mathbb{1}_{v(\theta,x)>0} = 0$ for every $u \in \mathrm{ant}(v)$;

– Otherwise this analytic function can only vanish a finite number of times on the compact $[t_k,t_{k+1}]$: there are times $t_k = s_0 < s_1 < \cdots < s_n = t_{k+1}$ such that for each j, $s \in (s_j,s_{j+1}) \mapsto \mathrm{pre}_v(\theta(s),x)$ has constant (nonzero) sign and can be extended into an analytic function on $\mathbb{C}$. For each segment (s_j,s_{j+1}) where the sign is negative, we deduce that for every $s \in [s_j,s_{j+1}]$ we have $v(\theta(s),x) = 0$, hence by Definition 2.3.3, $a_v(\theta(s),x) = a_{u\to v}(\theta(s),x) = 0$ for every $u \in \mathrm{ant}(v)$; on the other segments, we have $v(\theta(s),x) = \mathrm{pre}_v(\theta(s),x)$ for every $s \in [s_j,s_{j+1}]$, and therefore $a_v(\theta(s),x) = a_{u\to v}(\theta(s),x) = 1$ for every $s \in (s_j,s_{j+1})$ and $u \in \mathrm{ant}(v)$.

Overall, on all the resulting (finitely many) segments, we obtain all the properties establishing that Property (3.14) indeed holds for v.

• **Case of a K-max-pooling neuron.** Recall that by Definition 2.2.2, the output of v is the K-th largest component of $\mathrm{pre}_v(\theta,x) = (u(\theta,x)\theta^{u\to v})_{u\in\mathrm{ant}(v)}$, with ties between antecedents decided by lexicographic order. Since each $t \in [t_k,t_{k+1}] \mapsto u(\theta(t),x)$ is analytic, and so does $t \mapsto \theta(t)^{u\to v}$, this is also the case of each coordinate of $\mathrm{pre}_v(\theta(t),x)$.

Consider any k. I am going to prove that there are finitely many breakpoints $t_k = s_0 < s_1 < \cdots < s_\ell = t_{k+1}$ such that on each interval (s_j,s_{j+1}), there is an antecedent $u \in \mathrm{ant}(v)$ such that

$$v(\theta(s),x) = u(\theta(s),x)\theta(s)^{u\to v}, \text{ for every } s \in (s_j,s_{j+1}).$$

By the same reasoning as above this will imply that Property (3.14) holds for v.

For any neurons $u \neq u' \in \mathrm{ant}(v)$, denote $\delta_{u,u'}(\theta) := u(\theta(t),x)\theta(t)^{u\to v} - u'(\theta(t),x)\theta(t)^{u'\to v}$ and let U be the set of $u \in \mathrm{ant}(v)$ such that: for each $u' \in \mathrm{ant}(v)$, either $t \mapsto \delta_{u,u'}(\theta(t))$ is not identically zero on $[t_k,t_{k+1}]$, or u is before u' in lexicographic order. With this definition, for each pair $u \neq u' \in U$, the function $t \in [t_k,t_{k+1}] \mapsto \delta_{u,u'}(\theta(t),x)$ is not identically zero and is analytic, so that there are only finitely many breakpoints $t_k = s_0^{u,u'} < s_1^{u,u'} < \cdots < s_{\ell(u,u')}^{u,u'} = t_{k+1}$ where it vanishes on the compact $[t_k,t_{k+1}]$. Considering the union over all pairs $u,u' \in U$ of these finite families of breakpoints, we get a finite family of breakpoint $t_k = s_0 < s_1 < \cdots < s_\ell = t_{k+1}$ such that on each interval (s_j,s_{j+1}), the ordering between the coordinates of $\mathrm{pre}_v(\theta(s),x)$ in U is strict and stays the same. To conclude, it is not hard to check that, by the definition of U and of $*$-max-pooling, the output of v only depends on the coordinates of $\mathrm{pre}_v(\theta(s),x)$ indexed by U. This yields the claim and concludes the proof of Lemma 3.4.3. □

For $y \in \mathbb{R}$, recall that here $\mathrm{sgn}(y) = \mathbb{1}_{y>0} - \mathbb{1}_{y<0} \in \{-1,0,+1\}$ and it is extended to vectors by applying it coordinate-wise.

Corollary 3.4.2. *Consider two parameters θ,θ' with nonzero coordinates and such that $\mathrm{sgn}(\theta) = \mathrm{sgn}(\theta')$. Then the trajectory defined in Equation (3.13) is x-admissible for every input vector x.*

Proof. First, the trajectory is well-defined since the coordinates are nonzero, and it satisfies the boundary conditions $\theta(0) = \theta$ and $\theta(1) = \theta'$ since the coordinates have the same signs.

Second, Lemma 3.4.3 proves that for every x, there are finitely many breakpoints $0 = t_0 < t_1 < \cdots < t_m = 1$ such that the path-activations matrix $t \in [0,1] \mapsto A(\theta(t), x)$ is constant on each interval (t_{k-1}, t_k).

It now only remains to prove that Equation (3.10) holds to prove that this is an x-admissible trajectory. Consider a path p. For a coordinate i of the parameters, I write $i \in p$ either if $i = p_0$ and p_0 is a hidden neuron, or if $i = e$ is an edge along the path p. Define $\operatorname{sgn}(p) := \prod_{i \in p} \operatorname{sgn}(\theta_i)$ and note that $\operatorname{sgn}(p) \neq 0$ since θ has only nonzero coordinates by assumption. Denote $|\theta|$ the vector deduced from θ by applying the absolute value coordinate-wise. It is easy to check by definition of the path-lifting Φ that for every $t \in [0,1]$:

$$\Phi_p(\theta(t)) = \operatorname{sgn}(p)\Phi_p(|\theta|)^{1-t}\Phi_p(|\theta'|)^t = \operatorname{sgn}(p)\Phi_p(|\theta(t_0)|)^{1-t}\Phi_p(|\theta(t_m)|)^t.$$

Denote by $a := \Phi_p(|\theta'|) = \Phi_p(|\theta(t_m)|)$ and by $b = \Phi_p(|\theta|) = \Phi_p(|\theta(t_0)|)$. The latter rewrites:

$$\Phi_p(\theta(t)) = \operatorname{sgn}(p) a^t b^{1-t}.$$

Thus, Equation (3.10) holds if, and only if,

$$\sum_{k=1}^{m} |\operatorname{sgn}(p)| \left| a^{t_k} b^{1-t_k} - a^{t_{k-1}} b^{1-t_{k-1}} \right| \leqslant |\operatorname{sgn}(p)| \, |a - b| \, .$$

Simplifying by $\operatorname{sgn}(p) \neq 0$, Equation (3.10) is equivalent to:

$$\sum_{k=1}^{m} \left| a^{t_k} b^{1-t_k} - a^{t_{k-1}} b^{1-t_{k-1}} \right| \leqslant |a - b| \, .$$

Let me now observe that $t \mapsto a^t b^{1-t}$ is monotonic and conclude. I only do so when $a \geqslant b$, the other case being similar. Since by definition, we also have a and b positive, it holds for $t > t'$

$$a^{t-t'} \geqslant b^{t-t'} \text{ that is equivalent to } a^t b^{1-t'} \geqslant a^t b^{1-t'}.$$

This is a telescopic sum:

$$\begin{aligned}\sum_{k=1}^{m} \left| a^{t_k} b^{1-t_k} - a^{t_{k-1}} b^{1-t_{k-1}} \right| &= \sum_{k=1}^{m} a^{t_k} b^{1-t_k} - a^{t_{k-1}} b^{1-t_{k-1}} \\ &= a^{t_m} b^{1-t_m} - a^{t_0} b^{1-t_0} = a - b = |a - b| \, .\end{aligned}$$

This shows Equation (3.10), proving that $t \mapsto \theta(t)$ is an admissible trajectory, and thus the result. □

Proof of Theorem 3.4.1. **Equality case.** Consider an arbitrary neural network architecture, an input neuron v_0 and a path $p = v_0 \to v_1 \to \dots v_d$. Consider θ (resp. θ') with only zero coordinates, except for $\theta^{v_\ell \to v_{\ell+1}} = a > 0$ (resp. $(\theta')^{v_\ell \to v_{\ell+1}} = b > 0$) for every $\ell \in [\![0, d-1]\!]$. Consider the input x to have only zero coordinates except for $x_{v_0} > 0$. It is easy to check that $R_\theta(x) = a^d x_{v_0}$ and $R_{\theta'}(x) = b^d x_{v_0}$. Since $\|x\|_\infty = x_{v_0}$,

$\|\Phi^I(\theta) - \Phi^I(\theta')\|_1 = |a^d - b^d|$ and $\|\Phi^H(\theta) - \Phi^H(\theta')\|_1 = 0$, this shows that Equation (3.6) is an equality for these parameters.

Proof of the inequality. By continuity of both handsides of (3.6) with respect to θ, θ', it is enough to prove the result when all coordinates of θ, θ' are nonzero, i.e., under the stronger assumption that $\theta_i \theta'_i > 0$ for every coordinate index i. Under this assumption, by Corollary 3.4.2, the trajectory $t \mapsto \theta(t)$ defined in Equation (3.13) is x-admissible for every input vector x. The conclusion follows by Lemma 3.4.2. □

3.5 A first application: pruning

I just showed that the path-lifting $\Phi(\theta)$ provides Lipschitz bounds that hold for ReLU networks with pooling, skip connections etc. In the special of a LFCN, these bounds are at least af fine as standard ones derived in terms of the raw parameters thanks to the symmetry invariance of the path-lifting.

Let me now turn to a practical application of the Lipschitz bound in θ: compression of neural networks via pruning. In the quest to make neural networks more efficient, pruning has emerged as a critical technique. As modern neural networks grow increasingly large and complex, they demand significant computational resources and memory, which can be prohibitive for deployment in real-world applications such as mobile devices, embedded systems, and large-scale data centers. Pruning addresses this by reducing the size of the network, which can lead to faster inference times, lower energy consumption, and reduced storage requirements, all while maintaining or even improving the network's accuracy [Hoefler et al., 2021, Cheng et al., 2023].

Pruning methods typically involve pruning (setting to zero) parameters that are deemed less important, based on certain criteria. A large number of criteria have been proposed [Hoefler et al., 2021, Cheng et al., 2023]: some are based on the magnitude of the weights in θ, others on the sensitivity of $R_\theta(x_i)$ for a given dataset $(x_i)_i$, and still others first or second order information on a loss score $\ell(R_\theta(x_i), y_i)$ for a dataset $(x_i, y_i)_i$ and a loss function ℓ that is used to measure how well $R_\theta(x)$ approximates y for a given input x and target y. There is no single pruning method that is universally superior, and the choice of method often depends on the specific network architecture, dataset, and task at hand.

However, one of the most widely-used techniques is magnitude pruning [Hoefler et al., 2021, Cheng et al., 2023], which prunes weights with the smallest magnitudes. This method, while seemingly simple, has proven remarkably effective and can scale easily to large networks.

The relevance of magnitude pruning is underscored by the Lottery Ticket Hypothesis [Frankle and Carbin, 2019]. This hypothesis posits that within a randomly initialized neural network, there exists a much smaller subnetwork — a "winning ticket" — that can be trained to achieve performance similar to the full network. Magnitude pruning drew attention as a method to find these winning tickets: the process to find these winning tickets typically involves training the network first, followed by magnitude pruning. This method has been shown to produce subnetworks that train faster, require less memory, and often generalize better, acting as a form of regularization. This shows that magnitude pruning manages to uncover subnetworks with interesting properties.

However, magnitude pruning is not invariant to rescalings of the network parameters. I will show that this may harm its performance in practice. This is where the path-lifting

$\Phi(\theta)$ comes into play: I will use it to design a pruning method that is rescaling-invariant and that is competitive to magnitude pruning.

3.5.1 Pruning method based on the path-lifting

Notion of pruned parameters. Considering a DAG network G as described in Section 2.2, and recall the notation $\mathbb{R}^G$ to denote the corresponding set of parameters (Definition 2.2.2). By definition, a pruned version θ' of $\theta \in \mathbb{R}^G$ is a "Hadamard" product $\theta' = s \odot \theta$, where $s \in \mathbb{R}^G$ is a binary vector with all of its coordinates in $\{0, 1\}$ and $\|s\|_0$ is "small". A standard pruning method consists in selecting s from a pre-trained parameter θ typically by pruning out (setting to zero) entries of θ with magnitude below some threshold. This is clearly not rescaling-invariant, as the ranking of the magnitude of certain coefficients can change when applying certain rescalings.

Proposed rescaling-invariant pruning criteria Given any θ, the parameters θ and $\theta' = s \odot \theta$ satisfy the assumptions of Theorem 3.4.1, hence for all input x we have $|R_\theta(x) - R'_\theta(x)| \leqslant \|\Phi(\theta) - \Phi(\theta')\|_1 \max(1, \|x\|_\infty)$. Defining $\Delta(\theta, s) := \|\Phi(\theta) - \Phi(s \odot \theta)\|_1$, a first reasonable global pruning criterion is then to aim at solving the following problem for a given k

$$\min_{s:\|s\|_0 \leqslant k} \Delta(\theta, s). \tag{3.15}$$

However, while (3.8) guarantees that, given s, the cost $\Delta(\theta, s)$ is computed in two forward-passes as

$$\Delta(\theta, s) = \|R^{\tilde{G}}_{|\theta|}(\mathbf{1}_d)\|_1 - \|R^{\tilde{G}}_{|s\odot\theta|}(\mathbf{1}_d)\|_1$$

(where $\tilde{G}$ is the same as the original one G but with max-pooling activations replaced by the identity), Problem (3.15) is combinatorial because the set $\{s : \|s\|_0 \leqslant k\}$ has a size that grows exponentially in k.

Instead, we can approximate the solution of Problem (3.15) by minimizing an upper-bound of $\Delta(\theta, s)$. For each *individual parameter coordinate* i, we will consider the cost of setting it to zero

$$\Phi\text{-}\texttt{Cost}(\theta, i) := \Delta(\theta, s_i) \tag{3.16}$$

where $s_i := \mathbf{1}_G - e_i$ with $\mathbf{1}_G \in \mathbb{R}^G$ the vector filled with ones and $e_i \in \mathbb{R}^G$ the i-th canonical vector.

Bounding $\Delta(\theta, s)$ using Φ-$\texttt{Cost}(\theta, i), i \in G$. Consider any subset $I \subseteq G$ to be potentially pruned out. When $s = s(I) := \mathbf{1}_G - \mathbf{1}_I$ with $\mathbf{1}_I = \sum_{i\in I} e_i$, then for any enumeration i_j, $1 \leqslant j \leqslant |I|$ of elements in I, denoting $s_j := \mathbf{1}_G - \sum_{\ell=1}^{j} e_{i_\ell} = \mathbf{1}_G - \mathbf{1}_{\cup_{\ell=1}^{j}\{i_\ell\}}$ (and

$s_0 := \mathbf{1}_G$) we have

$$\Delta(\theta, s) \underset{(3.8)}{=} \|\Phi(\theta)\|_1 - \|\Phi(s \odot \theta)\|_1 = \sum_{j=1}^{|I|} \|\Phi(s_{j-1} \odot \theta)\|_1 - \|\Phi(s_j \odot \theta)\|_1 \underset{(3.8)}{=} \sum_{j=1}^{|I|} \Delta(s_{j-1} \odot \theta, s_j)$$

$$= \sum_{j=1}^{|I|} \Phi\text{-}\texttt{Cost}(s_{j-1} \odot \theta, i_j) \tag{3.17}$$

$$\leqslant \sum_{j=1}^{|I|} \Phi\text{-}\texttt{Cost}(\theta, i_j). \tag{3.18}$$

Φ-Pruning Method. Instead of solving the combinatorial Problem (3.15), we will minimize the upper-bound given in Inequality (3.18). This is achieved via simple *reverse* hard thresholding:

1. compute $\Phi\text{-}\texttt{Cost}(\theta, i)$ for all i (two forward-passes per i via Equation (3.8));
2. given the targeted sparsity k, select the set I of cardinal $|G| - k$ containing the k indices corresponding to the *smallest values* of this cost.

By Inequality (3.18) and Theorem 3.4.1, the index set I thus selected is such that $\|s(I)\|_0 \leqslant k$ and

$$|R_\theta(x) - R_{s(I)\odot\theta}(x)| \leqslant \left(\sum_{i \in I} \Phi\text{-}\texttt{Cost}(\theta, i)\right) \|(x, 1)\|_\infty. \tag{3.19}$$

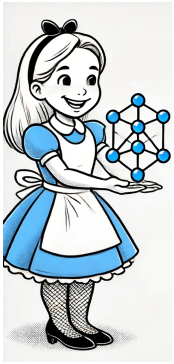
To the best of our knowledge, this is the first practical network pruning method invariant under rescaling symmetries that is endowed with guarantees on ReLU networks that contain skip connections, max-pooling, etc. The wide range of applicability of this method allows us to apply it to networks widely used in practice such as ResNets.

3.5.2 Experiments

To validate the approach I train a dense ResNet-18 on ImageNet-1k with standard hyperparameters (Appendix E.1). I prune some weights of the trained dense model with one of the following methods (recall the definition of Φ-costs in Equation (3.16)):

- *magnitude pruning (MP)*: pruning $p\%$ of the *smallest* weights in absolute value in each convolutional and fully-connected layer,
- *Φ-layerwise pruning (Φ-LP)*: pruning $p\%$ of the weights with the smallest Φ-costs in each convolutional and fully-connected layer,
- *Φ-global pruning (Φ-GP)*: *global* pruning of $p\%$ of the weights with the smallest Φ-costs.

Invariance under rescaling symmetries: MP versus Φ-pruning. A key difference between these pruning methods is that Φ-pruning is invariant to neuron-wise rescaling symmetries in ReLU networks, while MP is not. To illustrate the consequences of this lack of invariance, consider the following scenarios.



(a) Alice releasing original θ

(b) Malice releasing rescaled θ

Scenario 1 (Figure 3.6a). Imagine Alice, a researcher, trains a neural network and releases its parameters, θ. Bob, another researcher, receives these parameters and uses magnitude pruning (MP) to prune the smallest weights before retraining the network.

Scenario 2 (Figure 3.6b). Now consider a different scenario where Malice, a malicious actor, intervenes. Malice takes Alice's trained network, applies a rescaling to the weights, and then releases the rescaled parameters to Bob. Bob, unaware of the rescaling, performs the same magnitude pruning procedure. Due to the rescaling, the magnitudes of the weights are altered, potentially leading Bob to prune weights that are actually important in the original network.

Figure 3.7 shows the results of such a scenario. For simplicity here I did not even attempt to choose a rescaling in an adversarial manner, I simply applied rescaling at random (see Appendix E.1 for details) before applying magnitude pruning. This still led to a significant drop in accuracy for MP, demonstrating its vulnerability to rescaling, while Φ-pruning remained unaffected due to its inherent invariance to such transformations.

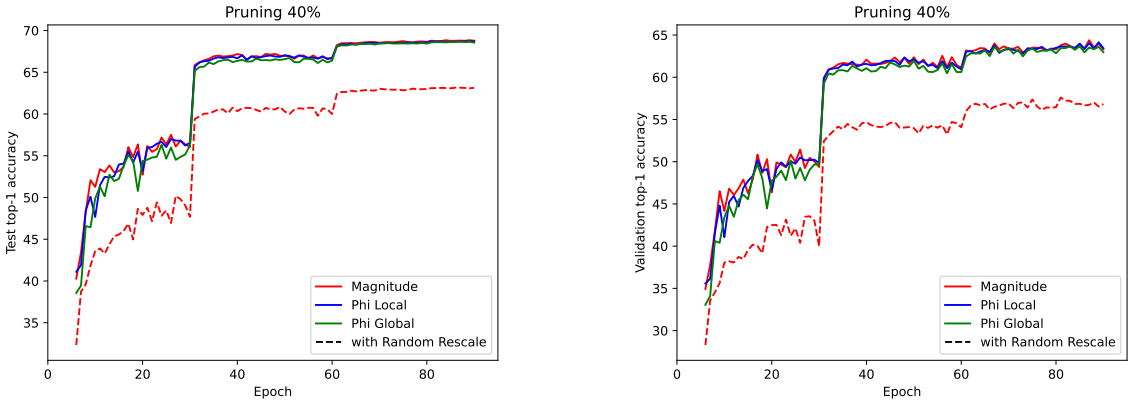


Figure 3.7: Training Curves: Test Top-1 Accuracy (left) and Validation Top-1 Accuracy (right) when finetuning the pruned models, with (dashed line) or without (plain line) rescaling. The results for Φ-pruning are the same with or without rescaling, hence the corresponding dashed line (random rescaling applied beforehand) perfectly overlaps with the plain line (no rescaling), unlike for MP.

Comparing criteria and masks. The left part of Figure 3.8 shows the magnitude (absolute value) versus the Φ-cost for each parameter index i, illustrating a positive correlation between these two quantities. This leads to a high overlap in the sets of pruned weights, as shown in Table 3.1. This suggests that the parameters obtained with SGD are naturally balanced in some sense, given that magnitude pruning, which is not invariant under rescaling, largely aligns with the rescaling-invariant pruning methods. The right part of Figure 3.8 shows that this correlation is largely reduced after applying a random rescaling as detailed in Appendix E.1. In this case, as we have seen on Figure 3.7, magnitude pruning indeed yields quite different results compared to Φ-pruning.

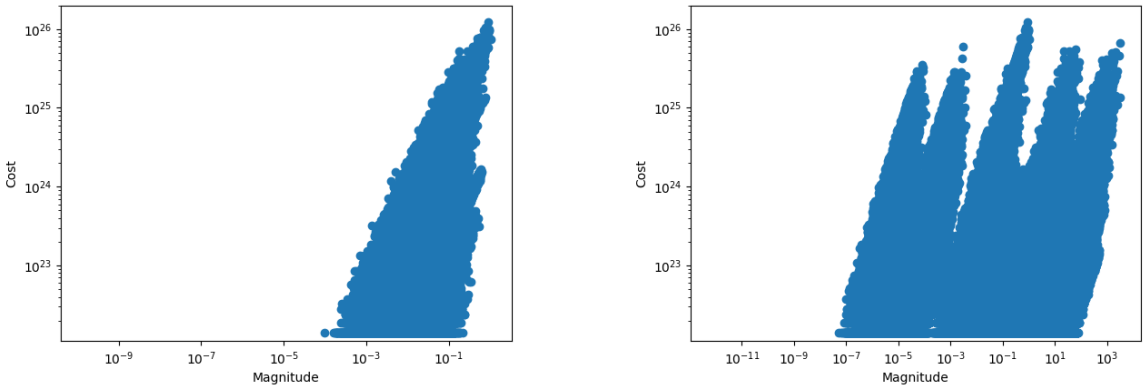


Figure 3.8: Scatter plot of the magnitude of each weight, versus its Φ-cost (Equation (3.16)). Left: dense model trained with SGD. Right: same but randomly rescaled as detailed in Appendix E.1.

Pruning level	10%	20%	40%	60%	80%
Overlap between MP and Φ-GP	70%	74%	76%	80%	86%
Overlap between MP and Φ-LP	87%	82%	90%	94%	96%

Table 3.1: Overlap between masks as a function of pruning level (percentage of pruned out coefficients). If S_1 and S_2 index the weights pruned (i.e. set to zero) by methods 1 and 2, the overlap is computed as $100 \times |S_1 \cap S_2|/|S_1|$. As all methods prune the same amount of weights we have $|S_1| = |S_2|$.

Comparing accuracies. I find that Φ-pruning methods achieve accuracies comparable to the standard magnitude pruning method. Specifically, both methods yield the same top-1 test accuracy at the end of training (Table 3.2), and their training curves are similar (Figure 3.7). This is noteworthy because, in this context, the choice of the pruning mask is crucial for achieving high accuracy, as demonstrated by the magnitude pruning method applied to a random rescaling of the parameters, which significantly underperforms (Table 3.2 and Figure 3.7).

Pruning level	none	10%	20%	40%	60%	80%
MP (+ *Random Rescale*)	67.7%	69.0 *(68.8)*	69.0 *(68.7)*	68.8 *(63.1)*	68.2 *(57.5)*	66.5 *(15.8)*
Φ-LP (∗)		68.8	68.9	68.7	68.1	66.1
Φ-GP (∗)		68.6	68.8	68.6	67.9	66.0

Table 3.2: Top-1 accuracy after pruning, rewind and retrain, as a function of the pruning level.
(∗) = results valid with as well as without rescaling, as Φ-pruning is invariant to rescaling. MP + Random Rescale corresponds to the case where I apply a random rescaling before applying MP (see Appendix E.1 for details).

Computation of the Φ-costs. Computationally speaking, all the Φ-costs can be computed easily all at once with the formula

$$(\Phi\text{-}\mathtt{Cost}(\theta, i))_{i\in G} = \theta \odot \nabla_\theta \|\Phi(\theta)\|_1$$

where $\odot$ is product-wise multiplication. Therefore, all the costs can be computed in one forward pass, to compute the path-norm $\|\Phi(\theta)\|_1$, and one backward pass, to compute the gradient $\nabla_\theta\|\Phi(\theta)\|_1$. The results of the computations show (cf the vertical axis of Figure 3.8) that the bound (3.19) is vacuous in this settings (on the order of 10^{26}). This will be further discussed in Chapter 7 (Section 7.2), along with leads to reduce this gap.

Overall, Φ-pruning is competitive with respect to the widespread magnitude pruning method in terms of accuracy, is inherently invariant to rescaling symmetries, and the emerging theory of Φ offers a promising foundation for future theoretical analysis of these pruning methods.

It is important to highlight that these results were achieved without any tuning effort: I used the same hyperparameters for the new Φ-pruning methods as those commonly employed for magnitude pruning in comparable situations [Frankle et al., 2021]. For further details, see Appendix E.1.

3.6 Conclusion

My main contribution in Chapter 3 has been to prove Lipschitz properties based on mixed path-norms, with practical applications to pruning. These properties can be numerically informative on the stability of the network output with respect to small perturbations of the input or the weights. The Lipschitz property in the input x is important for robustness matters, as it guarantees that for every parameters θ and every inputs x, x' (Theorem 3.3.1):

$$\|R_\theta(x) - R_\theta(x')\|_r \leqslant \|\Phi(\theta)\|_{1,r}\|x - x'\|_\infty.$$

This extends to the general class of DAG ReLU networks the same property that was known in the specific case of scalar-valued LFCNs without biases [Neyshabur et al., 2015].

The Lipschitz property in the weights θ, and based on Φ, is new and says that for every input x and every parameters θ, θ' with $\theta_i\theta'_i \geqslant 0$ for all i (Theorem 3.4.1):

$$\|R_\theta(x) - R_{\theta'}(x)\|_1 \leqslant \max(\|x\|_\infty, 1)\|\Phi(param) - \Phi(\theta')\|_1,$$

We have seen that these two Lipschitz properties have several desirable properties:

- they hold for general DAG ReLU networks, including widely used architectures such as ResNets and VGGs,
- they are invariant to rescaling and permutation symmetries,
- they are easy to compute,
- and they are at least as fine as other standard norms based on products of layers' norms.

Perspectives. There are many potential applications of these Lipschitz bounds. I already demonstrated a practical application of the Lipschitz bound in θ to design a pruning method that is not only competitive with traditional magnitude pruning but also invariant under rescaling symmetries, enhancing its robustness and efficiency. I will further show how this bound implies generalization guarantees in the next chapter (Chapter 4). Finally, and as already mentioned, the Lipschitz bound in θ can be used to get quantization error bounds, and I will give more details on that later, in Chapter 6 (Section 6.1.1).

Besides their theoretical interest, and practical interests for pruning, these Lipschitz bounds would be numerically informative only if the mixed-path-norms $\|\Phi(\theta)\|_{1,r}$ and the ℓ^1-path-metric $\|\Phi(\theta) - \Phi(\theta')\|_1$ are small enough in practical cases of interests.

While I already showed in the context of pruning (Section 3.5, and in particular Figure 3.8) that the ℓ^1-path-metric can be very large, I will further challenge the concrete promises of path-norm-based bounds in the specific context of generalization in the last chapter (Section 7.2). This will lead in Chapter 7 to a discussion on perspectives to improve the numerical tightness of these bounds.

Chapter 4

Generalization with path-norm

This chapter is based on results in Gonon et al. [2024a,b].

My car has learned to recognize stop signs on some images using a neural network R_θ. Will it recognize stop signs on *new* images?

Generalization is a fundamental aspect of machine learning, ensuring that models perform well on new, unseen data, not just the data they were trained on. In practice, generalization is a key concern when designing and training models, as the example of a car recognizing stop signs illustrates. In theory, generalization is a central question in statistical learning theory, aiming at understanding the factors influencing the generalization gap, the difference between the performance of a model on the training data and on the unknown underlying distribution of the data.

The path-norm has emerged as a promising complexity measure to bound the generalization gap of neural networks [Neyshabur et al., 2015, Kawaguchi et al., 2017, Barron and Klusowski, 2019, Jiang et al., 2020, Dziugaite et al., 2020]: it is one of the best weight-based measure in terms of *correlation with the generalization gap*, it is *easy to compute*, it is *tight for linear models*, it is *invariant to rescaling and permutation symmetries*, and it is a *more faithful* measure of complexity *than the standard product of layers' norms* in the case of simple LFCNs.

However, I already mentioned in Chapter 2 that the path-norm lacked versatility before this thesis, as it was only developed for simple LFCNs. Because of this, the reach of path-norm-based bounds has only be assessed on toy examples.

This chapter addresses this by providing the first generalization guarantees for neural networks based on the path-norm that are valid for general DAG networks. Our main bound is not only the most widely applicable path-norm-based bound but also either improves or recovers all the previous ones known in simpler settings.

The outline is as follows.

- Section 4.1 recalls the setup of supervised learning, the notion of expected risk, excess risk and generalization gap [Shalev-Shwartz and Ben-David, 2014, Goodfellow et al., 2016, Bach, 2024]. I then recall the notion of Rademacher complexity, a powerful and standard way to establish bounds on the excess risk and generalization gap (Theorem 4.1.1). *The reader already familiar with this can skip Section 4.1.*

- Section 4.2 motivates the use of the path-norm as a complexity measure to derive generalization guarantees for neural networks. *This section contains what motivated*

me in the first place to work on the path-norm. If I were to present my work chronologically, this section would come first.

- Section 4.3 proves the first bound on the Rademacher complexity (Rademacher bound, in short) using the path-norm that is valid for *general DAG networks* (Theorem 4.3.1)[Gonon et al., 2024b]. It is based on the classical Dudley's integral and the new Lipschitz property in θ provided by the ℓ^1-path-metric (Theorem 3.4.1 in Chapter 3). *Chronologically, this is the first generalization guarantee I proved using the path-norm in my thesis. I was happy, as this is the first path-norm-based bound that is valid for general DAG networks. However, I was also disappointed, as it is not as tight as the best-known bounds when applied to simpler settings such as LFCNs. This motivated me to look for bounds that are not only widely applicable but also tighter.*

- Section 4.4 improves on the previous Rademacher bound by using another proof technique based on a peeling argument (Theorem 4.4.1) [Gonon et al., 2024a]. *This time, this new bound either improves or recovers all the previous known ones based on the path-norm, while being more widely applicable to general DAG networks, rather than only LFCNs. This section is the main contribution of my thesis on path-norm-based generalization guarantees for neural networks.*

4.1 Supervised learning, generalization bounds, Rademacher complexity

This section recalls the classical and dominant approach to statistical learning based on the Rademacher complexity [Shalev-Shwartz and Ben-David, 2014, Goodfellow et al., 2016, Bach, 2024]. *The reader already familiar with this framework can directly go to Section 4.2.*

The main goal of machine learning is to find a function that minimizes the expected risk [Shalev-Shwartz and Ben-David, 2014, Goodfellow et al., 2016, Bach, 2024], as recalled in Section 4.1.1. However, this risk cannot be computed in practice. Instead, the classical approach to supervised learning consists in minimizing an empirical approximation of the expected risk, called the empirical risk (Section 4.1.2). The performance of the learned function is then evaluated by looking at the generalization gap and the estimation error, two statistical quantities recalled in this section, and that can be bounded by the standard Rademacher complexity (Section 4.1.3).

4.1.1 The goal is to minimize the expected risk

The goal of learning is to find network's parameters θ that minimize the so-called *expected risk*

$$R(\theta) := \mathbb{E}_{(x,y)\sim\mu}\ell(R_\theta(x), y),$$

where ℓ is a so-called *loss function* that measures the discrepancy between the true target y and the prediction $R_\theta(x)$ made by the network with parameters θ on input x. The expectation is taken over some *unknown* underlying distribution μ of the input-output pairs (x, y). This has nothing specific to neural networks so we might just as well consider

a set of functions $\mathcal{F}$, and define the *expected risk* for a general $f \in \mathcal{F}$. Thereafter, we will apply this discussion to neural networks by choosing $\mathcal{F} := \{R_\theta, \theta \in \Theta\}$, the set of functions R_θ realized by a neural network architecture with parameters $\theta \in \Theta$.

Definition 4.1.1 (Expected risk)**.** *The* expected risk *of a function $f : \mathcal{X} \to \mathcal{Y}$, with respect to a loss function $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ and a distribution μ on $\mathcal{X} \times \mathcal{Y}$, is defined as*

$$R(f) := R(f, \ell, \mu) := \text{expected risk}(f, \ell, \mu) := \mathbb{E}_{(x,y)\sim\mu}\ell(f(x), y). \tag{4.1}$$

For simplicity, I will from now on always drop the dependencies on ℓ and μ, e.g., writing $R(f)$ instead of $R(f, \ell, \mu)$, as they will always be clear from the context.

While there are different naming conventions in the literature, I will follow as much as possible the one of Bach [2024]. Note that the *expected risk* is also sometimes called the *generalization error* or the *testing error*.

Two standard situations are regression and classification.

- *Regression.* This corresponds to real-valued targets y ($\mathcal{Y} = \mathbb{R}^{d_{\text{out}}}$). For instance, x could be the configuration of a gas system and y the energy of the system and the function R_θ would be trained to predict the energy of a new configuration x. The loss is often the squared error:

$$\ell(y', y) = \|y - y'\|_2^2.$$

- *Classification.* This corresponds to discrete-valued targets y ($\mathcal{Y}$ finite). For instance, x could be an image and y the class/label of the object in the image ("traffic light", "stop sign", etc.). For a classification problem with C classes, the targets y are often encoded as one-hot vectors $y \in \{0, 1\}^C$ with a one at the index of the class and zeros elsewhere. Instead of choosing $\mathcal{Y} = \{0, 1\}^C$, it is often considered $\mathcal{Y} = \mathbb{R}^C$, to allow one to use the same models for regression and classification. However, for classification, the model's output $y' = R_\theta(x) \in \mathbb{R}^C$ is taken through a so-called softmax, an increasing function applied coordinate-wise to $y' \in \mathbb{R}^C$ to transform it in a probability distribution over the C classes, i.e., a vector $\text{softmax}(y') \in [0, 1]^C$ that sums to one:

$$\text{softmax}(y')_i := \frac{\exp(y_i)}{\sum_{j=1}^C \exp(y_j)}.$$

  The loss is often the cross-entropy loss, pre-composed by the softmax: this measures the discrepancy between the true distribution y and the predicted distribution $\text{softmax}(y')$:

$$\ell(y', y) = -\sum_{i=1}^C y_i \log(\text{softmax}(y')_i).$$

  Another common loss is the zero-one loss, or top-1 accuracy loss, that measures the number of misclassifications:

$$\ell(y', y) = \mathbb{1}_{\arg\max(y) \neq \arg\max(y')}.$$

  It is equal to zero if the prediction made by y' is the true class, i.e., if the most probable class $\arg\max(y')$ according to y' corresponds to the class encoded by the one-hot vector y, and one otherwise.

Machine learning is concerned with finding a minimizer f^* of the expected risk over *all measurable functions*[Shalev-Shwartz and Ben-David, 2014, Goodfellow et al., 2016, Bach, 2024]:

$$R(f^*) = \inf_{f \text{ measurable}} R(f).$$

Definition 4.1.2 (Bayes predictor)**.** *Consider sets $\mathcal{X}$ and $\mathcal{Y}$, a loss function $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$, and a distribution μ on $\mathcal{X} \times \mathcal{Y}$. A* Bayes predictor *is a measurable function $f^* : \mathcal{X} \to \mathcal{Y}$ that minimizes the expected risk:*

$$R(f^*) = \inf_{f \text{ measurable}} R(f).$$

It can be shown that a Bayes predictors exists, is unique up to a set of measure zero, and is defined by [Bach, 2024, Proposition 2.1]:

$$f^*(x) \in \arg\min_{y \in \mathcal{Y}} \mathbb{E}_z(\ell(y, z)|x),$$

where the joint distribution $(x, y) \sim \mu$ is *conditioned* to the given input value x.

- For regression with squared loss $\ell(y', y) = \|y - y'\|_2^2$, a Bayes predictor is the expected value y given input value x
$$f^*(x) = \mathbb{E}(y|x).$$
- For classification with top-1 accuracy loss $\ell(y', y) = \mathbb{1}_{\arg\max(y) \neq \arg\max(y')}$, a Bayes predictor f^* chooses the most probable label associated with an input x
$$f^*(x) = \arg\max_y \mathbb{P}(y|x).$$

In practice, Bayes predictors are not accessible since the underlying distribution μ is unknown, and we can only aim at approximating one of them. A quantity of interest is the *excess risk* of a given function f compared to the Bayes predictors.

Definition 4.1.3 (Excess risk)**.** *Consider sets $\mathcal{X}$ and $\mathcal{Y}$, a loss function $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$, and a distribution μ on $\mathcal{X} \times \mathcal{Y}$. The* excess risk *of a measurable function $f : \mathcal{X} \to \mathcal{Y}$ is defined as the difference between the expected risk of f (Definition 4.1.1) and the expected risk of a Bayes predictor $f^* : \mathcal{X} \to \mathcal{Y}$ (Definition 4.1.2):*

$$\text{excess risk}(f) := R(f) - R(f^*). \tag{4.2}$$

In general, only a subset $\mathcal{F}$ of all measurable functions can be reached by an algorithm that looks for an approximation f of a Bayes predictor. This leads to the definition of the *approximation error* and the *estimation error*.

Definition 4.1.4 (Approximation error, estimation error)**.** *Consider a set $\mathcal{F}$ of functions $\mathcal{X} \to \mathcal{Y}$, a loss $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$, and a distribution μ on $\mathcal{X} \times \mathcal{Y}$. The* approximation error *is a quantitative measure of how far are the Bayes predictors from the best function in $\mathcal{F}$:*

$$\text{approximation error}(\mathcal{F}) := \inf_{f \in \mathcal{F}} R(f) - R(f^*). \tag{4.3}$$

The estimation error *is a quantitative measure of how much worse is a given f compared to the best one in $\mathcal{F}$:*

$$\text{estimation error}(f, \mathcal{F}) := R(f) - \inf_{g \in \mathcal{F}} R(g). \tag{4.4}$$

.

With these notations, we can now decompose the *excess risk* as the sum of the *estimation error* and the *approximation error*:

$$\text{excess risk}(f) = \text{estimation error}(f, \mathcal{F}) + \text{approximation error}(mathcalF). \tag{4.5}$$

In this decomposition, the *approximation error* is of a fundamentally different nature than the *estimation error*: it depends only on the set $\mathcal{F}$ to which we are restricted, and not on the function f. The *estimation error* is a *statistical quantity* as soon as f is *learned from samples* of the distribution μ, as I now explain in Section 4.1.2.

4.1.2 In practice: empirical risk minimization (ERM)

Ideally, we would like to choose $f^*_{\mathcal{F}} \in \mathcal{F}$ as

$$f^*_{\mathcal{F}} \in \arg\min_{f \in \mathcal{F}} R(f).$$

However $R(f) = \mathbb{E}_{(x,y)\sim\mu}\ell(f(x), y)$ cannot be computed as the underlying distribution μ is unknown in practice. In many cases of interest, we have access to samples $S = (x_i, y_i)_{i=1,\dots,n}$ *independent and identically distributed (iid)* according to μ. This corresponds to *supervised learning*, where we not only have access to inputs x_i but also the desired outputs y_i [Shalev-Shwartz and Ben-David, 2014, Goodfellow et al., 2016, Bach, 2024]. The classical approach to supervised learning is to approximate $R(f)$ by the *empirical risk*, also called the *training error*:

$$R_S(f) := \text{training error}(f, S) := \frac{1}{n}\sum_{i=1}^{n} \ell(f(x_i), y_i)$$

and to learn a function $\hat{f}(S) \in \mathcal{F}$ by aiming at minimizing the training error (which depends on S):

$$\inf_{f\in\mathcal{F}} R_S(f). \tag{4.6}$$

Then, the function $\hat{f}(S) : \mathcal{X} \to \mathcal{Y}$ is used to predict the target $y \simeq \hat{f}(S)(x)$ for new inputs x. The ERM algorithm defines what is called an *estimator* $\hat{f}$ in statistics, that we will call the *ERM estimator* in the following.

Definition 4.1.5 (Estimator and its expected risk)**.** *Consider a set $\mathcal{F}$ of functions $\mathcal{X} \to \mathcal{Y}$ and a distribution μ on $\mathcal{X} \times \mathcal{Y}$. An* estimator *learned from $n \in \mathbb{N}_{>0}$ samples is defined as any measurable function $\hat{f} : S \in (\mathcal{X} \times \mathcal{Y})^n \mapsto \hat{f}(S) \in \mathcal{F}$, and we will simply write $\hat{f} \in \mathcal{F}$.*

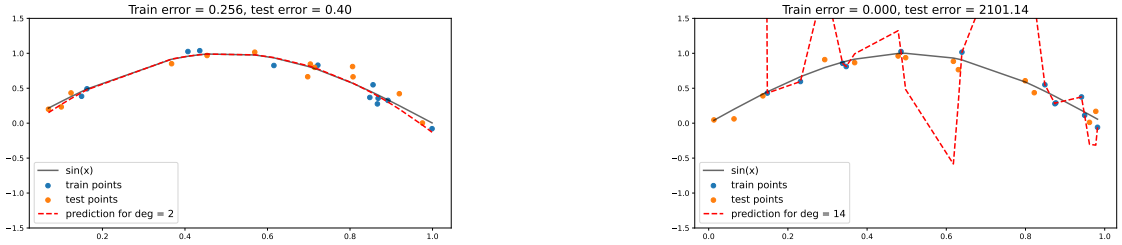
Since an estimator $\hat{f}$ is is a measurable function in the training samples S, there is for each S an associated expected risk $R(\hat{f}(S)) = \mathbb{E}_{(x,y)\sim\mu}\ell(\hat{f}(S)(x), y)$. Here, we need to be a little careful since we now have two sources of randomness: S and (x, y). The correct way to interpret the expectation $\mathbb{E}_{(x,y)\sim\mu}$ is that it averages only over (x, y), while S is considered to be fixed. While this informal definition is enough to understand what follows, let me mention for readers familiar with probability theory that it is formally defined as a *conditional expectation* with respect to S, where $(x, y) \sim \mu$ is independent of S:

$$R(\hat{f}(S)) := \mathbb{E}_{(x,y)\sim\mu}\left(\ell(\hat{f}(S)(x), y)|S\right). \tag{4.7}$$

An example of empirical risk minimization is given in Figure 4.1 for regression with $\mathcal{Y} = \mathbb{R}$, the squared loss $\ell(y', y) = (y' - y)^2$, an underlying distribution μ on (x, y) such that x is uniform on $[0, 1]$ and $y = \sin(\pi x) + \mathcal{N}(0, 0.01)$. The ERM estimator is searched into the set $\mathcal{F}_d$ of polynomial functions of degree at most d. For the *same* $n = 15$ iid samples $S = (x_i, y_i)_{i=1}^n$ drawn according to μ, we see that the ERM estimator $\hat{f}(S) \in \mathcal{F}_d$ has very different generalization abilities depending on the degree d considered.

In the case of a polynomial of degree $d = 2$, the ERM estimator generalizes well to new data: it is a pretty good approximation of the sinus function. In this case, the training error is close to the expected risk.

In the case of a polynomial of degree $d = 14$, the ERM estimator does not generalize well to new data: the degree is large enough to allow for an exact fit of the training samples. But in order to do so, the ERM estimator oscillates a lot between the training samples, which leads to a poor approximation of the sinus in-between the training samples. In this case, the training error is zero, while the expected risk is very large. It is said that the model *overfit* the training data [Shalev-Shwartz and Ben-David, 2014, Goodfellow et al., 2016, Bach, 2024].



(a) Polynomial regression of degree $d = 2$.

(b) Polynomial regression of degree $d = 14$.

Figure 4.1: For these experiments, I have fixed $n = 15$ training samples (x_i, y_i) drawn at random as described in the text, and considered squared-loss regression which consists in minimizing $\sum_{i=1}^n (f_\theta(x_i) - y_i)^2$. I minimized this over different sets of polynomial functions $f_\theta(x) = \sum_{i=0}^d \theta_i x^i$, corresponding to different degrees d. The left plot shows the result for $d = 2$, which generalizes well to new data, while the right plot shows the result for $d = 14$, which does not generalize well to new data.

In general, to gain deeper insights into the generalization of the ERM estimator $\hat{f}$, it is standard to *compare* its expected risk $R(\hat{f})$ (remember that this is a random variable depending on the training samples S, see the discussion above Equation (4.7)) to two other quantities.

- **Generalization gap: comparison to the training error of *the same function* $\hat{f}$.** Since $\hat{f}(S)$ minimizes the training error on S, a first natural comparison of the expected risk $R(\hat{f}(S))$ is with the training error. Let me define it for a general function $f : \mathcal{X} \to \mathcal{Y}$ and a set of samples S as

  $$\text{generalization gap}(f, S) := \text{expected risk}(f) - \text{training error}(f, S) = R(f) - R_S(f).$$

  It is small if and only if the function f performs similarly on the training set S and the underlying distribution μ, as in Figure 4.1a. When this is the case for $f := \hat{f}(S)$,

this means that using the training error on S as a proxy for the expected risk was *a posteriori* a good strategy to achieve a small expected risk. A large generalization gap is symptomatic of overfitting as in Figure 4.1b.

- **Estimation error: comparison to *the best function of the model*.**

  I recall here for convenience the definition of the estimation error for a general function $f : \mathcal{X} \to \mathcal{Y}$ (Definition 4.1.4):

  $$\text{estimation error}(f) = \text{expected risk}(f) - \inf_{g \in \mathcal{F}} \text{expected risk}(g).$$

  If it is small for $f = \hat{f}(S)$, the ERM estimator for learn on samples S, then this means that minimizing the training error on S was *a posteriori* a good strategy to find a (near) minimizer of the expected risk in $\mathcal{F}$.

Note that the generalization gap can be very well approximated in practice. Indeed, the expected risk of $\hat{f}(S)$ can be approximated by taking an average over m *new independent* samples $(x_i^{\text{test}}, y_i^{\text{test}})$ independent from the training set S,

$$\text{empirical expected risk}(\hat{f}(S)) := \frac{1}{m} \sum_{i=1}^{m} \ell(\hat{f}(S)(x_i^{\text{test}}), y_i^{\text{test}}).$$

This is a good approximation as soon as the empirical expected risk is highly concentrated around its expectation[1], which happens for large m [Boucheron et al., 2013, Kawaguchi et al., 2017]. While this is very useful *a posteriori* to assess the performance of a network in practice, it does not provide an *explanation* of *why* the network generalizes well or not, hence no *guidance a priori* to design better models, training algorithms etc. A recurrent goal in the literature [Shalev-Shwartz and Ben-David, 2014, Goodfellow et al., 2016, Bach, 2024] is to aim at better understanding the *factors influencing* the generalization gap and the estimation error, rather than just approximating them.

In the case of polynomial regression, we could guess by running experiments as in Figure 4.1 that good generalization arises for ERM if the ratio d/n is small, meaning that the degree of the polynomial is smaller than the number of training samples, and overfit arises if the ratio d/n is large. Therefore, the degree of the polynomial can serve as an explanation of the generalization ability of the ERM estimator. In theory, it is indeed possible to establish theoretical guarantees in this sense: the estimation error and the generalization gap are bounded by a function of d/n. In the case of *neural networks*, finding a good complexity measure that can be used to explain the generalization of ERM is an active area of research. In this thesis, I will investigate the promises and shortcomings of the path-norm as a complexity measure to bound the generalization gap and the estimation error of neural networks. Let me first recall in the next section the standard way to bound the generalization gap with Rademacher complexities.

4.1.3 The Rademacher complexity bounds the performance of ERM

The dominant statistical approach to bound either the generalization gap or the estimation error of a function $\hat{f}(S) \in \mathcal{F}$ learned from n training samples $S = (x_i, y_i)_{i=1,\dots,n}$ is via

[1] While this is not important, note that I should have said "expectation *conditioned to* S" to be precise, since the expected risk $R(\hat{f}(S))$ is an expectation conditioned to S (Equation (4.7)).

the so-called empirical Rademacher complexity of $\mathcal{F}$ on S [Bartlett and Mendelson, 2002, Shalev-Shwartz and Ben-David, 2014, Bach, 2024]. It is defined as

$$\mathcal{R}(\mathcal{F}, S) := \mathbb{E}_\sigma \left[\sup_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^{n} \langle \varepsilon_i, f(x_i) \rangle \right], \tag{4.8}$$

where $\varepsilon_i = (\varepsilon_{i,j})_j \in \mathbb{R}^{d_{\text{out}}}$ and the $\varepsilon_{i,j}$'s are iid Rademacher variables ($\mathbb{P}(\varepsilon_{i,j} = 1) = \mathbb{P}(\varepsilon_{i,j} = -1) = 1/2$). The empirical Rademacher complexity measures the ability of the functions in $\mathcal{F}$ to positively correlate with random signs. *Note that the Rademacher complexity only depends on the input samples x_i and not on the target samples y_i. In the future, we will simply consider $S = (x_i)_{i=1,\dots,n}$ with each x_i drawn iid according to a distribution μ_x (corresponding to the first marginal of the joint distribution μ on (x, y)).*

Definition 4.1.6 (Rademacher complexity)**.** *Consider a set $\mathcal{F}$ of functions $\mathcal{X} \to \mathbb{R}^{d_{out}}$ and a distribution μ_x on $\mathcal{X}$. The* Rademacher complexity *of $\mathcal{F}$ for the distribution μ_x is defined as [Shalev-Shwartz and Ben-David, 2014, Bach, 2024]:*

$$\mathcal{R}(\mathcal{F}, \mu_x) := \mathbb{E}_{S \sim (\mu_x)^{\otimes n}} \mathcal{R}(\mathcal{F}, S), \tag{4.9}$$

where each sample x_i of $S = (x_i)_{i=1}^n$ is drawn iid according to μ_x, and where the empirical Rademacher complexity $\mathcal{R}(\mathcal{F}, S)$ *has been defined in Equation (4.8).*

Theorem 4.1.1 (Generalization bound via Rademacher complexity [Shalev-Shwartz and Ben-David, 2014, Bach, 2024])**.** *Consider $\mathcal{F}$ a class of functions from an input space $\mathcal{X}$ to $\mathbb{R}^{d_{out}}$, a distribution μ on $\mathcal{X} \times \mathbb{R}^{d_{out}}$, and a loss function $\ell : \mathbb{R}^{d_{out}} \times \mathbb{R}^{d_{out}} \to \mathbb{R}$ such that $\ell(\cdot, y)$ is L-Lipschitz for every $y \in \mathbb{R}^{d_{out}}$. Denote by μ_x the marginal of μ on $\mathcal{X}$. For any estimator $\hat{f} \in \mathcal{F}$ (Definition 4.1.5), it holds:*

$$\begin{aligned} \mathbb{E}_S \text{generalization gap}(\hat{f}(S), S) &\leqslant \frac{2\sqrt{2}L}{n} \mathcal{R}(\mathcal{F}, \mu_x) \\ \mathbb{E}_S \text{estimation error}(\hat{f}(S), S) &\leqslant \frac{4\sqrt{2}L}{n} \mathcal{R}(\mathcal{F}, \mu_x). \end{aligned} \tag{4.10}$$

where the randomness is over n training samples $S = (x_i, y_i)_{i=1,\dots,n}$ drawn iid according to μ.

The bounds (4.10) hold in expectation. Classical concentration results [Boucheron et al., 2013] can be used to deduce a bound that holds with high probability on the choice of S, under additional mild assumptions on the loss, see, e.g.,Theorem 26.5 in Shalev-Shwartz and Ben-David [2014].

In practice, when we observe the output $\hat{f}(S)$ of ERM with samples S, the choice of $\mathcal{F}$ that guarantees $\hat{f}(S) \in \mathcal{F}$ *for all* S is not clear. I will come back on this subtlety in Chapter 7 (see Section 7.3.1).

Proof of Theorem 4.1.1. The proof is standard [Shalev-Shwartz and Ben-David, 2014, Theorem 26.3][Bach, 2024, Equation (13.3)] and uses two arguments: the symmetrization property given by Shalev-Shwartz and Ben-David [2014, Theorem 26.3], and the vector-valued contraction property given by Maurer [2016]. These are the relevant versions of very classical arguments that are widely used to reduce the problem to the

Rademacher complexity of the model [Bach, 2024, Propositions 4.2 and 4.3][Wainwright, 2019, Equations (4.17) and (4.18)][Bartlett and Mendelson, 2002, Proof of Theorem 8][Shalev-Shwartz and Ben-David, 2014, Theorem 26.3][Ledoux and Talagrand, 1991, Equation (4.20)]. □

Example 4.1.1 (Cross-entropy loss)**.** *The cross-entropy loss is the standard loss chosen to train neural networks for classification problems. Recall that it is defined as*

$$\ell(y', y) = -\sum_{i=1}^{C} y_i \log(\text{softmax}(y')_i), \tag{4.11}$$

where

$$\text{softmax}(y')_i = \frac{\exp(y'_i)}{\sum_{j=1}^{C} \exp(y'_j)}$$

is the softmax function that maps the output of the network $y' \in \mathbb{R}^C$ to a probability distribution over the C classes, and where y is a one-hot encoding of a class c, meaning that $y = (\mathbb{1}_{c'=c})_{c' \in \{1,\dots,d_{out}\}}$.

The cross-entropy loss does satisfy *the assumption that $\ell(\cdot, y)$ is L-Lipschitz with respect to the ℓ^2-norm with $L = \sqrt{2}$, for every possible one-hot encoding vector y of a class c [Bartlett et al., 2017, Lemma 2.1]. Therefore, Theorem 4.1.1 applies to the cross-entropy loss with $L = \sqrt{2}$. Indeed, the softmax is 1-Lipschitz, and the function $z \mapsto -\sum_{i=1}^{C} y_i \log(z_i)$ is $\sqrt{2}$-Lipschitz. I provide the details for the sake of completeness in Appendix C.6.*

Example 4.1.2 (Top-1 Accuracy Loss)**.** *The top-1 accuracy loss is the standard loss used to evaluate the performance of a function in classification problems. Recall that it is defined as*

$$\ell(y', y) = \mathbb{1}_{\arg\max(y) \neq \arg\max(y')} \tag{4.12}$$

where $y \in \mathbb{R}^C$ is a one-hot encoding of a class $c \in \{1, \dots, C\}$, meaning that $y = (\mathbb{1}_{c'=c})_{c' \in \{1,\dots,d_{out}\}}$.

The function $\ell(\cdot, y)$ is not *L-Lipschitz for any finite $L > 0$ in general. Indeed, the top-1 accuracy loss is not continuous. Therefore, Theorem 4.1.1 does* not directly apply *to the top-1 accuracy loss. It is still possible to bound it with a Rademacher complexity but associated to another loss: the* margin loss*, a relaxed definition of the top-1 accuracy loss [Bartlett et al., 2017, Lemma A.4].*

I invite the readers to skip the remaining details on the margin loss, to go directly to Section 4.1.4, and refer back later if needed.

For $y' \in \mathbb{R}^{d_{out}}$ and a one-hot encoding $y \in \mathbb{R}^{d_{out}}$ of the class c, the margin $M(y', y)$ is defined by

$$M(y', y) := [y']_c - \max_{c' \neq c} [y']_{c'}.$$

For $\gamma > 0$, the γ-margin-loss is defined by [Bartlett et al., 2017]

$$\ell_\gamma(y', y) = \begin{cases} 0 & \text{if } \gamma < M(y', y), \\ 1 - \frac{M(y', y)}{\gamma} & \text{if } 0 \leqslant M(y', y) \leqslant \gamma, \\ 1 & \text{if } M(y', y) < 0. \end{cases} \tag{4.13}$$

For any class c and one-hot encoding y of c, it is known that $y' \in \mathbb{R}^{d_{out}} \mapsto M(y', y)$ is 2-Lipschitz with respect to the L^2-norm on y' [Bartlett et al., 2017, Lemma A.3]. Moreover, the function

$$r \in \mathbb{R} \mapsto \begin{cases} 0 & \text{if } r < -\gamma, \\ 1 + \frac{r}{\gamma} & \text{if } -\gamma \leqslant r \leqslant 0, \\ 1 & \text{if } r > 0. \end{cases}$$

is $\frac{1}{\gamma}$-Lipschitz. By composition, this shows that $y' \in \mathbb{R}^{d_{out}} \mapsto \ell_\gamma(y', y)$ is $\frac{2}{\gamma}$-Lipschitz with respect to the L^2-norm so the generic Rademacher bound applies to the γ-margin-loss. Moreover, the γ-margin-loss is a relaxation of the top-1 accuracy loss: if $\ell_\gamma(y', y) \geqslant \ell(y', y)$ for every y' and one-hot encoding y. This leads to the following bound for the expected top-1 accuracy risk.

Theorem 4.1.2 (Bound on the probability of misclassification [Bartlett et al., 2017])**.** *Let $\mathcal{F}$ be a class of functions from an input space $\mathcal{X}$ to $\mathbb{R}^{d_{out}}$. Consider μ a distribution on $\mathcal{X} \times \mathbb{R}^{d_{out}}$, and $S = (x_i)_{i=1,\dots,n}$ be n training samples drawn iid according to the marginal of μ on $\mathcal{X}$. We have for all estimator $\hat{f} \in \mathcal{F}$ and every $\gamma > 0$:*

$$\text{expected risk}(\hat{f}, \ell, \mu) \leqslant \text{expected risk}(\hat{f}, \ell_\gamma, \mu)$$

In particular, using classical concentration results [Boucheron et al., 2013] and the bound in expectation of Theorem 4.1.1 applied to ℓ_γ, which satisfies the Lipschitz assumption for $L = 2/\gamma$, with probability at least $1 - \delta > 0$ over the choice of $S \sim \mu^{\otimes n}$ [Bartlett et al., 2017, Lemma 3.1]:

$$\text{expected risk}(\hat{f}(S), \ell, \mu) \leqslant \text{training error}(\hat{f}(S), \ell_\gamma, S) + \frac{4\sqrt{2}}{n\gamma}\mathcal{R}(\mathcal{F}, \mu_x) + 3\sqrt{\frac{\ln(2/\delta)}{2n}}. \tag{4.14}$$

Note that the result is homogeneous: scaling both $\hat{f}(S)$ and γ by the same scalar leaves the losses and the Rademacher complexity *unchanged.*

Remark 4.1.1 (Case where f is a minimizer of the training error)**.** *In the special case where for every S, the function f is chosen to be exactly a minimizer of the training error on S, we even have [Shalev-Shwartz and Ben-David, 2014, Theorem 26.3]:*

$$\mathbb{E}_S \text{estimation error}(f, S) \leqslant \mathbb{E}_S \text{generalization gap}(f, S) \leqslant \frac{2\sqrt{2}L}{n}\mathcal{R}(\mathcal{F}, \mu_x).$$

because for any $f^* \in \mathcal{F}$,

$$\text{expected risk}(f^*) = \mathbb{E}_S \text{training error}(f^*, S) \geqslant \mathbb{E}_S \text{training error}(f, S)$$

and we can let f^* approximate the model's oracle error to deduce the bound.

In order to bound the generalization gap or the estimation error, it is therefore enough to bound the Rademacher complexity, as shown by Theorem 4.1.1 and Theorem 4.1.2. Bounding the Rademacher complexity gets specific to $\mathcal{F}$. Before delving into the case of neural networks, let me mention practical cases where $\mathcal{F}$ and n are known to correspond to a Rademacher complexity that is too large to be informative [Zhang et al., 2021].

4.1.4 Cases where the Rademacher complexity is too large

The Rademacher complexity is a measure of the richness of the set $\mathcal{F}$ of functions compared to a given number of samples n. By definition (Equation (4.9)), it is large if, in expectation over the choice of n samples $S = (x_i)_{i=1,\dots,n}$, the set $\mathcal{F}$ is so rich that for any choice of labels $\varepsilon_i \in \{-1, 1\}^{d_{\text{out}}}$, we can find $f \in \mathcal{F}$ that adapts to this arrangement of labels: $\langle f(x_i), \varepsilon_i \rangle \geqslant 0$.

If the set $\mathcal{F}$ is fixed, one can expect the existence of a threshold $N \in \mathbb{N}_{>0}$ such that the bounds are vacuous for $n \leqslant N$, $\mathcal{F}$ being too expressive for small n's, and informative for $n \geqslant N$. In practice, the training datasets are standardized, hence n is fixed. The architectures are also standards, so the DAG network G is also fixed. Therefore, to have a Rademacher complexity small enough to be informative, $\mathcal{F}$ should be taken as *a subset of the functions realized by the network G that is not too expressive compared to the number of samples n*.

If we just take $\mathcal{F}$ as the set of *all* functions realized by the network G, this set is *known to be too expressive for classical choices of n and architectures G* to allow for informative bounds [Zhang et al., 2021]. Indeed, with the same set of n inputs $S = (x_i)_{i=1}^n$ corresponding to images divided in C classes, Zhang et al. [2021] empirically show that the network G can be trained to fit any set of random labels $y_i \in \{1, \dots, C\}$, leading to a vacuous bound if $\mathcal{F}$ contains all these functions.

To understand why, let me reproduce a discussion from Zhang et al. [2021] with our notations. Denote by $\text{generalization gap}(p)$ the generalization gap when trained on the original data distribution modified to have a proportion $p \in [0, 1]$ of random labels chosen uniformly in the C different classes. The generalization bound $B(p) := \frac{2\sqrt{2}L}{n}\mathcal{R}(\mathcal{F}, \mu_x)$ (Equation (4.10)) is the same for all the level of random labels since the Rademacher complexity only depends on the input distribution. Therefore, we have $B(p) = B$ for all p, and we get

$$\sup_p \text{generalization gap}(p) \leqslant B,$$

where by definition,

$$\begin{aligned}\text{generalization gap}(p) = {} & \text{expected risk when trained on } p\% \text{ of random labels} \\ & - \text{training error when trained on } p\% \text{ of random labels.}\end{aligned}$$

It is empirically shown in Zhang et al. [2021] that the network G can be trained to zero training error for all p. Therefore, $\text{generalization gap}(p)$ is just equal to the expected risk. Zhang et al. [2021] uses this to argue that the Rademacher bound must be vacuous at least in the case of the top-1 accuracy loss $\ell(y', y) = \mathbb{1}_{\arg\max(y) = \arg\max(y')}$. Although this argument does *not* formally apply to the top-1 accuracy loss, simply because the bound

$$\sup_{p \in [0,1]} \text{generalization gap}(p) \leqslant B$$

does not hold (the left-hand side has essentially to be corrected by a factor[2]), I still

[2] A well-known result [Bartlett et al., 2017, Lemma 3.1], that I have reproduced earlier in Theorem 4.1.2, indicates that the generalization gap has essentially to be multiplied by the data margin for the bound to hold. However, this margin is typically not large enough to compensate to make the right-hand side informative in practice, so the conclusion of the argument of Zhang et al. [2021] still holds.

reproduce it below as this argument gives a good qualitative idea of what is going on. Moreover, the same type of argument is expected to *formally* apply to other losses.

First, observe that the expected top-1 accuracy risk must be at least $p(1-1/C)$ since there is a probability p of having an input with a random label, and since the label is chosen uniformly, there is only $1/C$ chances of being right in this case. And since the generalization gap is equal to the expected risk (the training error is zero), we would have if the Rademacher bound was holding in this case

$$\sup_{p\in[0,1]} p(1-1/C) = 1-1/C \leqslant B.$$

For the top-1 accuracy loss, the bound B is informative only if $B \leqslant 1$, vacuous otherwise. Therefore, Zhang et al. [2021] concludes that this bound is vacuous as soon as $\mathcal{F}$ contains at least one function realized by G that fits random labels for p large enough (just $C \geqslant 2$ already implies $B \geqslant 1/2$).

This motivates to look for a *smaller* set $\mathcal{F}$ of functions realized by the network G that is not too expressive compared to the number of samples n. A widespread idea is to look for a complexity measure that would be large for the functions that fit random labels, and small for the functions that do not, so we could define $\mathcal{F}$ as the set of functions that have a small complexity measure. In this thesis, I have been interested in the path-norm as a complexity measure, for reasons that I now explain.

4.2 Path-norm as a complexity measure

If I were to tell the story of my thesis chronologically, I would have placed this section first. It was precisely by discovering the concepts in the order presented in this section that I became interested in the path-norm initially and felt compelled to extend it to more general networks.

The goal of many studies in generalization is to find a complexity measure that explains the generalization ability of the ERM estimator. A typical bound on the Rademacher complexity (Rademacher bound for short) looks like

$$\mathcal{R}(\mathcal{F}, \mu_x) \leqslant \frac{\text{complexity measure}(\mathcal{F}, \mu_x)}{\sqrt{n}}.$$

In this thesis, I take interest in the ℓ^1-path-norm as a complexity measure for generalization. The reasons for this are multiple.

- *Path-norm measures the size of the weights, not the size of the network architecture.* For polynomial regression, Figure 4.1 shows that the degree d seems to be a good complexity measure. It is indeed possible to establish theoretical guarantees in this sense: the estimation error and the generalization gap are bounded by $\sqrt{d/n}$ times something that depends on how far μ is from a polynomial model [Bach, 2024]. For DAG neural networks (Definition 2.2.2), the equivalent would be to measure the size of the architecture (graph size). However, it has been known for a long time that *the size of the weights is more important than the size of the network* [Bartlett, 1996, Zhang et al., 2021, Bach, 2024]. Indeed, current neural network

architectures are so large compared to the current training datasets that they can essentially fit any function on these training sets [Zhang et al., 2021]. Among these functions, some generalize well, others poorly. Therefore, the same architecture can lead to very different generalization abilities and its size is not relevant to explain generalization. *This motivates the use of a complexity measure that accounts for the size of the weights* [Zhang et al., 2021].

- *Path-norm correlates empirically better with the generalization gap than other weight-based complexity measures.* Jiang et al. [2020] and Dziugaite et al. [2020] made large scale experiments to compare different weight-based complexity measures for generalization. They found that the path-norm is one of the measure that correlates the more positively with the generalization gap. It even does so almost as well as an oracle does, where the oracle has access to the true generalization gap plus a small noise. *This makes the path-norm a promising candidate for establishing theoretical guarantees on the generalization of neural networks.*

- *Path-norm is invariant to symmetries (Theorem 2.4.1).* I already evoked in Section 1.1.2 that some bounds can be vacuous if they are not invariant to symmetries of the network. This also applies to Rademacher bounds. For instance, consider the bound

$$\mathcal{R}(\mathcal{F}_\Theta, \mu_x) \leqslant \frac{\text{complexity measure}(\mathcal{F}_\Theta, \mu_x)}{\sqrt{n}}$$

with $\mathcal{F}_\Theta := \{R_\theta, \theta \in \Theta \subset \mathbb{R}^G\}$. The left-hand side is invariant to symmetries, since for any rescaling vector $\lambda \in \mathbb{R}^H_{>0}$ (Definition 2.4.1), we have $\mathcal{F}_\Theta = \mathcal{F}_{\lambda \diamond \Theta}$, and the Rademacher complexity of $\mathcal{F}_\Theta$ is the same as the Rademacher complexity of $\mathcal{F}_{\lambda\diamond\Theta}$ since they contain the same functions ($R_\theta = R_{\lambda\diamond\theta}$). However, the amplitude of the right-hand side depends on the complexity measure. A weight-based complexity measure is typically of the form

$$\text{complexity measure}(\mathcal{F}_\Theta, \mu) = \sup_{\theta\in\Theta} \psi(\theta)$$

for some real-valued function ψ. If $\psi(\lambda \diamond \theta) \to \infty$ as $\|\lambda\| \to \infty$, the bound can be vacuous depending on the scaling of the weights in Θ. If $\psi(\theta)$ is the ℓ^1-path-norm $\|\Phi(\theta)\|_1$, this problem *does not occur* since it is invariant to symmetries (Theorem 2.4.1).

- *Path-norm is the infimum over weakly rescaling-equivalent parameters of the product of the layers' norms.* A common weight-based complexity measure *for LFCNs* with matrices $\theta = (M_1, \ldots, M_L)$ is the product of the layers' norms $\psi(\theta) := \prod_{l=1}^L \|M_\ell\|_{1,\infty}$ [Bartlett et al., 2017, Neyshabur et al., 2018]. We already saw in Theorem 3.3.2 that the path-norm is a finer measure as it is the infimum over all weakly rescaling-equivalent parameters of the product of the layers' norms.

- *Path-norm is easy to compute (Theorem 3.1.1).* This is in contrast to other complexity measures that require more complex computations, such as the spectral norm for convolutional layers [Sedghi et al., 2019, Araujo et al., 2021, Delattre et al., 2024].

Because of all these reasons, the path-norm is a promising candidate to establish theoretical guarantees on the generalization of neural networks. It turns out that the path-norm has already been used for this purpose in the literature, but only for simple LFCNs [Neyshabur et al., 2015, Barron and Klusowski, 2019]. Thus, the promises of theoretical guarantees based on the path-norm were currently out of reach before this thesis: they could not even be tested on standard practical networks such as ResNets. This prevented us from both understanding the reach of path-norm-based generalization guarantees and from diagnosing their strengths and weaknesses, which is necessary to either improve them in order to make them actually operational, if possible, or to identify without concession the gap between theory and practice.

4.3 Path-norm Rademacher bounds via covering numbers

For a general set of functions $\mathcal{F}$, the so-called Dudley's inequality guarantees that the Rademacher complexity can be bounded if we know how to control covering numbers of $\mathcal{F}$ with respect to the pseudo-metric

$$\|f(x) - g(x)\|_2$$

for all inputs x and all functions $f, g \in \mathcal{F}$. For DAG networks with $f = R_\theta$ and $g = R_{\theta'}$, I showed how to control this in terms of the ℓ^1-path-metric in Theorem 3.4.1, as soon as $\theta_i \theta'_i \geqslant 0$ for all i:

$$\|f(x) - g(x)\|_2 \leqslant \max(\|x\|_\infty, 1) \|\Phi(\theta) - \Phi(\theta')\|_1.$$

In this section, I build upon this to control covering numbers of $\mathcal{F} := \{R_\theta, \theta \in \Theta\}$ *without* sign assumptions ($\theta_i \theta'_i \geqslant 0$) on the pairs $\theta, \theta' \in \Theta$. Thanks to that, I derive *the first ℓ^1-path-norm-based Rademacher bound valid for general DAG networks (Theorem 4.3.1). This bridges the gap between networks that are theoretically analyzable with path-norm and the ones used in practice.*

4.3.1 Main result

The main result of this section is the following theorem: a Rademacher bound for general DAG networks that depends on the ℓ^1-path-norm of the parameters.

Theorem 4.3.1 (Path-norm Rademacher bound via covering numbers [Gonon et al., 2024b])**.** *Consider a DAG network G (Definition 2.2.2) , $\Theta \subset \mathbb{R}^G$ and the associated functions $\mathcal{F}_\Theta := \{R_\theta, \theta \in \Theta\}$. Consider n iid input samples $S = (x_i)_{i=1,\dots,n}$ drawn according to a distribution μ_x and denote $\sigma = \mathbb{E}_S \left(\sum_{i=1}^n \max(1, \|x_i\|_\infty^2)\right)^{1/2}$. Denote by $r := \sup_{\theta \in \Theta} \|\Phi(\theta)\|_1$. We have:*

$$\mathcal{R}(\mathcal{F}_\Theta, \mu_x) \leqslant 144 \sigma \max(D, d_{out}) \sqrt{\#\text{params}} \times r, \tag{4.15}$$

where we recall that $D = \max_{p \in \mathcal{P}} \texttt{length}(p)$ is the depth of the graph, $d_{out} = |N_{out}|$ is the output dimension, and $\#$params is the number of coordinates in a given $\theta \in \mathbb{R}^G$.

Note that #params counts all the coordinates of θ, even if some are *shared*, i.e., constrained to be equal. Weight-sharing occurs for instance for convolutional layer. For a convolutional kernel K, the bound is counting all the coefficients in the matrix corresponding to the linear transformation induced by this kernel, which contains many repetitions of the coefficients in K. It is possible to *improve* Theorem 4.3.1 in the case of weight-sharing to a bound that only counts the number of *free* coordinate parameters. For the sake of simplicity, I do not consider weight-sharing in this section. There is no particular difficulty to extend to weight-sharing, it just needs to be done properly. I include the details for completeness in Appendix C.2.

The next section improves Theorem 4.3.1 using a different proof technique that completely removes the number of parameters from the bound. *I decided to still include Theorem 4.3.1 in this manuscript for three reasons: 1) chronologically, it was the first Rademacher bound I proved, which then convinced me that it was indeed possible to extend the statistical analysis based on the path-norm to general DAG networks, 2) the proof technique is different from the one I used in the next section so it provides an alternative approach for future improvements, and 3) this proof technique explicitly demonstrates how any Lipschitz control of $\theta \mapsto R_\theta(x)$ directly implies a generalization bound via covering numbers.*

The proof of Theorem 4.3.1 is given at the end of Section 4.3. Let me first introduce intermediary results: the classical Dudley's inequality that bounds the Rademacher complexity using so-called covering numbers.

4.3.2 Dudley's integral

Let me start by recalling the definition of covering numbers (see, e.g., Definition 5.5 in Van Handel [2014]).

Definition 4.3.1. *Consider a pseudo-metric space $(\mathcal{F}, d)$. Let $B_t(f)$ be the closed ball centered at $f \in \mathcal{F}$ of radius $t > 0$. A family $f_1, \dots, f_n$ of points of $\mathcal{F}$ is called a t-covering of $(\mathcal{F}, d)$ if $\mathcal{F} \subset \cup_{i=1}^n B_t(f_i)$. The covering number of $(\mathcal{F}, d)$ for radius $t > 0$, denoted $\mathcal{N}(\mathcal{F}, d, t)$, is the minimum cardinality of a t-covering of $(\mathcal{F}, d)$.*

In the sequel, I will use the classical Dudley's inequality Shalev-Shwartz and Ben-David [2014], Van Handel [2014], Maurer [2016]. Since the statements in the literature always have small variations, at least in notations, I include the proof for completeness in Appendix C.1 that is essentially based on Corollary 5.25 in Van Handel [2014].

Lemma 4.3.1 (Corollary 5.25 in [Van Handel, 2014])**.** *Consider a set $\mathcal{F}$ of measurable functions from $\mathbb{R}^{d_{in}}$ to $\mathbb{R}^{d_{out}}$. Consider a probability distribution μ_x on $\mathbb{R}^{d_{in}}$. For n iid samples $S = (x_i)_{i=1}^n \sim (\mu_x)^{\otimes n}$, denote $f(S) = (f(x_i))_{i=1}^n$ for any $f \in \mathcal{F}$ and define the pseudo-metric d_S on $\mathcal{F}$ by:*

$$d_S(f,g)^2 := \|f(S) - g(S)\|_2^2 = \sum_{i=1}^n \|f(x_i) - g(x_i)\|_2^2 . \tag{4.16}$$

We have (recalling the definition of the Rademacher complexity in Equation (4.9) and covering numbers in Definition 4.3.1)

$$\mathcal{R}(\mathcal{F}, \mu) \leqslant 12 \mathbb{E}_S \left(\int_0^\infty \sqrt{\ln \mathcal{N}(\mathcal{F}, d_S, t)} dt \right) . \tag{4.17}$$

My goal is now to bound the covering numbers $\mathcal{N}(\mathcal{F}, d_S, t)$ in terms of the path-norm when $\mathcal{F} = \{R_\theta, \theta \in \Theta\}$ corresponds to DAG network functions. Lemma 4.3.1 shows that it is sufficient to control each $\|R_\theta(x) - R_{\theta'}(x)\|_2$. Using Theorem 3.4.1, we can lift to covering numbers of the path-lifting image $\Phi(\Theta)$ with respect to the ℓ^1-norm. This leads to the next result, the main result of Section 4.3.2.

Theorem 4.3.2 (Reducing to covering numbers of $(\Phi(\Theta), \|\cdot\|_1)$ [Gonon et al., 2024b])**.** *Consider a DAG network G (Definition 2.2.2) and a set of associated parameters $\Theta \subset \mathbb{R}^G$. Denote by $r = \sup_{\theta\in\Theta} \|\Phi(\theta)\|_1$. Consider the set of functions $\mathcal{F}_\Theta := \{R_\theta, \theta \in \Theta\}$, n training samples $S = (x_i)_{i=1}^n$ and define the pseudo metric $d_S(R_\theta, R_{\theta'}) := (\sum_{i=1}^n \|R_\theta(x_i) - R_{\theta'}(x_i)\|_2^2)^{1/2}$. Define also $\sigma_S = (\sum_{i=1}^n \max(1, \|x_i\|_\infty^2))^{1/2}$. Denote by Θ^* the set of parameters with only nonzero coordinates,* $\mathrm{sgn}(\Theta) = \{\mathrm{sgn}(\theta), \theta \in \Theta^*\}$ *the associated set of sign vectors (with* $\mathrm{sgn}(x) = \mathbb{1}_{x\geqslant 0} - \mathbb{1}_{x\leqslant 0} \in \{-1, 0, 1\}$*), and for each $s \in \mathrm{sgn}(\Theta)$ denote $\Theta_s = \Theta \cap \{\theta : \theta_i s_i \geqslant 0, \forall i\}$. We have for any $t > 0$:*

$$\mathcal{N}(\mathcal{F}_\Theta, d_S, t) \leqslant |\mathrm{sgn}(\Theta)| \max_{s\in\mathrm{sgn}(\Theta)} \mathcal{N}(\Phi(\Theta_s), \|\cdot\|_1, t/\sigma_S).$$

Improved bound for t large enough. *If $t \geqslant 2\sigma_S r$, we have $\mathcal{N}(\mathcal{F}_\Theta, d_S, t) = 1$.*

The number of signs in Theorem 4.3.2 is one of the reason why the number of parameters #params appears in the final Rademacher bound of Theorem 4.3.1. But this is not the only reason for that, as we will see that even for fixed signs s, the covering number of $(\Phi(\Theta_s), \|\cdot\|_1)$ that I derive below (Theorem 4.3.3) also makes #params appear in the bound of Theorem 4.3.1. These two occurrences of #params are additive, and even if we could improve one of them, the other one would still be there. I will point out the two occurrences of #params in the proof of Theorem 4.3.1. Let me now turn to the proof of Theorem 4.3.2.

Proof of Theorem 4.3.2. Recall that Θ^* is the set of parameters with only nonzero coordinates, $\mathrm{sgn}(\Theta) = \{\mathrm{sgn}(\theta), \theta \in \Theta^*\}$ is the associated set of sign vectors, and for each $s \in \mathrm{sgn}(\Theta)$, $\Theta_s = \Theta \cap \{\theta : \theta_i s_i \geqslant 0, \forall i\}$. Therefore, $\mathcal{F}_\Theta = \cup_{s\in\mathrm{sgn}(\Theta)} \mathcal{F}_{\Theta_s}$ and the union of t-coverings of each $\mathcal{F}_{\Theta_s}$ is a t-covering of $\mathcal{F}_\Theta$. So for each $t > 0$ we have

$$\mathcal{N}(\mathcal{F}_\Theta, d_S, t) \leqslant \sum_{s\in\mathrm{sgn}(\Theta)} \mathcal{N}(\mathcal{F}_{\Theta_s}, d_S, t).$$

Theorem 3.4.1 implies (through Corollary 3.4.1) that for each s, and every $\theta, \theta' \in \Theta_s$:

$$d_S(R_\theta, R_{\theta'}) = \left(\sum_{i=1}^n \|R_\theta(x_i) - R_{\theta'}(x_i)\|_2^2\right)^{1/2} \leqslant \sigma_S \|\Phi(\theta) - \Phi(\theta')\|_1,$$

Thus, a t-covering of $(\mathcal{F}_{\Theta_s}, d_S)$ is obtained by picking an arbitrary pre-image by the path-lifting Φ of each element in a t/σ_S-covering of $(\Phi(\Theta_s), \|\cdot\|_1)$, hence

$$\mathcal{N}(\mathcal{F}_{\Theta_s}, d_S, t) \leqslant \mathcal{N}\left(\Phi(\Theta_s), \|\cdot\|_1, t/\sigma_S\right) \leqslant \max_{s\in\mathrm{sgn}(\Theta)} \mathcal{N}(\Phi(\Theta_s), \|\cdot\|_1, t/\sigma_S).$$

Putting the pieces together proves the desired result for a general $t > 0$.

Improved bound for t large enough. For any parameters θ and input x, it holds

$$\|R_\theta(x)\|_2 = \|R_\theta(x) - R_0(x)\|_2 \underset{\text{Corollary 3.4.1}}{\leqslant} \max(1, \|x\|_\infty)\|\Phi(\theta) - \Phi(0)\|_1 = \max(1, \|x\|_\infty)\|\Phi(\theta)\|_1.$$

Recall that $r = \sup_{\theta\in\Theta} \|\Phi(\theta)\|_1$. Therefore, for every $\theta, \theta' \in \Theta$ and every input x, we have

$$\|R_\theta(x) - R_{\theta'}(x)\|_2 \leqslant \|R_\theta(x)\|_2 + \|R_{\theta'}(x)\|_2 \leqslant 2\max(1, \|x\|_\infty)r.$$

Since $\sigma_S^2 = \sum_{i=1}^n \max(1, \|x_i\|_\infty^2)$, we have:

$$d_S(R_\theta, R_{\theta'})^2 = \sum_{i=1}^n \|R_\theta(x_i) - R_{\theta'}(x_i)\|_2^2 \leqslant 4\sigma_S^2 r^2.$$

This shows that any single $R_\theta \in \mathcal{F}_\Theta$ is a $2\sigma_S r$-covering of $\mathcal{F}_\Theta$ with respect to d_S, and concludes the proof. □

As a corollary, we get a bound on the Dudley's integral for the set of functions realized by a general DAG network (Definition 2.2.2).

Corollary 4.3.1 ([Gonon et al., 2024b])**.** *Consider the setup of Theorem 4.3.2. For $t > 0$, define the shorthand notation*

$$\mathcal{N}(t) := \max_{s\in\operatorname{sgn}(\Theta)} \mathcal{N}(\Phi(\Theta_s), \|\cdot\|_1, t). \tag{4.18}$$

We have:

$$\int_0^\infty \sqrt{\ln \mathcal{N}(\mathcal{F}_\Theta, d_S, t)}dt \leqslant 2r\sigma_S\sqrt{\ln|\operatorname{sgn}(\Theta)|} + \sigma_S \int_0^{2r} \sqrt{\ln(\mathcal{N}(u))}\, du. \tag{4.19}$$

Proof of Corollary 4.3.1. Theorem 4.3.2 guarantees that $\mathcal{N}(\mathcal{F}_\Theta, d_S, t) = 1$ for $t \geqslant 2\sigma_S r$ so we have:

$$\int_0^\infty \sqrt{\ln \mathcal{N}(\mathcal{F}_\Theta, d_S, t)}dt = \int_0^{2\sigma_S r} \sqrt{\ln \mathcal{N}(\mathcal{F}_\Theta, d_S, t)}dt.$$

I now bound the latter using the general case of Theorem 4.3.2:

$$\begin{aligned}\int_0^{2\sigma_S r} \sqrt{\ln \mathcal{N}(\mathcal{F}_\Theta, d_S, t)}\, dt &\leqslant \int_0^{2\sigma_S r} \sqrt{\ln\left(|\operatorname{sgn}(\Theta)|\,\mathcal{N}(t/\sigma_S)\right)}\, dt \\ \overset{u=t/\sigma_S}{=} \sigma_S \int_0^{2r} \sqrt{\ln\left(|\operatorname{sgn}(\Theta)|\,\mathcal{N}(u)\right)}\, du &\leqslant 2r\sigma_S\sqrt{\ln|\operatorname{sgn}(\Theta)|} + \sigma_S \int_0^{2r} \sqrt{\ln(\mathcal{N}(u))}\, du.\end{aligned}$$ □

4.3.3 Bounding covering numbers in the path-lifting space

Corollary 4.3.1 shows that in order to bound the Dudley's integral, it is sufficient to bound the covering numbers of each $(\Phi(\Theta_s), \|\cdot\|_1)$. Let me start by noticing that in many cases, these covering numbers are independent on the choice of the signs s.

Lemma 4.3.2 ([Gonon et al., 2024b])**.** *Consider the setting of Theorem 4.3.2. Denote by $\mathbf{1}$ the vector constant equal to one and by $|\theta| \in \Theta_{\mathbf{1}}$ the vector deduced from $\theta \in \Theta_s$ by applying $x \mapsto |x|$ coordinate-wise. Consider $s \in \operatorname{sgn}(\Theta)$. If the map $x \in \Theta_s \mapsto |x| \in \Theta_1$ is one-to-one (with inverse $x \in \Theta_{\mathbf{1}} \mapsto s \odot x \in \Theta_s$), then*

$$\mathcal{N}(\Phi(\Theta_s), \|\cdot\|_1, t) = \mathcal{N}(\Phi(\Theta_{\mathbf{1}}), \|\cdot\|_1, t).$$

In particular, Lemma 4.3.2 applies to $\Theta := \{\theta, \|\Phi(\theta)\|_1 \leqslant r\}$.

Proof. For every $\theta, \theta' \in \Theta_s$, it is easy to check that by definition (Definition 2.3.3) $\|\Phi(\theta) - \Phi(\theta')\|_1 = \|\Phi(|\theta|) - \Phi(|\theta'|)\|_1$. This shows that under the assumptions, there is a one-to-one correspondence between the t-coverings of $(\Phi(\Theta_s), \|\cdot\|_1)$ and of $(\Phi(\Theta_{\mathbf{1}}), \|\cdot\|_1)$. □

Let me now establish bounds for the covering numbers of each $(\Phi(\Theta_s), \|\cdot\|_1)$. Using standard bounds for covering $\Phi(\Theta)$ would lead to an *exponential dependence on the ambient dimension* of $\Phi(\Theta)$, which is undesirable since the number of paths is combinatorial (as already discussed at the beginning of Section 3.1). My aim is to *avoid this exponential dependence in* #paths, and instead having an exponential dependence on the *degrees of freedom* of $\Phi(\Theta)$. Although little is known about the image $\Phi(\Theta)$, it is possible to intuitively infer its degrees of freedom: it should be equal to *the number of parameters minus the degrees of freedom associated with rescaling symmetries*. Indeed, $\Phi(\theta)$ is the image of Θ, with a dimension corresponding to *the number of parameters*, and mapping through Φ removes *the redundancy due to rescaling symmetries*. Recent findings support this, showing that $\Phi(\Theta)$ has a manifold dimension bounded by the number of parameters [Bona-Pellissier et al., 2022, Theorem 7]. In line with this, let me now demonstrate that the intuitive number of degrees of freedom I inferred above can indeed *replace the algebraic ambient dimension* of $\Phi(\Theta)$ (the number of paths) in *standard coverings*. This is the main result of Section 4.3.3.

Theorem 4.3.3 ([Gonon et al., 2024b])**.** *Consider a DAG network G and $\Theta \subset \mathbb{R}^G$. Denote by $r := \sup_{\theta\in\Theta} \|\Phi(\theta)\|_1$, #params the number of coordinates of any $\theta \in \mathbb{R}^G$, and by #rescalings the number of hidden neurons of G (i.e., the dimension of a rescaling vector $\lambda \in \mathbb{R}^H_{>0}$, see Definition 2.4.1). It holds:*

$$\mathcal{N}(\Phi(\Theta), \|\cdot\|_1, t) \leqslant 2^{\#\text{rescalings}} \max\left(1, \frac{24\max(D, d_{out})r}{t}\right)^{\#\text{params}-\#\text{rescalings}}$$

where we recall that $D = \max_{p\in\mathcal{P}} \texttt{length}(p)$ is the depth of the graph and $d_{out} = |N_{out}|$ is the output dimension.

Theorem 4.3.3 does not make any signs assumptions on the parameters. However, we will only apply it to sets of parameters with fixed signs because of Theorem 4.3.2 that reduced the problem to bounding such covering numbers. I did not try to derive something finer as I do not expect that taking into account the signs would lead to a large gain in Theorem 4.3.3. Indeed, for an ℓ^q-ball in dimension d and radius r, fixing the signs of the parameters reduces the volume of the ball from $\propto r^d$ to $\propto (r/2)^d$. In our case, we are not working with balls, but rather with intersections of balls with the image of the path-lifting Φ. If we still build our intuition on balls, taking into account the fixed signs would only replace r by $r/2$ in the bound of Theorem 4.3.3, only improving the constant 24 by a factor 2.

The proof of Theorem 4.3.3 relies on the comparison between the ℓ^1-path-metric and classical norms on the raw parameters, just as the ones established in Section 3.4.2 in Chapter 3. To get the tightest possible comparison, I carefully select in the weak equivalence class (Definition 2.4.3) of the ℓ^1-path-metric the representation of the parameters that is the most favorable to the covering number. This again leads to considering *normalized* parameters (Definition 3.2.1). The proof is not of particular interest and is a

bit tedious so I leave it to Appendix C.2.3, where I directly prove the same result with possible weight-sharing.

4.3.4 Proof of the main result, Theorem 4.3.1

Let me now prove the main result of Section 4.3.

Proof of Theorem 4.3.1. Consider Θ^* the set of parameters with only nonzero coordinates, $\operatorname{sgn}(\Theta) = \{\operatorname{sgn}(\theta), \theta \in \Theta^*\}$ the associated set of sign vectors (with $\operatorname{sgn}(x) = \mathbb{1}_{x \geqslant 0} - \mathbb{1}_{x \leqslant 0} \in \{-1, 0, 1\}$), and for each $s \in \operatorname{sgn}(\Theta)$ denote $\Theta_s = \Theta \cap \{\theta : \theta_i s_i \geqslant 0, \forall i\}$. For each $u > 0$, denote

$$\mathcal{N}(u) := \mathcal{N}(\Phi(\Theta_{\mathbf{1}}, \|\cdot\|_1, u).$$

We have:

$$\begin{aligned}
\mathcal{R}(\mathcal{F}_\Theta, \mu_x) &\leqslant 12\mathbb{E}_S\left(\int_0^\infty \sqrt{\ln \mathcal{N}(\mathcal{F}_\Theta, d_S, t)}dt\right) && \textit{Lemma 4.3.1} \\
&\leqslant \mathbb{E}_S\left(2r\sigma_S\sqrt{\ln|\operatorname{sgn}(\Theta)|} + \sigma_S\int_0^{2r}\sqrt{\ln(\mathcal{N}(u))}\,du\right). && \textit{Corollary 4.3.1}
\end{aligned}$$

The number $|\operatorname{sgn}(\Theta)|$ of signs is at most equal to $2^{\#\text{params}}$. Moreover, Theorem 4.3.3 guarantees for every $u > 0$

$$\begin{aligned}
\mathcal{N}(u) &= \mathcal{N}(\Phi(\Theta_{\mathbf{1}}), \|\cdot\|_1, u) \\
&\leqslant 2^{\#\text{rescalings}} \max\left(1, \left(\frac{24\max(D, d_{\text{out}})r}{u}\right)^{\#\text{params}-\#\text{rescalings}}\right). \qquad (4.20)
\end{aligned}$$

We get

$$\begin{aligned}
\int_0^{2r}\sqrt{\ln(\mathcal{N}(u))}du &= \int_0^{2r}\sqrt{\ln\left(2^{\#\text{rescalings}}\left(\frac{24\max(D, d_{\text{out}})r}{u}\right)^{\#\text{params}-\#\text{rescalings}}\right)}du \\
&\leqslant 2r\sqrt{\ln(2)\#\text{rescalings}} + \sqrt{\#\text{params} - \#\text{rescalings}}\int_0^{2r}\sqrt{\ln\left(\frac{24\max(D, d_{\text{out}})r}{u}\right)}du.
\end{aligned}$$

For the last integral, do a change of variable $t = u/24\max(D, d_{\text{out}})r$ to get:

$$\begin{aligned}
\int_0^{2r}\sqrt{\ln\left(\frac{24\max(D, d_{\text{out}})r}{u}\right)}du &= 24\max(D, d_{\text{out}})r\int_0^{1/12\max(2D, d_{\text{out}})}\sqrt{\ln(1/t)}dt \\
&\leqslant 24\max(D, d_{\text{out}})r\underbrace{\int_0^{1/12}\sqrt{\ln(1/t)}dt}_{\leqslant 1/3} \\
&\leqslant 8\max(D, d_{\text{out}})r.
\end{aligned}$$

Putting everything together, we get:

$$\begin{aligned}
\mathcal{R}(\mathcal{F}_\Theta, \mu_x) &\leqslant 12\mathbb{E}_S\left(2r\sigma_S\sqrt{\ln|\operatorname{sgn}(\Theta)|} + \sigma_S\int_0^{2r}\sqrt{\ln(\mathcal{N}(u))}\,du\right)\\
&\leqslant 12\mathbb{E}_S\left(2r\sigma_S\sqrt{\ln(2)\#\text{params}} + 2r\sigma_S\sqrt{\ln(2)\#\text{rescalings}}\right.\\
&\quad \left.+8r\sigma_S\max(D, d_{\text{out}})\sqrt{\#\text{params} - \#\text{rescalings}}\right)\\
&\leqslant 24\sigma r\left(\sqrt{\ln(2)\#\text{params}} + \sqrt{\ln(2)\#\text{rescalings}}\right.\\
&\quad \left.+4\max(D, d_{\text{out}})\sqrt{\#\text{params} - \#\text{rescalings}}\right)\\
&\leqslant 24\sigma r\sqrt{\#\text{params}}\left(\underbrace{2\sqrt{\ln(2)}}_{\simeq 1.67 \leqslant 2\max(D, d_{\text{out}})} + 4\max(D, d_{\text{out}})\right)\\
&\leqslant 144\sigma r\sqrt{\#\text{params}}.
\end{aligned}$$

This concludes the proof. Along the way, we see in the previous inequalities that $\sqrt{\#\text{params}}$ appears in the bound *both because of the number of signs (Theorem 4.3.2) and because of the degrees of freedom appearing in the covering bound of each* $(\Phi(\Theta_s), \|\cdot\|_1)$ *(Theorem 4.3.3).* □

4.3.5 Discussion on Theorem 4.3.1

Let me discuss the main result of this section, Theorem 4.3.1, which is a bound on the Rademacher complexity of neural networks based on path-norm.

Recall that before Theorem 4.3.1, the generalization guarantees based on the path-norm were only established for *LFCNs* [Neyshabur et al., 2015, Barron and Klusowski, 2019]. Thanks to Theorem 4.3.1, we now know that the path-norm can also be used to derive generalization guarantees for *general DAG networks* (Definition 2.2.2).

However, if we look at the dependencies in Theorem 4.3.1, we see that they are worse than those obtained for *LFCNs*. Indeed, this can be seen by comparing in Table 4.1 the line corresponding to [Barron and Klusowski, 2019], which does *not* contain any dependence on #params, and the line corresponding to Theorem 4.3.1, which *has* a dependence on #params. This suggests that the bound in Theorem 4.3.1 is *not the best we could hope for*.

In the next section, I improve on Theorem 4.3.1 by extending to *general DAG networks the best known Rademacher bound*, which is the one of Barron and Klusowski [2019] and that is established in the specific case of *scalar-valued LFCNs without biases* (see again Table 4.1).

Table 4.1: Bounds on the Rademacher complexity $\mathcal{R}(\mathcal{F}, \mu_x)$ (up to universal multiplicative constants) for the set $\mathcal{F} := \{R_\theta, \|\Phi(\theta)\|_1 \leqslant r\}$ associated to a DAG network G or a LFCN. The exception is the result of Golowich et al. [2018] where LFCNs without biases are considered, and the set $\mathcal{F}$ is defined via a constraint on the product of layers' norms: $\mathcal{F} := \{R_\theta, \theta = (M_1, \ldots, M_L), \prod_{d=1}^{D} \|M_\ell\|_{1,\infty} \leqslant R\}$. The distribution μ_x on the inputs is arbitrary, and for n iid input samples $S = (x_i)_{i=1,\ldots,n}$ drawn according to μ_x, we denote by $\sigma = \mathbb{E}_S \left(\sum_{i=1}^{n} \max(1, \|x_i\|_\infty^2)\right)^{1/2}$. In this table, $d_{\text{in}}/d_{\text{out}}$ are the input/output dimensions, #params is the number of parameters without redundancy when there is weight-sharing, $K = \max_{v \in N_{*\text{-pool}}} |\operatorname{ant}(v)|$ is the maximum kernel size (see Definition 2.2.2 in the appendix) of the $*$-max-pooling neurons, and D is the depth (maximum length of a path, which coincides with the number of affine layers for LFCNs).

	Architecture	Generalization bound
[Kakade et al., 2008, Eq. (5)] [Bach, 2024, Sec. 4.5.3]	LFCN with depth $D = 1$, no bias, $d_{\text{out}} = 1$ (linear regression)	$r \times \sigma\sqrt{\ln(d_{\text{in}})}$
[E et al., 2022, Thm. 6] [Bach, 2017, Proposition 7]	LFCN with $D = 2$, no bias, $d_{\text{out}} = 1$ (two-layer network)	$r \times \sigma\sqrt{\ln(d_{\text{in}})}$
[Neyshabur et al., 2015, Corollary 7]	DAG, no bias, $d_{\text{out}} = 1$	$r \times \sigma 2^D \sqrt{\ln(d_{\text{in}})}$
[Golowich et al., 2018, Theorem 3.2]	LFCN with arbitrary D, no bias, $d_{\text{out}} = 1$	$R \times \sigma\sqrt{D + \ln(d_{\text{in}})}$
[Barron and Klusowski, 2019, Corollary 2]	LFCN with arbitrary D, no bias, $d_{\text{out}} = 1$	$r \times \sigma\sqrt{D + \ln(d_{\text{in}})}$
This thesis, Theorem 4.3.1 (and Theorem C.2.2 for weight-sharing)	DAG, with biases, arbitrary d_{out}, with ReLU, identity and $*$-max-pooling neurons	$r \times \sigma \max(D, d_{\text{out}})\sqrt{\#\text{params}}$
This thesis, Theorem 4.4.1	DAG, *with biases*, arbitrary d_{out}, with ReLU, identity and k-max-pooling neurons for $k \in \{k_1, \ldots, k_P\} \subset \{1, \ldots, K\}$	$r \times \sigma\sqrt{D \ln(PK) + \ln(d_{\text{in}} d_{\text{out}})}$

4.4 Path-norm Rademacher bounds via peeling

This section improves on the previous generalization bound by using a different proof technique, inspired by the peeling technique originally given in Golowich et al. [2018] in the case of scalar-valued layered fully-connected ReLU networks. In order to adapt it to more practical networks, this new generalization bound relies on the following main theoretical contributions: *a new contraction lemma (to deal with max-pooling neurons) and a new peeling argument (because the network is not layered anymore).*

4.4.1 New contraction and peeling lemmas

Denote by $\mathcal{F}_L$ the set of functions realized by a LFCN with L affine layers, with some layer norm bounded by r. The peeling argument of Golowich et al. [2018] peels off the network one *layer* at a time:

$$\mathcal{R}(\mathcal{F}_L, \mu) \leqslant r \times \mathcal{R}(\mathcal{F}_{L-1}, \mu)$$

and every time a layer gets peeled off, its norm r appears.

In our case, peeling off layers is impossible since the network is *not* assumed to be layered, for instance because of the possible presence of skip-connections. Instead, I will peel off neurons, as I now describe when there is a single output neuron v:

$$\mathcal{R}(\mathcal{F}_\Theta, \mu) = \mathbb{E}_{S,\varepsilon} \sup_\theta \sum_{i=1}^n \varepsilon_i v(\theta, x_i)$$

where the ε_i's are iid Rademacher variables ($\mathbb{P}(\varepsilon_i = 1) = \mathbb{P}(\varepsilon_i = -1) = 1/2$) independent of the samples $S = (x_i)_{i=1}^n \sim (\mu)^{\otimes n}$. Recall that for a DAG $G = (N, E)$ and a neuron $v \in N$, its set of antecedents in G is defined as $\mathrm{ant}(v) = \{u \in N, u \to v \in E\}$ (Definition 2.2.2), see Figure 4.2 for an example.

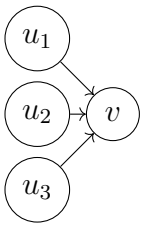


Figure 4.2: A DAG with a neuron v and its antecedents u_1, u_2, u_3.

Similarly, I will write $\mathrm{ant}^d(v)$ the set of neurons that can reach v with a path of length at most equal to d. My aim is to peel off the network one neuron at a time to write something like:

$$\begin{aligned}
\mathbb{E} \sup_\theta \sum_{i=1}^n \varepsilon_i v(\theta, x_i) &\leqslant \mathbb{E} \sup_\theta \max_{u \in \mathrm{ant}(v)} \sum_{i=1}^n \varepsilon_i u(\theta, x_i) \\
&\leqslant \mathbb{E} \sup_\theta \max_{u \in \mathrm{ant}^2(v)} \sum_{i=1}^n \varepsilon_i u(\theta, x_i) \\
&\leqslant \dots
\end{aligned}$$

Iterating this argument until reaching a maximum over input neurons u that satisfy $u(\theta, x_i) = x_i$, we will have reduced to:

$$\mathbb{E} \sup_{\theta} \sum_{i=1}^{n} \varepsilon_i x_i$$

which is easy to bound. There are two main difficulties to overcome: 1) the presence of $*$-max-pooling neurons, and 2) the fact that the proof of [Golowich et al., 2018] makes *local* complexity measures appear (that *isolates* the weights from one another), while I would like the path-norm $\|\Phi(\theta)\|_1$ to appear, a *global* complexity measure (that takes into account the *interactions* of the weights along the different paths). I now explain how to address these two difficulties.

From ReLU to $*$-max-pooling neurons. The peeling argument of Golowich et al. [2018] removes ReLU layers using a *contraction lemma* of the type

$$\mathbb{E} \sup_{t \in T} \varepsilon \operatorname{ReLU}(t) \leqslant \mathbb{E} \sup_{t \in T} \sum_{i} \varepsilon_i t_i$$

This result uses that the ReLU activation function has a *scalar-valued input*, is 1-Lipschitz and satisfies $f(0) = 0$. This is not the case for $*$-max-pooling neurons: it has a *vector-valued input*. To overcome this, I proved a new contraction lemma to cope with $*$-max-pooling neurons.

The classical contraction lemmas are of the type:

$$\mathbb{E} \sup_{t \in T} \varepsilon f(t) \leqslant \mathbb{E} \sup_{t \in T} \sum_{i} \varepsilon_i t_i$$

in a situation where f is the activation function of a neuron, t is the pre-activation of the neuron, and the epsilon variables are independent Rademacher variables. This type of contraction result is well-known for arbitrary T when $f : \mathbb{R} \mapsto \mathbb{R}$ is 1-Lipschitz and satisfies $f(0) = 0$ [Ledoux and Talagrand, 1991, Equation (4.20)]. I extended it to $*$-max-pooling neurons, which fail to fall into the classical framework because of their vector-valued input.

Lemma 4.4.1 ([Gonon et al., 2024a])**.** *Consider a finite set W, a set T of elements $t = (t_1, t_2) \in \mathbb{R}^W \times \mathbb{R}$ and a function $f : \mathbb{R}^W \to \mathbb{R}$. Consider also a convex non-decreasing function $F : \mathbb{R} \to \mathbb{R}$ and a family of iid Rademacher variables $(\varepsilon_j)_{j \in J}$ where J will be clear from the context. Assume that we are in one of the two following situations.*

Scalar input case. (already known, [Ledoux and Talagrand, 1991]) *f is 1-Lipschitz, satisfies $f(0) = 0$ and has a scalar input ($|W| = 1$).*

$*$-max-pooling case. (new) *There is $k \in \mathbb{N}_{>0}$ such that f computes the k-th largest coordinate of its input.*

Denoting $t_1 = (t_{1,w})_{w \in W}$, it holds:

$$\mathbb{E} \sup_{t \in T} F\left(\varepsilon_1 f(t_1) + t_2\right) \leqslant \mathbb{E} \sup_{t \in T} F\left(\sum_{w} \varepsilon_{1,w} t_{1,w} + t_2\right).$$

The proof is deferred to Appendix C.3. Let me now turn to the second obstacle to overcome, resolved with a new peeling lemma.

From local to global complexity measures using the symmetries. In the peeling argument of Golowich et al. [2018], peeling a layer make a layer's norm r appear. The norm of a layer can be understood as a *local* complexity measure: this kind of measure is blind to how this layer takes part in the global computations made by the network. In our case, we are going to peel off neurons, so we can again expect that this *local* surgery will make a *local* complexity measure appear. However, we need to relate this local complexity to the path-norm $\|\Phi(\theta)\|_1$, that can be understood as a *global* complexity measure since each weight is considered as being a part of all the paths going through it, and each path either contributes to the output or not.

In order to relate a local complexity to a global complexity, I will crucially use the *symmetries* of the network. Recall Theorem 3.3.2 in Chapter 3 that shows that if we carefully select the representation of the parameters in its weak equivalence class to be *normalized* parameters, we can make the path-norm $\|\Phi(\theta)\|_1$ coincide with local complexity measures. I will exploit this by first reducing to normalized parameters before peeling off neurons. Therefore, the new peeling lemma I establish is only stated for *normalized* parameters. Let me now state an informal version of this lemma, and leave the details to the appendix (Appendix C.4, Lemma C.4.3). The lemma shows that is is possible to peel the neurons at distance d of the output neurons, to reduce to the neurons at distance $d+1$, and *controls the constants that appear in the process in the specific case of normalized parameters*. The proof relies on the contraction inequality in Lemma 4.4.1.

Lemma 4.4.2. ***Informal version of Lemma C.4.3 [Gonon et al., 2024a].***
Consider a family of independent Rademacher variables $(\varepsilon_j)_{j\in J}$ with J that will be clear from the context. Consider a convex non-decreasing function $g:\mathbb{R}\to\mathbb{R}_{\geqslant 0}$. Define $P := |\{k\in\mathbb{N}_{>0}, \exists u\in N_{k\text{-pool}}\}|$ as the number of different types of $*$-max-pooling neurons in G, and $K := \max_{u\in N_{*\text{-pool}}} |\operatorname{ant}(u)|$ the maximal kernel size of the network ($K:=1$ if $P=0$). Assume the parameters to be 1-normalized, basically meaning that the vector of incoming weights of each neuron has ℓ^1-norm equal to one (Definition 3.2.1). It holds (highlighting in orange the changes):

$$\mathbb{E}_\varepsilon g\left(\max_{\substack{v\in N_{out},\\ m=1,\dots,M}} \max_{u\in \operatorname{ant}^d(v)} \sup_\theta \left|\sum_{i=1}^n \varepsilon_{i,v,m} u(\theta,x_i)\right|\right) \leqslant P\mathbb{E}_\varepsilon g\left(\max_{\substack{v\in N_{out},\\ m=1,\dots,KM}} \max_{u\in \operatorname{ant}^{d+1}(v)} \sup_\theta \left|\sum_{i=1}^n \varepsilon_{i,v,m} u(\theta,x_i)\right|\right).$$

Let me explain how the peeling lemma will be used. After a few easy steps on the Rademacher complexity:

$$\begin{aligned}\mathcal{R}(\mathcal{F}_\Theta,\mu) &= \mathbb{E}_{S,\varepsilon}\sup_\theta \sum_{i=1}^n \varepsilon_i v(\theta,x_i)\\ &= \mathbb{E}_{S,\varepsilon}\log\left(\exp\left(\sup_\theta\sum_{i=1}^n \varepsilon_{i,v} v(\theta,x_i)\right)\right)\\ &\underset{Jensen}{\leqslant} \mathbb{E}_S\log\left(\mathbb{E}_\varepsilon\exp\left(\sup_\theta\sum_{i=1}^n \varepsilon_{i,v} v(\theta,x_i)\right)\right)\end{aligned}$$

I will apply the peeling lemma on the last term, corresponding to $g(x) = \exp(x)$ in Lemma 4.4.2. This is to mitigate the apparition of the additional constants P and K at every application of Lemma 4.4.2, as they will here only appear in a logarithm, a trick introduced in Golowich et al. [2018].

4.4.2 Main result

Thanks to the new contraction and peeling lemmas presented above, it is now possible to improve on the Rademacher bound I established in the previous Section 4.3. While all the other results of this thesis hold for arbitrary biases, the next theorem holds **only if the $*$-max-pooling neurons have null biases**. In the model I introduced in Definition 2.2.2, max-pooling neurons, which are widely used in practice, correspond to 1-max-pooling neurons with *null* biases. *Therefore, the assumption of null biases on $*$-max-pooling neurons has no impact on the practical applicability of the result to classical max-pooling neurons.* For simplicity, I state the result when *all* the biases are null. I then explain how to extend the result when neurons $u \notin N_{*\text{-pool}}$ may have $b_u \neq 0$.

Theorem 4.4.1 (Path-norm Rademacher bound via peeling [Gonon et al., 2024a])**.** *Consider a DAG network G (Definition 2.2.2) with $N_{in} \cap N_{out} = \emptyset$, $\Theta \subset \mathbb{R}^G$ with null biases and denote $\mathcal{F}_\Theta := \{R_\theta, \theta \in \Theta\}$. Denote $r := \sup_\theta \|\Phi(\theta)\|_1$, d_{in}/d_{out} the input/output dimensions, D the depth of G (the maximal length of a path from an input to an output), $P := |\{k \in \mathbb{N}_{>0}, \exists u \in N_{k\text{-pool}}\}|$ the number of distinct types of $*$-max-pooling neurons in G, and $K := \max_{u \in N_{*\text{-pool}}} |\operatorname{ant}(u)|$ its maximal kernel size ($K := 1$ if $P = 0$). Consider n random samples $S = (x_i)_{i=1}^n$ drawn iid according to some distribution μ_x on $\mathcal{X}$ and denote $\sigma := \left(\mathbb{E}_{S\sim(\mu_x)^{\otimes n}} \max(n, \sum_{i=1}^n \|x_i\|_\infty^2)\right)^{1/2}$. We have:*

$$\mathcal{R}(\mathcal{F}_\Theta, \mu_x) \leqslant \sqrt{2}\sigma C r.$$

with ($\log$ being the natural logarithm)

$$C := \left(D \log((3+2P)K) + \log\left(\frac{3+2P}{1+P} d_{in} d_{out}\right)\right)^{1/2}.$$

Extension to the case with nonzero biases for non $*$-max-pooling-neurons. Any neural network with nonzero biases can be transformed into an equivalent network (with same path-norms and same R_θ) with null biases for every $v \notin N_{*\text{-pool}}$: add an input neuron $v_{\texttt{bias}}$ with constant input equal to one, add edges between this input neuron and every neuron $v \notin N_{*\text{-pool}}$ with parameter $\theta^{v_{\texttt{bias}} \to v} := b_v$, and set $b_v = 0$. Thus the same result holds with $b_v \neq 0$ for $v \notin N_{*\text{-pool}}$, with d_{in} replaced by $d_{\text{in}} + 1$ in the definition of C, and with an additional constant input coordinate equal to one in the definition of σ so that $\sigma = \left(\mathbb{E}_S \max\left(n, \max_{u=1,\dots,d_{\text{in}}} \sum_{i=1}^n (x_i)_u^2\right)\right)^{1/2} \geqslant \sqrt{n}$. The proof in Appendix C.5 is directly given for networks with nonzero biases (except $*$-max-pooling neurons), using this construction.

Comparison with other bounds. On ImageNet, it holds $1/\sqrt{n} \leqslant \sigma/n \leqslant 2.6/\sqrt{n}$ (as explained in the last Chapter 7, see Section 7.2). This yields a bounds that decays in $\mathcal{O}(n^{-1/2})$ which is better than the generic $\mathcal{O}(n^{-1/d_{\text{in}}})$ generalization bound for Lipschitz

functions [von Luxburg and Bousquet, 2004, Thm. 18] that suffer from the curse of dimensionality. Besides its wider range of applicability, this bounds also recovers or improves on the sharpest known ones based on the path-norm, see Table 4.1 for a comparison.

Extension of Theorem 4.4.1 to other models. Despite its applicability to a wide range of standard modern networks, the generalization bound in Theorem 4.4.1 does not cover networks with other activations than ReLU, identity, and $*$-max-pooling. The same proof technique could be extended to new activations that: 1) are positively homogeneous, so that the weights can be rescaled without changing the associated function; and 2) satisfy a contraction lemma similar to the one established here for ReLU and max neurons (typically requiring the activation to be Lipschitz). A plausible candidate is Leaky ReLU. For smooth approximations of the ReLU, such as the SiLU (for Efficient Nets) and the Hardswish (for MobileNet-V3), parts of the technical lemmas related to contraction may extend since they are Lipschitz, but these activations are not positively homogeneous.

Improving Theorem 4.4.1. Note that Theorem 4.4.1 can be tightened: the same bound holds without counting the neurons having the identity as activation function when computing D. Indeed, for any neural network and parameters θ, it is possible to remove all the neurons $v \in N_{\mathrm{id}}$ by adding a new edge $u \to w$ for any $u \in \mathrm{ant}(v), w \in \mathrm{suc}(v)$ with new parameter $\theta^{u\to v}\theta^{v\to w}$ (if this edge already exists, just add the latter to its already existing parameter). This still realizes the same function, with the same path-norm, but with less neurons, and thus with D possibly decreased. The proof technique would also yield a tighter bound but not by much: the occurrences of 3 in C would be replaced by 2.

Sketch of proof for Theorem 4.4.1. The proof idea is explained below. Details are in Appendix C.5.

Already known ingredients. In the case of *LFCNs no biases and scalar output*, Golowich et al. [2018] proved that it is possible to bound the Rademacher complexity with no exponential factor in the depth, by peeling, one by one, each layer off the Rademacher complexity. To get more specific, for a class of functions $\mathcal{F}$ and a function $\Psi : \mathbb{R} \to \mathbb{R}$, denote $\mathcal{R} \circ \Psi(\mathcal{F}) = \mathbb{E}_\varepsilon \Psi(\sup_{f\in\mathcal{F}} \sum_{i=1}^n \varepsilon_i f(x_i))$ the Rademacher complexity of $\mathcal{F}$ associated with n inputs x_i and Ψ, where the ε_i are iid Rademacher variables ($\varepsilon_i = 1$ or -1 with equal probability). The goal for a generalization bound is to bound this in the case $\Psi(x) = \mathrm{id}(x) = x$. In the specific case where $\mathcal{F}_D$ is the class of functions that correspond to layered fully-connected ReLU networks with depth D, assuming that some operator norm of each layer d is bounded by r_d, Golowich et al. [2018] basically guarantees $\mathrm{Rad} \circ \Psi_\lambda(\mathcal{F}_D) \leqslant 2\,\mathrm{Rad} \circ \Psi_{\lambda r_D}(\mathcal{F}_{D-1})$ for every $\lambda > 0$, where $\Psi_\lambda(x) = \exp(\lambda x)$. Compared to previous works of Golowich et al. [2018] that were directly working with $\Psi = \mathrm{id}$ instead of Ψ_λ, the important point is that working with Ψ_λ gets the 2 outside of the exponential. Iterating over the depth D, optimizing over λ, and taking a logarithm at the end yields (by Jensen's inequality) a bound on $\mathrm{Rad} \circ \mathrm{id}(\mathcal{F}_D)$ with a dependence on D that grows as $\sqrt{D\log(2)}$ instead of 2^D for previous approaches [Neyshabur et al., 2015], see Table 4.1.

Novelties for general DAG ReLU networks. Compared to the setup of Golowich et al. [2018], there are at least three difficulties to do something similar here. While I already mentioned them above (Section 4.4.1), let me summarize them here.

First, *the neurons are not organized in layers* as the model can be an arbitrary DAG. So what should be peeled off one by one? Second, *the neurons are not necessarily ReLU neurons* as their activation function might be the identity (average-pooling) or $*$-max-pooling. Finally, Golowich et al. [2018] has a *local* constraint on the weights of each layer, which makes it possible to pop out the constant r_d when layer d is being peeled off. *Here, the only constraint is global*, since it constrains the paths of the network through $\|\Phi(\theta)\|_1 \leqslant r$. In particular, due to rescalings, the weights of a given neuron could be arbitrarily large or small under this constraint.

The first difficulty is primarily addressed using a new peeling lemma that peels off *the neurons one at a time* (an informal version of this lemma has been given above in Lemma 4.4.2, the formal version is the appendix, see Appendix C.4). The second difficulty is addressed with a new contraction lemma for $*$-max-pooling neurons (it has been given in its simplest form in Lemma 4.4.1 above, and the full version is in appendix, see Appendix C.3), and splitting the ReLU, k-max-pooling and neurons with the identity as activation function in different groups before each peeling step. The number of different groups yields a $\log(2)$ term in Golowich et al. [2018], against a $\log(3+2P)$ term here (P being the number of different k's for which k-max-pooling neurons are considered). Finally, the third obstacle is overcome by *normalizing the parameters* with Algorithm 3.2.1. This type of rescaling has also been used for *LFCNs* in Neyshabur et al. [2015], Barron and Klusowski [2019]. □

Remark 4.4.1 (Improved bound with assumptions on $*$-max-pooling neurons)**.** *In the specific case where there is a single type of k-max-pooling neurons ($P = 1$), assuming that these k-max-pooling neurons are grouped in layers, and that there are no skip connections going over these k-max-pooling layers (satisfied by ResNets, not satisfied by U-nets), a sharpened peeling argument can yield the same bound but with C replaced by $C_{sharpened} = (D\log(3) + M\log(K) + \log((d_{in}+1)d_{out}))^{1/2}$ with M being the number of k-max-pooling layers (cf. Appendix C.4). The details are tedious so I only mention this result without proof. This basically improves $\sqrt{D\log(5K)}$ into $\sqrt{D\log(3)+M\log(K)}$. For Resnet152, $K = 9$, $D = 152$ and $M = 1$, $\sqrt{D\log(5K)} \simeq 24$ while $\sqrt{D\log(3)+M\log(K)} \simeq 13$.*

4.5 Conclusion

Section 4.2 explained how the norm of the path-lifting is a promising complexity measure to explain generalization properties of neural networks [Neyshabur et al., 2015, Kawaguchi et al., 2017, Barron and Klusowski, 2019, Jiang et al., 2020, Dziugaite et al., 2020]:

- it is one of the best weight-based existing measures in terms of correlation with the generalization gap,
- it is easy to compute,
- it is invariant to rescaling and permutation symmetries,
- and it is at least as fine as the product of layers' norms, a complexity measure that has widely been investigated for generalization, see, e.g., [Bartlett et al., 2017].

However, I already mentioned in the introduction of Chapter 2 that the path-lifting and its norm have been limited in the literature to simple network architectures unable to combine in a single framework standard ingredients such as pooling layers, skip-connections, biases or even multi-dimensional outputs [Neyshabur et al., 2015, Kawaguchi et al., 2017, Barron and Klusowski, 2019, Jiang et al., 2020, Dziugaite et al., 2020].

Due to this lack of versatility, the path-norm-based bounds *have only been tested for simple LFCNs*. This prevents us from both understanding the reach of the path-norm as a complexity measure for generalization, and from diagnosing its strengths and weaknesses, which is necessary to either improve this complexity measure in order to make it actually operational, if possible, or to identify without concession the gap between theory and practice.

This chapter filled this gap by establishing a path-norm-based generalization bound that is not only applicable for the first time to general DAG networks, but that also recovers or improves on all previous ones known in simpler settings (Table 4.1).

Perspectives. Thanks to that, we can now assess numerically the path-norm-based bounds in practical situation for the first time. I will assess the promises of the path-norm as a complexity measure for generalization by numerically evaluating the generalization bounds on ResNets trained on ImageNet in Section 7.2. I will show that there are still challenges ahead to improve these bounds, notably by investigating average-case measures based on the path-lifting, instead of worst-case ones.

Chapter 7 will also be the time to further reflect on the results of this chapter. First, I briefly mentioned after Theorem 4.1.1 that it is not clear how to strictly apply *in practice* the dominant statistical approach based on the Rademacher complexity. I will elaborate on this in Section 7.3. Second and last, I will discuss the challenge of theoretically accounting for subtle phenomena between the training set and the generalization of the network, which is not addressed by the path-norm-based bounds, see Section 7.3.

Chapter 5

Efficient inference with Kronecker-sparse matrices

This chapter tackles resource efficiency for neural networks, one of the main subjects addressed in this thesis. This corresponds to Gonon et al. [2024c] and it is disconnected from the theory on the path-lifting and path-activations developed in Chapters 2 to 4.

This chapter is the result of a collaboration with two PhD students, Léon Zheng and Quoc-Tung Le, and a research engineer, Pascal Carrivain, all of us being from the same research team at ENS Lyon. I was interested in the potential gains associated with the sparsity of neural networks. Léon and Tung are specialized in the sparse butterfly structure. When I asked them about practical gains in time and energy for this kind of sparsity, we realized that it wasn't clear: the literature results are not comprehensive enough to clearly identify the situations where there is a gain, and what kind of gain can be hoped for. However, it was clear to us that there was a potential. So we set as our first goal to benchmark existing implementations. While benchmarking, we noticed that memory transfers in existing implementations are very expensive, so we aimed at improving that by proposing our own CUDA implementation. This project was the opportunity for Pascal and me to discover CUDA, which led the two of us to jointly develop our own kernel that improved the state of the art. More details on the origin of the contributions will be provided within the chapter.

As briefly evoked in Chapter 1, the exponential growth in computational requirements for training state-of-the-art AI models has raised significant concerns regarding efficiency. According to an analysis made by OpenAI [OpenAI, May 2018], the amount of computations (floating point operations per second) for the largest AI training runs doubled every 3.4 months between 2012 and 2018 (300000× increase). Since then, similar exponential trends have been observed in various aspects of AI research such as for the state-of-the-art model size or energy consumption [Strubell et al., 2019, Kaplan et al., 2020, Wu et al., 2022, Narayanan et al., 2021, Bender et al., 2021, Sevilla et al., 2022, Sastry et al., 2024]. Models can have up to 1 trillion parameters [Narayanan et al., 2021, Bender et al., 2021, Tables 1] with an associated end-to-end training estimated to 3 months using 3072 A100 GPUs[1] [Narayanan et al., 2021, $Training times estimates]. The estimated CO2 emissions

[1]This would cost $25 million according to the Azure Pricing calculator.

for training[2] a Transformer with 10^8 parameters in 2019 was equivalent to an average 20 years of American life[3] [Strubell et al., 2019, Table 1]. Latest figures on Llama 3 models correspond to 150 years of American life [Meta, 2024]. And training is by far the less expensive phase as inference is reported to represent more than 90% of the cost of machine learning at scale according to independent reports from both NVIDIA [HPCwire, 2019] and Amazon Web Services [Barr, 2019]. This constantly growing resource requirements *calls for designing efficient models to achieve gains in time, energy and memory.*

At the very heart of neural network efficiency is the acceleration of matrix multiplication on GPU, which is one of the main operations during both training and inference. For instance, in a forward pass of a so-called vision transformer (ViT) [Dosovitskiy et al., 2021], between 30% and 60% of the total time is spent in fully-connected layers doing matrix multiplications (see Appendix D.2.3 for details). One key approach[4] that aims to accelerate computations is by enforcing *sparsity* constraints on certain weight matrices in the model.

The *Kronecker sparsity* has recently emerged as a promising form of sparsity for neural networks as it shows similar empirical accuracy to dense matrices in some settings, while having a sub-quadratic *theoretical* complexity for matrix-vector multiplication [Dao et al., 2022a]. A *Kronecker-sparse* matrix has a support defined by a *Kronecker* product $\mathbf{S}_\pi = \mathbf{I}_a \otimes \mathbf{1}_{b\times c} \otimes \mathbf{I}_d$ (Figure 5.2) for some tuple of integers $\pi = (a, b, c, d)$, see Definition 5.1.1 [Lin et al., 2021, Le, 2023], where $\mathbf{I}_n$ denotes the identity of size $n \times n$, and $\mathbf{1}_{n\times m}$ the matrix full of ones of size $n \times m$. *In order to stick to the conventional notations of* (a, b, c, d) *for a Kronecker sparsity pattern [Le, 2023], I drop the convention of using* d_{in} *and* d_{out} *for the input and output dimensions of the Kronecker-sparse matrix, and use* N *and* M *instead. Because of that, and only in this chapter, I will use bold capital letters* $\mathbf{M}$ *for matrices to avoid confusion with dimension integers* M*.*

The name "Kronecker-sparse matrix" is something I introduce in this thesis as a result of a discussion with Rémi Gribonval, one of my supervisor, and my co-authors Léon, Tung and Pascal [Gonon et al., 2024c]. It is meant to correspond to the building blocks of butterfly matrices [Lin et al., 2021, Le, 2023, Dao et al., 2019, Chen et al., 2022, Dao et al., 2022a, Alizadeh-Vahid et al., 2020, Lin et al., 2021, Fu et al., 2023]. The exact definition of *butterfly matrices* is still evolving in the literature [Dao et al., 2019, 2020, Lin et al., 2021, Dao et al., 2022a, Fu et al., 2023, Le, 2023], and is not yet fully satisfying as it is either very restrictive (e.g., only for square matrices with dyadic dimensions [Dao et al., 2019, Alizadeh-Vahid et al., 2020]) or too expressive (e.g., it also encapsulates dense matrices [Lin et al., 2021, Le, 2023]). However, all the definitions agree on the fact that they are products of matrices with Kronecker constraints on their supports, hence the name *Kronecker-sparse matrices* proposed here, and the particular focus on it in this chapter.

The fact that Kronecker-sparse matrices are the most basic building block of butterfly matrices is another motivation for studying them, as butterfly matrices are themselves promising. For instance, approximating a matrix by a *butterfly matrix* can be done

[2]Including tuning and experiments.

[3]You can also click on this link to estimate your machine learning carbon impact [Lacoste et al., 2019].

[4]The other key approach is *quantization*, that is complementary to sparsity. It consists in reducing the number of bits used to store the weights and do the computations. We will study the approximation properties of quantized networks in Chapter 6.

using an efficient algorithm that outperforms gradient descent[5], with some guarantees of reconstruction if the target matrix admits exactly or approximately a butterfly form [Le et al., 2022, Zheng et al., 2023, Le, 2023]; butterfly matrices can be quantized more efficiently than by naive rounding [Gribonval et al., 2023]; and butterfly matrices have a nearly linear *theoretical* complexity for matrix-vector multiplication. The Discrete Fourier Transform (DFT) or the Hadamard Transform are examples of linear operators with small *theoretical* complexity and whose associated matrix admits a butterfly decomposition. See Figure 5.1 for an example of the decomposition of the DFT matrix into a product of Kronecker-sparse matrices, in dimension 16.

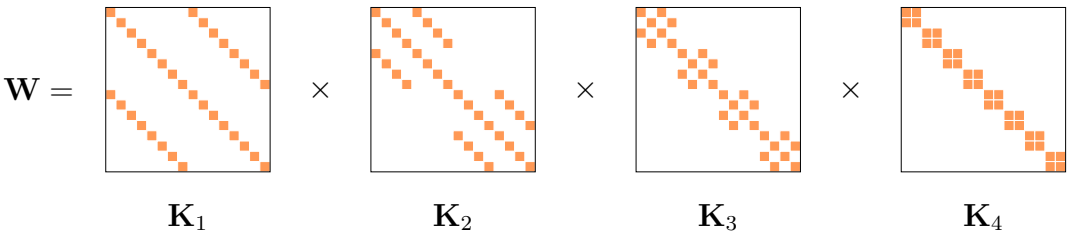


Figure 5.1: Example of butterfly factorization $\mathbf{W} = \mathbf{K}_1 \dots \mathbf{K}_L$, for $L = 4$. Here, the factor $\mathbf{K}_\ell \in \mathbb{R}^{N \times N}$ (with $N = 2^L$) has support $\mathbf{S}_\ell = \mathbf{I}_{2^{\ell-1}} \otimes \mathbf{1}_{2\times 2} \otimes \mathbf{I}_{2^{L-\ell}}$. This corresponds to the butterfly factorization of the Discrete Fourier Transform matrix $\mathbf{W}$, up to a permutation of its column indices.

In practice, the goal is to reparameterize a dense fully-connected layer $\mathbf{W}$ as a product of *Kronecker-sparse* matrices $\mathbf{W} = \mathbf{K}_1 \dots \mathbf{K}_L$ while having (i) at least the same accuracy for the learning task at hand, (ii) less parameters to store, and (iii) an accelerated inference and training phase. Previous works mostly focused on (i) and (ii) [Alizadeh-Vahid et al., 2020, Lin et al., 2021, Chen et al., 2022, Dao et al., 2022a].

This chapter addresses the challenge of **benchmarking the gains in time and energy** that can be achieved by exploiting the sparsity of the Kronecker-sparse matrices, and better understand the strengths and weaknesses of existing implementations. We also propose a **new CUDA kernel that fuses operations to decrease the cost of memory transfers between the different levels of GPU memory**.

The outline is as follows.

- Section 5.1 introduces the framework to study Kronecker-sparse matrix multiplication, and describes baseline GPU implementations on `PyTorch`.
- Section 5.2 reveals an important bottleneck of existing implementations, thanks to the new benchmark they spend up to 50% of their total runtime on GPU memory rewriting operations (Figure 5.5). Section 5.2 aims at explaining why. Essentially, this lack of memory efficiency is caused by the fact that existing implementations call high-performance libraries *from* `Python`, but `Python` lacks routines to explicitly control the GPU memory transfers.

[5]Let me detail what this means. Approximating a matrix by a butterfly matrix can be formulated as an optimization problem where we minimize the approximation error, measured, e.g., as the distance in Frobenius norm between the given matrix and a product of Kronecker-sparse factors. When I say that this algorithm "outperforms gradient descent", I mean that the algorithm in question can find a butterfly matrix not only associated with a *smaller approximation error* but also in *less time* than gradient descent.

- Section 5.3 introduces a new CUDA kernel that *fuses the operations to decrease the cost of the transfers between the different levels of GPU memory.*
- Section 5.4 benchmarks the execution time and the energy consumption of the new kernel and the other baseline `PyTorch` GPU implementations, for multiplying a batch of vectors with a *Kronecker-sparse* matrix. This includes implementations relying on existing efficient routines for batch GEMM[6], block-sparse matrix multiplication and tensor contraction. The goal is to provide a benchmark easily adaptable and useful to select the best implementation for given settings. The kernel not only improves the **speed of the computation**, but also the **energy efficiency**, with a median speed-up factor of $\times 1.4$ in *float-precision* and a median reduction factor of $\times 0.85$ in energy consumption. Moreover, we clearly identify the situations where the new kernel is advantageous: when the relative number of memory rewritings increases, confirming the relevance of our approach to reduce memory transfers. We also introduce a *theoretical complexity* associated with a Kronecker sparsity pattern (a, b, c, d), that we show that this *theoretical* complexity can be used to identify *a priori* the patterns that are *efficient in practice*. This leads to a new **heuristic to choose efficient Kronecker sparsity patterns** and paves the way to **new design of efficient Kronecker-sparse neural networks.**
- Section 5.5 concretely illustrates broader implications of this work: the new kernel can be used to speed up the inference of neural networks.

The code is available at `github.com/PascalCarrivain/ksmm`.

5.1 Background on Kronecker-sparse matrices

We call a *Kronecker-sparse* matrix any matrix whose *support* satisfies some specific constraint defined in terms of Kronecker products. Let me emphasize that this Kronecker structure is imposed only on the *support*, not on the values of the weights, which are free to take any value for those that are not zero.

Definition 5.1.1 (Kronecker-sparse matrix [Le, 2023])**.** *A* Kronecker sparsity pattern *(or simply Kronecker pattern) is a tuple* $\pi := (a, b, c, d) \in (\mathbb{N}_{>0})^4$*. A* π-Kronecker-sparse matrix *(or simply Kronecker-sparse matrix when* π *is clear from the context) is a matrix* $\mathbf{K} \in \mathbb{R}^{abd \times acd}$ *satisfying* $\text{support}(\mathbf{K}) \subseteq \text{support}(\mathbf{S}_\pi)$*, where* $\mathbf{S}_\pi := \mathbf{I}_a \otimes \mathbf{1}_{b\times c} \otimes \mathbf{I}_d$ *(see Figure 5.2) and where* $\text{support}(\mathbf{M}) := \{(i, j), \mathbf{M}_{i,j} \neq 0\}$*. The set of* π*-Kronecker-sparse matrices is denoted* Σ^π*.*

A π-Kronecker-sparse factor is *sparse* and *structured*. For $\pi = (a, b, c, d)$, it has at most $abcd$ nonzero entries, which yields a sparsity ratio $\frac{abcd}{a^2bcd^2} = \frac{1}{ad}$ since it is of size $abd \times acd$. Kronecker-sparse matrices can represent a *wide variety of matrices that have been used to train neural networks*, in particular so-called *butterfly matrices* that correspond to some products of Kronecker-sparse matrices [Lin et al., 2021, Le, 2023, Dao et al., 2019, Chen et al., 2022, Dao et al., 2022a, Alizadeh-Vahid et al., 2020, Lin et al., 2021, Fu et al., 2023]. *Léon and Tung, two of my co-authors, have created a table with the examples they*

[6] GEMM stands for General Matrix Multiplication.

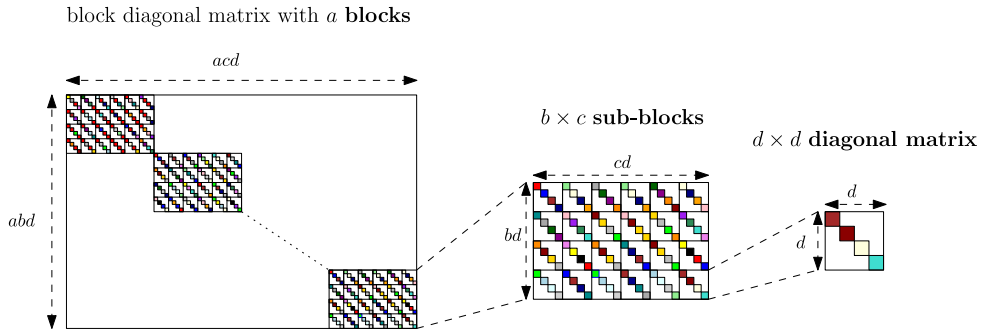


Figure 5.2: A π-Kronecker-sparse matrix with $\pi = (a, b, c, d)$ is a block-diagonal matrix with a blocks, where each block itself is a block matrix composed by $b \times c$ diagonal matrices of size $d \times d$. The colored cells correspond to the nonzeros. We color the cells with different colors to indicate that the corresponding weights are free to take different values. *Courtesy of Tung for the figure.*

know of butterfly matrices that have been tested for neural networks, and how they can be expressed in terms of Kronecker-sparse matrices, see Table 5.1.

Table 5.1: Examples of matrices used in neural networks, which can be expressed in terms of products of Kronecker-sparse matrices. For a matrix of the form $\mathbf{W} = \mathbf{K}_1 \dots \mathbf{K}_L$, the column "Kronecker patterns" describes the list of Kronecker sparsity patterns $\pi_\ell = (a, b, c, d)$ for each Kronecker-sparse matrix $\mathbf{K}_\ell$.

	Matrix size	Kronecker patterns
Dense	$M \times N$	$(1, M, N, 1)$
Low-rank	$M \times N$	$(1, M, r, 1), (1, r, N, 1)$
Square dyadic [Dao et al., 2019, Alizadeh-Vahid et al., 2020]	$N \times N$ with $N = 2^L$	$(2^{\ell-1}, 2, 2, 2^{L-\ell})_{\ell=1}^{L}$
Kaleidoscope [Dao et al., 2022a]	$N \times N$ with $N = 2^L$	Concatenate $(2^{\ell-1}, 2, 2, 2^{L-\ell})_{\ell=1}^{L}$ and $(2^{L-\ell}, 2, 2, 2^{\ell-1})_{\ell=1}^{L}$
Block butterfly [Chen et al., 2022]	$N \times N$ with $N = 2^L t$	$(2^{\ell-1}, 2t, 2t, 2^{L-\ell})_{\ell=1}^{L}$
Monarch [Dao et al., 2022a, Fu et al., 2023]	$M \times N$	$(1, M/p, \min(M, N)/p, p), (p, \min(M, N)/p, N/p, 1)$
Deformable butterfly [Lin et al., 2021]	$M \times N$ with $M = a_1 b_1 d_1$ and $N = a_L c_L d_L$	$(a_\ell, b_\ell, c_\ell, d_\ell)_{\ell=1}^{L}$ s.t. $a_\ell c_\ell d_\ell = a_{\ell+1} b_{\ell+1} d_{\ell+1}$.

5.1.1 Generic algorithm for Kronecker-sparse matrix multiplication

Léon and Tung, that worked a lot on the butterfly structure during their PhD, proposed a generic algorithm for Kronecker-sparse matrix multiplication, see Algorithm 5.1.1, and a mathematically *equivalent one in Algorithm 5.1.2.* While these algorithms are described below, let me emphasize early on that the core challenge to achieve efficiency in *practice* lies not in the *mathematical* formulation, but in the details of the *GPU implementation.* Algorithm 5.1.1 is a generalization of the one proposed by Dao et al. [2022a] for the specific cases $a = 1$ or $d = 1$.

Before going into the details of the algorithms, let me also explain why in all this work, the multiplication of an input $\mathbf{X} \in \mathbb{R}^{B \times N}$ with a matrix $\mathbf{M} \in \mathbb{R}^{M \times N}$ is systematically written $\mathbf{X}\mathbf{M}^T$ (multiplication on the right) instead of $\mathbf{M}\mathbf{X}^T$. *Transposition* is a memory operation that is *expensive* in terms of time and energy, and it is better to avoid it when possible. In a neural network with a layer $\mathbf{M}$, it is possible to store $\mathbf{M}^T$ instead of $\mathbf{M}$

once and for all, so that the multiplication $\mathbf{X}\mathbf{M}^T$ involves *zero new transposition* when a new input $\mathbf{X}$ is given. This is *not* the case for $\mathbf{M}\mathbf{X}^T$ where the transposition of $\mathbf{X}$ is *necessary for each new input*. This is why we choose to write the multiplication on the right.

Algorithm 5.1.1 Kronecker-sparse matrix multiplication

Require: $\pi = (a, b, c, d)$, $\mathbf{K} \in \Sigma^\pi$, $\mathbf{X} \in \mathbb{R}^{B \times N}$ ($N := acd$)
Ensure: $\mathbf{Y} = \mathbf{X}\mathbf{K}^\top \in \mathbb{R}^{B \times M}$ ($M := abd$)
1: $\mathbf{Y} \leftarrow \mathbf{0}_{B \times M}$
2: **for** $(i, j) \in [\![0, a-1]\!] \times [\![0, d-1]\!]$ **do**
3: $\quad \texttt{col} \leftarrow \left\{ i\frac{N}{a} + j + \ell d \,|\, \ell \in [\![0, c-1]\!] \right\}$
4: $\quad \texttt{row} \leftarrow \left\{ i\frac{M}{a} + j + kd \,|\, k \in [\![0, b-1]\!] \right\}$
5: $\quad \mathbf{Y}[:, \texttt{row}] \leftarrow \mathbf{X}[:, \texttt{col}]\mathbf{K}^\top[\texttt{col}, \texttt{row}]$

Algorithm 5.1.2 Mathematically equivalent formulation

Require: $\pi, \mathbf{X}, \tilde{\mathbf{K}} := \mathbf{P}^\top \mathbf{K} \mathbf{Q}^\top$ with $\mathbf{K} \in \Sigma^\pi$, $\mathbf{P} := (\mathbf{I}_a \otimes \mathbf{P}_{b,d})$, $\mathbf{Q} := (\mathbf{I}_a \otimes \mathbf{P}_{c,d})^\top$ cf. (5.1)
Ensure: $\mathbf{Y} = \mathbf{X}\mathbf{K}^\top \in \mathbb{R}^{B \times M}$
1: $\tilde{\mathbf{X}} \leftarrow \mathbf{X}\mathbf{Q}^\top$
2: $\tilde{\mathbf{Y}} \leftarrow \tilde{\mathbf{X}}\tilde{\mathbf{K}}^\top$
3: $\mathbf{Y} \leftarrow \tilde{\mathbf{Y}}\mathbf{P}^\top$

Notations. $\mathbf{X} \in \mathbb{R}^{B \times N}$ is the input matrix (batch size B, input dimension N). Σ^π is the set of matrices with Kronecker sparsity pattern $\pi = (a, b, c, d)$ (Definition 5.1.1). $\mathbf{0}_{m \times n}$ is the $m \times n$ matrix filled with zeros. For integers $a \leqslant b$, $[\![a, b]\!] := \{a, a+1, \ldots, b\}$. For a matrix $\mathbf{M}$, $\mathbf{M}[I, :]$ is the sub-matrix restricted to rows I, and $\mathbf{M}[I, J]$ is the restriction to rows I and columns J. Matrix transposition is represented by $\top$. Matrix indices start at zero.

Theoretical complexity. It is not hard to see that the theoretical complexity of Algorithm 5.1.1, defined as the number of scalar multiplications, is $Babcd$, for a batch size B and a Kronecker sparsity pattern $\pi = (a, b, c, d)$.

On Algorithm 5.1.1, and the *mathematically* equivalent Algorithm 5.1.2. When $d = 1$, the Kronecker-sparse matrix $\mathbf{K}$ is block-diagonal with a dense blocks (where "a" is the "a" in the sparsity pattern $\pi = (a, b, c, d)$ of $\mathbf{K}$), as can be seen from Figure 5.2. In this special case, Algorithm 5.1.1 loops over *each of these a blocks*. At each iteration, one block $\mathbf{K}[\texttt{row}, \texttt{col}]$ is considered, for the subsets `row` and `col` from Algorithm 5.1.1, indexed by some $i \in [\![0, a-1]\!]$. This iteration performs the multiplication between this block and the corresponding sub-matrix of $\mathbf{X}$, and accumulates the result in the output $\mathbf{Y}$. **The general case $d \geqslant 1$ is similar: the Kronecker-sparse matrix $\mathbf{K}$ is,** *up to permutation operations*, **block-diagonal with ad *dense* blocks, and Algorithm 5.1.1 loops over each of these dense blocks**, given by $\mathbf{K}[\texttt{row}, \texttt{col}]$ with `row` and `col` defined in lines 3 and 4. See Figure 5.3 for an illustration. More precisely, for any $\pi = (a, b, c, d)$, the support $\mathbf{S}_\pi$ of the Kronecker-sparse matrix $\mathbf{K}$ can be permuted as follows, to reduce to the case block-diagonal with ad *dense* blocks of size $b \times c$, corresponding to $\tilde{\pi} = (ad, b, c, 1)$:

$$\mathbf{S}_\pi = \underbrace{(\mathbf{I}_a \otimes \mathbf{P}_{b,d})}_{:=\mathbf{P}} \underbrace{(\mathbf{I}_{ad} \otimes \mathbf{1}_{b \times c})}_{=\mathbf{S}_{\tilde{\pi}}} \underbrace{(\mathbf{I}_a \otimes \mathbf{P}_{c,d})^\top}_{:=\mathbf{Q}} = \mathbf{P}\mathbf{S}_{\tilde{\pi}}\mathbf{Q}, \tag{5.1}$$

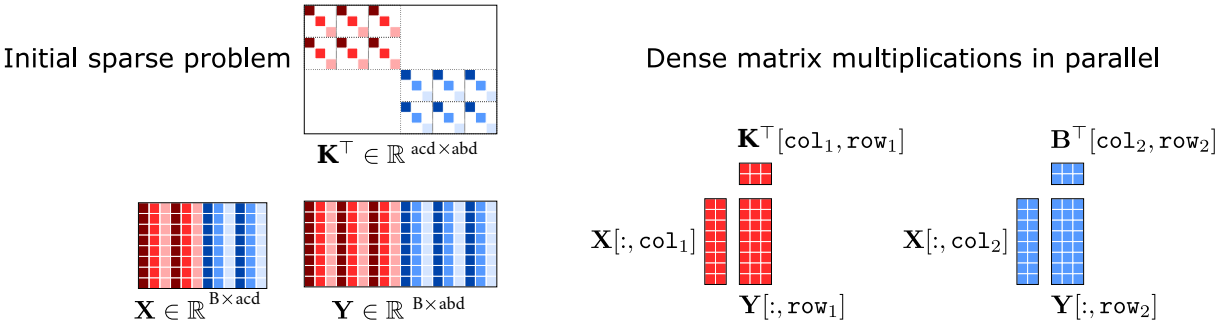


Figure 5.3: Illustration of Algorithm 5.1.1 for sparsity pattern $\pi = (2, 3, 2, 3)$ and batch size $B = 8$. The subsets of rows and columns $(\texttt{row}_1, \texttt{col}_1)$ are associated with the values $(i, j) = (0, 1)$ in the "for" loop of Algorithm 5.1.1, whereas $(\texttt{row}_2, \texttt{col}_2)$ are associated with $(i, j) = (1, 1)$. *Courtesy of Léon for the figure.*

where $\mathbf{P}_{p,q}$ for two integers p, q is the so-called (p, q) *perfect shuffle* permutation matrix of size $pq \times pq$ [Van Loan, 2000] (see Appendix D.3 for details, *courtesy of Léon and Tung*). Therefore, for any $\mathbf{K} \in \Sigma^\pi$, we have $\mathbf{K} = \mathbf{P}\tilde{\mathbf{K}}\mathbf{Q}$ with $\tilde{\mathbf{K}} := \mathbf{P}^\top\mathbf{K}\mathbf{Q}^\top \in \Sigma^{\tilde{\pi}}$ that is block-diagonal with *ad dense* blocks of size $b \times c$. While Algorithm 5.1.1 loops over each of the ad dense sub-matrices $\mathbf{K}[\texttt{row}, \texttt{col}]$ of $\mathbf{K}$ directly, many concrete implementations of Algorithm 5.1.1 presented below are based on the equivalent formulation that directly manipulates $\tilde{K}$, the permuted version of $\mathbf{K}$ with dense diagonal blocks. A *mathematical* formulation that is more faithful to these implementations based on $\tilde{K}$ is given in Algorithm 5.1.2. Since $\tilde{\mathbf{K}}^\top$ is directly manipulated instead of $\mathbf{K}^\top$, it is now the entry of the algorithm, and Algorithm 5.1.2 consists in three steps to compute $\mathbf{Y} = \mathbf{X}\mathbf{K}^\top = \mathbf{X}\mathbf{Q}^\top\tilde{\mathbf{K}}^\top\mathbf{P}^\top$: permute the inputs with $\mathbf{Q}$, multiply with $\tilde{\mathbf{K}}^\top$, and repermute with $\mathbf{P}$.

5.1.2 Baseline GPU implementations

The code is available at `github.com/PascalCarrivain/ksmm`. We now describe concrete baseline GPU implementations of Algorithms 5.1.1 and 5.1.2. The relevant lines of code are given in Appendix D.4.1 and the full code is available at `github.com/PascalCarrivain/ksmm`.

`bmm` and `bsr` implementations. We consider the implementation of Algorithm 5.1.2 from Dao et al. [2022a], that we choose to call `bmm` since it mainly relies on batched GEMM NVIDIA routines called through a function of the `PyTorch` library named `torch.bmm`. The original `bmm` implementation from Dao et al. [2022a] *only works for a pattern* $\pi = (a, b, c, d)$ *satisfying* $a = 1$ *or* $d = 1$. *Léon and Tung* are the ones who *extended* `bmm` *to the general case.*

I also developed a new implementation of Algorithm 5.1.2 that we call `bsr`, since it mainly relies on the `PyTorch` block-sparse library called BSR. More details on how `bmm` and `bsr` implement Algorithm 5.1.2 are given in Table 5.2.

`einsum` implementation. *Léon and me* developed another baseline, this time implementing Algorithm 5.1.1. We call it `einsum` as it relies on tensor contractions, mainly through the `einsum` routine of the `Python einops` library [Rogozhnikov, 2021]. It stores

	`bmm`	`bsr`
Storage format for $\tilde{\mathbf{K}}$	3D-tensor of shape (ad, b, c)	2D-tensor of shape (abd, acd) stored in BSR[7] format
Line 1 of Algorithm 5.1.2	`torch.reshape`	
Line 2 of Algorithm 5.1.2	`torch.bmm`	`torch.nn.functional.linear`
Line 3 of Algorithm 5.1.2	`torch.reshape`	

Table 5.2: Differences in the implementation of Algorithm 5.1.2 between `bmm` and `bsr`.

the nonzero entries of $\mathbf{K} \in \Sigma^{\pi}$ using a 4D-tensor `B_einsum` of shape (a, b, c, d), in such a way that the slice `B_einsum`$[i, :, :, j]$ for $(i, j) \in [\![0, a-1]\!] \times [\![0, d-1]\!]$ stores the entries of $\mathbf{K}$[`row`, `col`] where `row`, `col` are defined in lines 3 and 4 of Algorithm 5.1.1. The batched matrix multiplication operations at line 5 are then implemented using Einstein summation between this 4D-tensor and a reshaped input tensor.

The above implementations (`bmm`, `bsr`, `einsum`) can be compared to the two following generic implementations (`dense` and `sparse`) that ignore the Kronecker sparsity.

`dense` implementation. This ignores the sparsity of $\mathbf{K}$, by storing *all* its entries, including *zeros*, in a tensor of shape (M, N). The multiplication is done with `torch.nn.functional.linear`, the default `PyTorch` implementation for linear layers.

`sparse` implementation. This exploits the sparsity of $\mathbf{K}$ but *not its structure* (recall that the sparsity pattern is not arbitrary, but structured as Kronecker products, see Definition 5.1.1). The nonzero entries of the factor $\mathbf{K}$ are saved in a tensor stored in the Compressed Sparse Row (CSR) format, and the matrix multiplication is done with `torch.nn.functional.linear`.

The two implementations `dense` and `sparse` are ones made available by `PyTorch`, and we use them without further modification.

Batch-size-first *vs.* batch-size-last. The entries of the input $\mathbf{X} \in \mathbb{R}^{B \times N}$ can be stored either in a `PyTorch` tensor `X_bsf` of shape (B, N), or in a `PyTorch` tensor `X_bsl` of shape (N, B). This impacts the memory layout of $\mathbf{X}$ and can drastically impact the performance of the implementation depending on the way $\mathbf{X}$ is accessed. Indeed, `PyTorch` makes the convention of storing *all* tensors in the *row-major* convention, meaning that the *entries of each row are stored contiguously in memory*. To understand the difference between `X_bsf` and `X_bsl`, consider the example of $\mathbf{X} = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$.

In the *batch-size-first* layout, the tensor `X_bsf` will be `X_bsf = torch.tensor([[1, 2], [3, 4]])`, whereas in the *batch-size-last* layout, the tensor `X_bsl` corresponds to the transpose of $\mathbf{X}$ since the batch-size is placed at last, so it will be `X_bsl = torch.tensor([[1, 3], [2, 4]])`. Since `PyTorch` is row-major, in the case of *batch-size-first*, the entries are stored in memory in the order:

$$1, 2, 3, 4,$$

whereas in the case of *batch-size-last*, the entries are stored in memory in the order:

$$1, 3, 2, 4.$$

This is either storing the *rows* of $\mathbf{X}$ contiguously in memory (*batch-size-first*), or its columns (*batch-size-last*).

When using a Kronecker-sparse matrix $\mathbf{K}$ to replace a dense matrix in a neural network, we have the *freedom* to store $\mathbf{K}$ in the format that is *most efficient for the implementation*. However, the input $\mathbf{X}$ is what is received from the previous layer, and we have *no control over its format*. Similarly, the output $\mathbf{Y}$ is what is sent to the next layer, and we should send it in the format *expected* by the next layer. Because of `PyTorch`'s row-major convention, the *standard* is to receive and send tensors in the *batch-size-first* format. However, all the implementations above can also be implemented assuming the input and output tensors to be in the *batch-size-last* format. While the main point of this chapter is to compare the implementations regardless of the memory format, we will also study the effect of the choice between *batch-size-first* and *batch-size-last* for the input and output tensors.

Finally, let me mention that we choose to call these two different memory layouts by *batch-size-first* and *batch-size-last*, by analogy with the recent `PyTorch` optimization "channels last" that moves the channel dimension to the last position for convolutional layers.

5.2 Memory accesses in baseline implementations

The implementations `bmm` and `bsr` explicitly perform the $\mathbf{P}$ and $\mathbf{Q}$ permutation operations from Algorithm 5.1.2, before calling high-performance multiplication routines for the multiplication with $\tilde{\mathbf{K}}$ (line 2 in Algorithm 5.1.2). This section assesses for the first time the cost of these *permutations* in practice, and is the fruit of discussions between *Léon, Pascal and me*.

Importance of data transfers. GPU memory management plays a critical role in optimizing performance. Memory in a GPU is organized hierarchically, with global memory being the largest and slowest, followed by shared memory, and finally registers, which are the smallest and fastest [NVIDIA, 2024, Sec. 2.3]. By default, data resides in the global memory of the GPU. Each thread of the GPU runs a kernel that reads data from global memory into registers, performs register-level computations, and writes the results back to global memory. Therefore, when operations are bottlenecked by memory accesses, it is critical to minimize data transfers between global memory, shared memory, and registers to obtain an efficient GPU implementation [NVIDIA, 2024, Sec. 5.3].

Data transfers in baseline implementations. In this section, we argue that the baseline `bmm`, `bsr` and `einsum` implementations for Kronecker-sparse matrix multiplication require performing *several passes between global memory and registers*, which can account for a *large proportion of the total runtime* in practice. This suggests that there is room for improvement in the memory accesses of these implementations.

Let us focus on `bmm`, as we will find it to be faster than the other baseline implementations. The data flow of `bmm` is illustrated in Figure 5.4. There is one pass between the global memory and the registers to perform the permutation with $\mathbf{P}$ (line 3

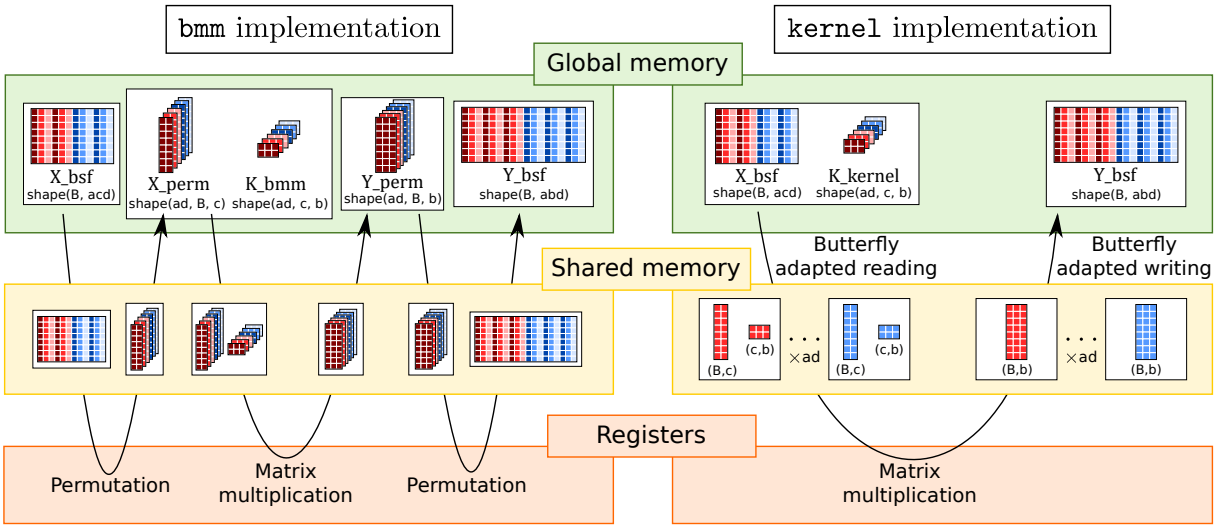


Figure 5.4: Data flow between the different levels of GPU memory for the `bmm` implementation (Section 5.1.2) from Dao et al. [2022a] and the new `kernel` (Section 5.3). *Made in collaboration with Léon.*

in Algorithm 5.1.2), one for the multiplication with $\tilde{\mathbf{K}}$ (line 2), and another one for the permutation with $\mathbf{Q}$ (line 1).

The fact that `bmm` has three passes between the global memory and the registers has nothing specific to `bmm`. We argue that this has to be the case for *any implementation* that implements the multiplications $\mathbf{X}[:, \texttt{col}]\mathbf{K}^\top[\texttt{col}, \texttt{row}]$ (line 5 in Algorithm 5.1.1, or the *mathematically* equivalent line 2 in Algorithm 5.1.2) by calling high-performance libraries *from common* ***`Python`*** *interfaces* (`PyTorch`, `Tensorflow`).

Let me explain why we expect this to be the case. When calling a multiplication routine *from* ***`Python`*** to perform a multiplication $\mathbf{AB}$, this generally requires having the entries of both matrices $\mathbf{A}$ and $\mathbf{B}$ stored *contiguously in the global memory of the GPU*. Therefore, in the special case where we want to perform the multiplication $\mathbf{X}[:, \texttt{col}]\mathbf{K}^\top[\texttt{col}, \texttt{row}]$, both $\mathbf{X}[:, \texttt{col}]$ and $\mathbf{K}^\top[\texttt{col}, \texttt{row}]$ have to be stored *contiguously in global memory*. Unfortunately, *this is not the case for the entries of* $\mathbf{X}[:, \texttt{col}]$ when $\mathbf{X}$ is directly received from the previous layer of a neural network. Indeed, `PyTorch`'s convention would store $\mathbf{X} \in \mathbb{R}^{B \times N}$ in row-major as a 2D-tensor of shape either (B, N) or (N, B) (depending if we choose to place the batch in first or last position). In these two cases, the sub-matrix $\mathbf{X}[:, \texttt{col}]$ is *not* stored contiguously in memory as soon as $d > 1$ (the "d" of the Kronecker sparsity pattern $\pi = (a, b, c, d)$). Indeed, the indices in `col` are equally spaced by d (see line 4 in Algorithm 5.1.2).

Therefore, *a first pass between the global memory and the registers is required to rewrite the entries of* $\mathbf{X}[:, \texttt{col}]$ *contiguously in global memory*, see Figure 5.4. In the `bmm` implementation, this first pass corresponds to the application of the permutation matrix $\mathbf{Q}$ from Algorithm 5.1.2, to the input $\mathbf{X}$, as illustrated on the left of Figure 5.4. In what follows, we call **memory rewriting** each operation that consists of moving one entry of a tensor, to rewrite it somewhere else in memory.

Similarly, the result of the multiplication $\mathbf{Y}[:, \texttt{row}] = \mathbf{X}[:, \texttt{col}]\mathbf{K}^\top[\texttt{col}, \texttt{row}]$, as returned by an efficient routine called from `Python`, will in general be stored contiguously in *global memory* for each pair $(\texttt{row}, \texttt{col})$. But indices in `row` are equally spaced by d: *this*

requires another pass to rewrite them equally spaced by d in the *final* output 2D-tensor storing all the entries of $\mathbf{Y}$ contiguously in memory. In the `bmm` implementation, this pass corresponds to the application of the permutation matrix $\mathbf{P}$ from Algorithm 5.1.2, as illustrated by the last pass of `bmm` in Figure 5.4. With the pass implied by the actual multiplication routine, this results in *three passes* between the global memory and the registers (Figure 5.4) for any implementation that performs all the multiplications $\mathbf{X}[:, \texttt{col}]\mathbf{K}^\top[\texttt{col}, \texttt{row}]$ by calling high-performance libraries from *common* ***Python*** *interfaces*.

Estimated time for memory rewritings in `bmm`. We benchmark the relative time spent on memory rewritings in `bmm`, which is, as we will find out later (Section 5.4), the fastest of the baseline implementations. We find that **the memory rewritings can take up to** 45% **of the total runtime**[8]. This can be seen by looking at the y-axis in Figure 5.5 (see Appendix D.2.2 for details on the experiments). Therefore, *it is crucial to optimize the data transfers between the different levels of GPU memory* to improve current implementations. To the best of our knowledge, this is the first time that the cost of memory rewritings in baseline implementations such as `bmm` is discussed in the literature.

A proxy for the time spent on memory rewritings in the baseline implementations. As we can see from Figure 5.5, the time spent on memory rewritings (y-axis) increases with the ratio $(b+c)/bc$ (x-axis). We now give a *theoretical explanation* for this observation, *and credit goes to Léon for suggesting this.*

Consider input and output dimensions N and M, and a batch size B. We saw above that for implementations that perform the multiplications $\mathbf{X}[:, \texttt{col}]\mathbf{K}^\top[\texttt{col}, \texttt{row}]$ by calling efficient routines from ***Python*** *interfaces*, such as `bmm`, all the entries of $\mathbf{X}[:, \texttt{col}]$ and $\mathbf{Y}[:, \texttt{row}]$ must be rewritten in memory *in order to store these matrices contiguously.* Doing that for all pairs $(\texttt{row}, \texttt{col})$ (Algorithm 5.1.1), this results in moving a total of $BN + BM$ different entries: each entry of the input and output tensors is moved exactly once. Let us compare this to the total number of scalar multiplications performed when computing the products $\mathbf{Y}[:, \texttt{row}] = \mathbf{X}[:, \texttt{col}]\mathbf{K}^\top[\texttt{col}, \texttt{row}]$ for all $(\texttt{row}, \texttt{col})$, which is equal to $B \times \#\text{nnz}$ (the batch-size times the number of nonzero in $\mathbf{K}$). For a Kronecker-sparse matrix with sparsity pattern $\pi = (a, b, c, d)$, we have $N = acd$, $M = abd$ and $\#\text{nnz} = abcd$. Therefore, the ratio $(b + c)/bc$ is precisely equal to

$$\frac{\texttt{number of memory rewritings}}{\texttt{number of scalar multiplications}} = \frac{BN + BM}{B \times \#\text{nnz}} = \frac{b+c}{bc}. \tag{5.2}$$

This explains why the ratio $(b + c)/bc$ is a good proxy for the relative time spent on memory rewritings in practice, as can be seen from Figure 5.5. Let me anticipate a little bit here and explain why this is particularly interesting to have such a proxy. Our new kernel will reduce the cost of memory rewritings, so the Kronecker sparsity patterns with a large value of $(b + c)/(bc)$ will benefit *the most* from our new implementation. An important consequence of this provides *a heuristic to identify efficient Kronecker patterns* and therefore to *help designing efficient Kronecker-sparse neural networks*, as I will discuss

[8] Regardless of the memory layout convention, *batch-size-first* or *batch-size-last*.

in the perspectives (Chapter 7). For instance, this ratio can be used to *identify*, among all the Kronecker sparsity patterns with a *given number of nonzeros*, the ones for which our new implementation is expected to be the most efficient.

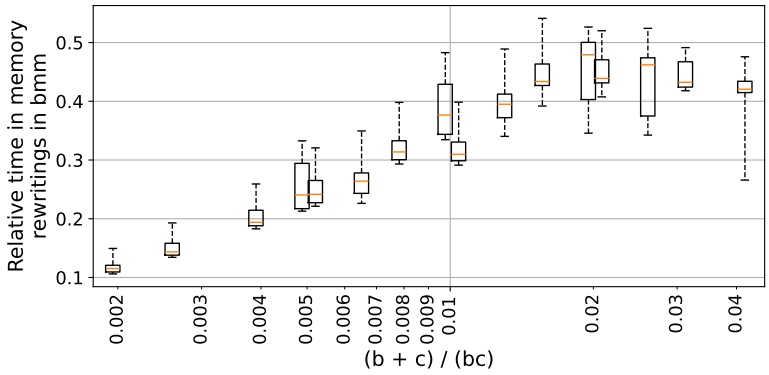


Figure 5.5: Estimated relative time spent on memory rewritings in `bmm` for the multiplication with $\mathbf{K} \in \Sigma^{\pi}$, for several $\pi = (a, b, c, d)$. We regroup patterns by their value of $(b + c)/(bc)$, and plot a boxplot to summarize the corresponding measurements.

5.3 New CUDA kernel with reduced memory transfers

We saw in the previous section that the baseline implementations `bmm`, `bsr` and `einsum` require several passes between the global memory and the registers to perform the multiplication with a Kronecker-sparse matrix $\mathbf{K}$, and that this can account for up to 45% of the total runtime. We also explained that this number of passes is expected to be inherent to any implementation that performs all the multiplications $\mathbf{X}[:, \texttt{col}]\mathbf{K}^{\top}[\texttt{col}, \texttt{row}]$ by calling high-performance libraries from *common* ***`Python`*** *interfaces*.

For this reason, we now consider going at a lower level than `Python` to better manage the memory accesses. This was the occasion for *Pascal and me* to discover CUDA, the programming language for NVIDIA GPUs, and to jointly develop a new kernel that performs the multiplication with a Kronecker-sparse matrix $\mathbf{K}$ with *reduced memory transfers*. This new kernel does only *one pass* between the global memory and the registers to perform the multiplication with $\mathbf{K}$, against *three* for the fastest baseline implementation `bmm`, as illustrated in Figure 5.4. Doing only one pass is made possible by tailoring the memory accesses to the known sparsity structure of $\mathbf{K}$. We directly access the sub-matrices $\mathbf{X}[:, \texttt{col}]$ and $\mathbf{K}^{\top}[\texttt{col}, \texttt{row}]$ *even if all their entries are not stored contiguously*, in a way that ensures that most of the memory accesses yet remain contiguous and efficient.

The idea is basically that as long as *enough* entries are stored contiguously, the associated memory accesses can be made efficient. For instance, consider two rows of a matrix. While each *individual* row is stored contiguously in memory, the memory accesses can be made efficient, even if the two rows are not stored *consecutively* in memory. The only moment where we have to pay a price is if we want to access entries of the two rows at the *same time*, and if these rows are *far from each other* in memory. *This will be fine as long as this does not happen too often.*

The new kernel is the main contribution of this chapter. We will empirically show that it is faster than the baselines in the next section (Section 5.4), and that it is particularly efficient for Kronecker sparsity patterns with a large value of $(b+c)/(bc)$, as anticipated in the previous section.

Implementation. The kernel implements Algorithm 5.1.1. It performs the multiplications $\mathbf{X}[:, \texttt{col}]\mathbf{K}^\top[\texttt{col}, \texttt{row}]$ in parallel for all the pairs $(\texttt{row}, \texttt{col})$, as defined in Algorithm 5.1.1. To perform one of these multiplication, the kernel starts by reading into *global memory* the entries in $\mathbf{X}[:, \texttt{col}]$ and $\mathbf{K}^\top[\texttt{col}, \texttt{row}]$, and load them into *shared memory*. Then, it performs the multiplication, which involves passing the data from shared memory to registers, performing the multiplication, and storing the result in shared memory. The kernel then reads the result from shared memory and accumulates it in the output stored in global memory. This is illustrated in Figure 5.4.

The main novelty of the kernel is the reading and writing phase. Their main feature is to *not* require to read or write entries in *contiguous* ways. Let me detail the case of the reading phase, as this is similar for the writing phase. For the reading phase, the entries of $\mathbf{X}[:, \texttt{col}]$ are *not* required to be *contiguous* in global memory[9]. The kernel *is aware* of the sparsity pattern (a, b, c, d), so it can *explicitly* compute the set of indices in `col` and directly access them in global memory. These indices correspond to columns equally spaced by d (the "d" of the Kronecker sparsity pattern $\pi = (a, b, c, d)$, see Figure 5.3), so the whole sub-matrix is not contiguous as soon as $d > 1$. Therefore, some consecutive memory accesses have to be non-contiguous (when changing the column being accessed). Yet, if the entries of a same column are stored contiguously in memory, *the memory accesses* can be made efficient. The kernel is therefore particularly *efficient* when the *columns* are stored *contiguously*, corresponding to the *batch-size-last* memory layout (recall that *batch-size-first* has been defined at the end of Section 5.1.2).

However, the kernel is expected to be *less* efficient when these are the *rows* that are stored contiguously, corresponding to the *batch-size-first* memory layout[10]. Indeed, in this case it would be efficient to make consecutive accesses to the entries in a single row, but the entries of $\mathbf{X}[:, \texttt{col}]$ in a *same* row are equally spaced by d. Therefore, we expect the kernel to be particularly efficient for the *batch-size-last* memory layout, and less efficient for the *batch-size-first* memory layout. This is confirmed by the benchmarks in the next section (Section 5.4).

After the reading phase, all the entries are in shared memory. The `kernel` then implements the classic tile matrix multiplication algorithm[11] [Li et al., 2019, Boehm, 2022, NVIDIA, 2023a,b, 2024] to compute each product $\mathbf{X}[:, \texttt{col}]\mathbf{K}^\top[\texttt{col}, \texttt{row}]$ for each subsets `row`, `col` in Algorithm 5.1.1. Once these products are computed, the result is stored contiguously in *shared memory*, and the kernel accumulates this result in the output stored in *global memory*. For this, the kernel again has a custom writing phase from shared to global memory, rewriting the results that are stored *contiguously* in shared memory, to *non-contiguous* locations in global memory, because each sub-matrix $\mathbf{Y}[:, \texttt{row}]$ corresponds

[9] Contrasting with efficient `Python` multiplication routines that would require them to be contiguous.

[10] Unfortunately for us, *batch-size-first* is the default convention of `PyTorch`, which seems to be mainly motivated by *historical* reasons. Despite this unfavorable convention for the kernel, we will see that it still performs well in *batch-size-first*.

[11] Classical optimizations are used, as detailed in Appendix D.4.2.

to columns of the output $\mathbf{Y}$ (see Figure 5.3) that are not consecutive as soon as $d > 1$. This writing phase is also expected to be more efficient in the *batch-size-last* memory layout, ensuring the entries of a same column of $\mathbf{Y}[:, \texttt{row}]$, and thus the write operations, to be contiguous in memory.

Comparison with other baseline implementations. This new kernel has fewer global memory accesses compared to the baselines `einsum`, `bsr` and `bmm` (Section 5.1.2), because it only reads each coefficient of $\mathbf{X}$ and $\mathbf{K}$ once and writes the result of the multiplication $\mathbf{Y}$ once, while those baseline implementations read $\mathbf{X}$ and $\mathbf{Y}$ twice and rewrite them once (to permute them). In a sense, the kernel has fused the multiplication and the permutation operations of the baseline implementations, avoiding for several passes between the global memory and the registers.

Compared to the `dense` implementation, it also has fewer global memory accesses, since the `dense` implementation additionally reads the *zero* entries of $\mathbf{K}$ and the corresponding coefficients of $\mathbf{X}$, while our kernel does not. Compared to the generic `sparse` implementation, we have the same number of memory accesses, but the `sparse` implementation is agnostic to the sparsity structure of $\mathbf{K}$, so it is expected to be less efficient since the memory accesses are not tailored to the known location of the nonzero entries.

5.4 Benchmarking the multiplication with a Kronecker-sparse matrix

We now benchmark the different implementations described so far for Kronecker-sparse matrix multiplication. In particular, we validate numerically the benefits of the new `kernel` implementation, with improved memory transfers, compared to the baselines `einsum`, `bsr` and `bmm`.

Protocol. The benchmark is run in *float-precision* on a subset of 600 sparsity patterns $\pi = (a, b, c, d)$ in $\alpha \times \beta \times \beta \times \alpha$, with $\alpha := \{1, 2, 3, 4, 6, 8, 12, 16, 24, 32, 48, 64, 96, 128\}$, $\beta := \{48, 64, 96, 128, 192, 256, 384, 512, 768, 1024\}$, such that $b = c$ or $b = 4c$ or $c = 4b$. These patterns correspond to dimensions of Kronecker-sparse matrices $\mathbf{K} \in \mathbb{R}^{M \times N}$ with $(M, N) = (abd, acd)$ in the linear layers of Transformers (up projection for $b = 4c$, down projection for $c = 4b$, fully-connected layers for $b = c$) and more generally in any neural network. We choose as batch size $B = 128 \times 196 = 25088$, a standard effective batch size for fully-connected layers in Vision Transformers (ViTs) [Dosovitskiy et al., 2021], corresponding to a number of sequences per batch equal to 128, multiplied by a number of tokens per sequence equal to 196. *Credit goes to Léon for suggesting this protocol, that we set up with Pascal.*

Further details are given in Appendix D.2.1.

Implementations specialized to Kronecker sparsity improves over generic implementations. The first line of Table 5.3 shows that at least one of the implementations specialized to the Kronecker structure among `kernel`, `bmm`, `einsum` and `bsr` improves over the generic `dense` and `sparse` implementation, which do not take into account the

Table 5.3: Percentage out of 600 patterns (a, b, c, d) where `algo1` is *faster* than the `algo2` (denoted by time(`algo1`) < time(`algo2`)), and the median acceleration factor in such cases (that is, the median ratio $\frac{\text{time of algo2}}{\text{time of algo1}}$). For each implementation, we take the minimum time between the *batch-size-first* and the *batch-size-last* memory layout.

min time $\begin{pmatrix}\texttt{kernel}\\ \texttt{bmm}\\ \texttt{einsum}\\ \texttt{bsr}\end{pmatrix}$ < min time $\begin{pmatrix}\texttt{dense}\\ \texttt{sparse}\end{pmatrix}$	time(`bmm`) < min time $\begin{pmatrix}\texttt{einsum}\\ \texttt{bsr}\\ \texttt{dense}\\ \texttt{sparse}\end{pmatrix}$	time(`kernel`) < min time $\begin{pmatrix}\texttt{bmm}\\ \texttt{einsum}\\ \texttt{bsr}\\ \texttt{dense}\\ \texttt{sparse}\end{pmatrix}$
99.67% (×6.57)	92.66% (×1.37)	88.10% (×1.39)

Kronecker sparsity. The speedup increases with the matrix size $M \times N$ of the Kronecker-sparse matrix $\mathbf{K}$, see Figure 5.6.

The baseline `bmm` is faster than the other baselines `einsum` and `bsr`. This is shown in the second line of Table 5.3, where the `bmm` implementation improves over min(`einsum`, `bsr`) in 93% of the tested cases. The speedup increases with the matrix size $M \times N$ of the Kronecker-sparse matrix $\mathbf{K}$, see Figure 5.7. Therefore, when comparing the new `kernel` implementation to other baselines, we will mainly focus on the comparison between `bmm` and `kernel`.

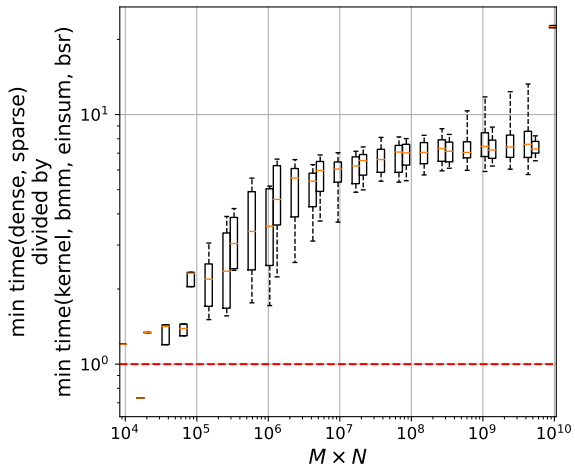


Figure 5.6: Speed-up factor of min time(`kernel`, `bmm`, `bsr`, `einsum`) compared to min time(`dense`, `sparse`) as a function of the matrix size $M \times N$.

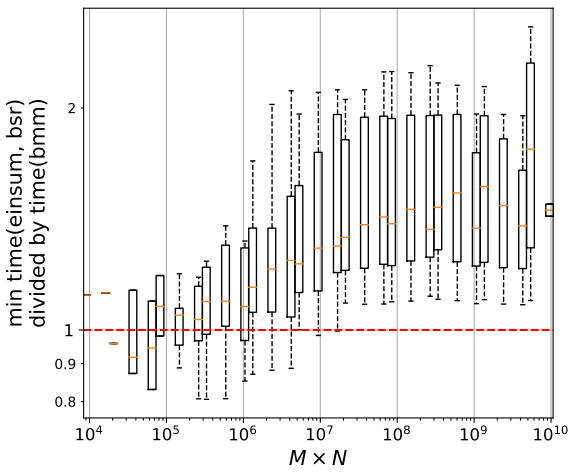


Figure 5.7: Speed-up factor of time(`bmm`) compared to min time(`einsum`, `bsr`) as a function of the matrix size $M \times N$.

The new `kernel` implementation is faster than existing baselines. The third row of Table 5.3 shows that `kernel` is faster than all other baselines in 88% of the tested patterns. This empirically validates the benefits of the reduced memory transfer in the `kernel` implementation. In the following, we provide further details on the influence of the memory layout (*batch-size-first* vs. *batch-size-last*) on this improvement. Additionally, we analyze the patterns $\pi = (a, b, c, d)$ for which the `kernel` outperforms baseline implementations.

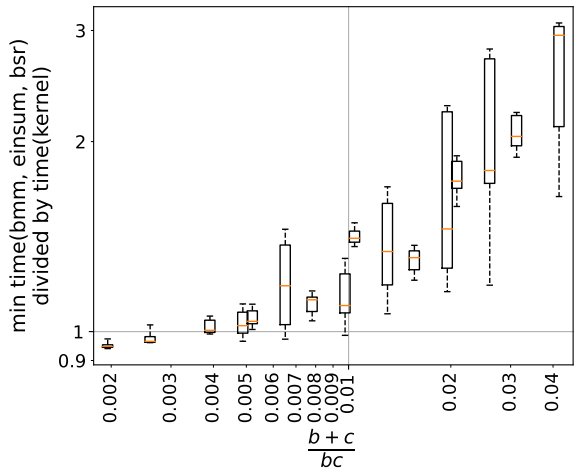


Figure 5.8: Time speedup factor of `kernel` compared to min(`bmm`, `einsum`, `bsr`). For each implementation, we take the minimum time between the *batch-size-first* and *batch-size-last* memory layouts. We regroup the patterns by their value of $(b + c)/(bc)$, and plot a boxplot to summarize the corresponding measurements.

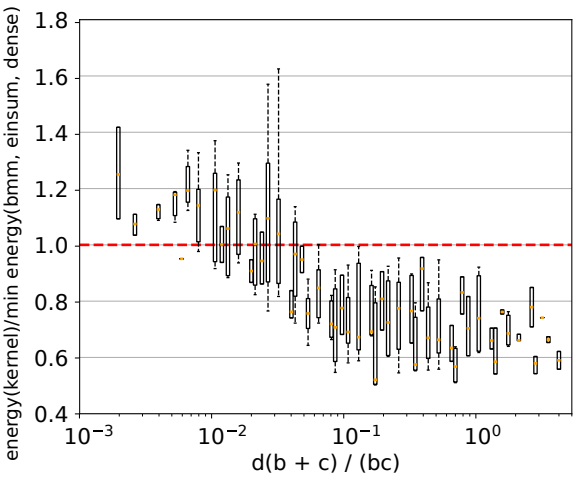


Figure 5.9: Energy consumed by `kernel` compared to the minimum consumed by `bmm`, `einsum` and `bsr`. For each implementation, we take the minimum energy consumed between the *batch-size-first* and *batch-size-last* memory layouts. We regroup patterns by their value of $d(b + c)/(bc)$.

Impact of the memory layout. For baseline implementations, switching to *batch-size-last* yields a high systematic speedup for `sparse`, high variability in the speedup of `bsr`, and essentially no impact to negative impact for the other methods, see Figure 5.10. The important part is that it has no impact on `bmm`, and since `bmm` is the fastest baseline implementation (Table 5.3), switching to *batch-size-last* has no impact on the best of the baseline implementations. However, it yields a systematic speedup (about $\times 2$) for the `kernel` implementation. This acceleration is expected, since the *batch-size-last* memory layout allows for more efficient memory accesses in the `kernel` implementation, as detailed in Section 5.3. Table 5.4 shows the percentage of patterns for which the `kernel` implementation improves over all baseline implementations, either in the *batch-size-first* or the *batch-size-last* memory layout. When restricting all implementations to the *batch-size-first* layout, the `kernel` still improves on 20% of the tested patterns despite non-contiguous memory accesses (Section 5.3).

Table 5.4: Percentage out of 600 patterns (a, b, c, d) where `algo1` is *faster* than the `algo2` (denoted by time(`algo1`) $<$ time(`algo2`)), and the median acceleration factor in such cases (that is, the median ratio $\frac{\text{time of } \texttt{algo2}}{\text{time of } \texttt{algo1}}$).

	time(`kernel`) $<$ min time(`bmm`, `einsum`, `bsr`, `dense`, `sparse`)
Batch-size-first	20.0% ($\times 1.28$)
Batch-size-last	88.1% ($\times 1.39$)

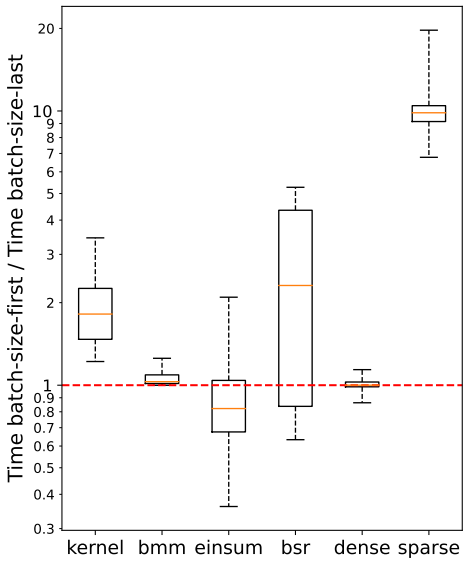


Figure 5.10: Boxplots of the ratio $\frac{\text{time of } \textit{batch-size-first}}{\text{time of } \textit{batch-size-last}}$.

Analyzing the cases where `kernel` outperforms baselines. As seen in Section 5.3, the `kernel` has an improved memory access design compared to the rest of the baselines. Figure 5.8 confirms this experimentally: **the `kernel` implementation becomes increasingly time-efficient compared to the baseline implementations as the relative number of memory accesses increases**, i.e. when the following ratio increases (introduced in (5.2))

$$\frac{\texttt{number of memory rewritings}}{\texttt{number of scalar multiplications}} = (b+c)/(bc).$$

The `kernel` improves on energy efficiency. Overall, the median energy reduction factor is $\times 0.85$, and the new kernel improves the energy consumption in 72% of the tested cases. The energy measurements are done with the software `pyJoules`. *I have assisted with the design of these experiments, and Pascal handled their execution.* More details about the measurements are in Appendix D.2.1. It demonstrates that **the `kernel` not only achieves higher time efficiency but also reduces energy consumption** compared to other baselines. This twofold advantage makes the `kernel` an effective solution for improving both performance and sustainability.

A proxy for the energy spent on memory rewritings in the baseline implementations. Figure 5.9 shows further that the energy efficiency of the `kernel` increases with the value of $d(b+c)/(bc)$. We now give a theoretical explanation for this. For a sparsity pattern $\pi = (a, b, c, d)$, we already discussed that the ratio $(b+c)/bc$, corresponding to the relative number of memory rewritings (Equation (5.2)), is a good proxy of the relative *time* spent on memory rewritings in practice. To transform it in a good proxy for the relative *energy* spent on memory rewritings, I proposed to multiply this ratio by d. The reason for this is that the integer d corresponds to the distance between the columns to be rewritten contiguously (i.e., the columns in `col` from Algorithm 5.1.1). Therefore, I expect the energy spent by `bmm` on memory rewritings to increase with d. This is confirmed

by the results in Figure 5.9. As I discuss in the perspectives (Chapter 7), I expect this to be a useful heuristic to design Kronecker-sparse neural networks that are energy-efficient.

5.5 Broader implications for neural networks: accelerating inference

We already saw in the introduction that the inference of neural networks is represents more than 90% of the cost of machine learning at scale [HPCwire, 2019, Barr, 2019]. We now investigate whether replacing fully-connected layers by products of Kronecker-sparse matrices accelerates the inference. While the same could also apply to other architectures, we will consider so-called Vision Transformers (ViTs) [Dosovitskiy et al., 2021]. We find that the computational cost of fully-connected layers is significant in such architectures: depending on the size of the ViT, from 30% to 60% of the total time in a forward pass is spent in fully-connected layers (see Appendix D.2.3 for details), *courtesy of Léon for these experiments.*

Protocol. We benchmark in *float-precision* various components of a ViT-S/16 architecture: a linear layer with bias, an MLP with non-linear activation and/or normalization layers, a multi-head attention module, etc. As in Dao et al. [2022a], we replace by a product of *two* Kronecker-sparse matrices the weight matrices of linear layers in feed-forward network modules, and the projection matrices for keys, queries and values in multi-head attention modules. Each product of two Kronecker-sparse matrices is of the form $\mathbf{K}_1\mathbf{K}_2$ (Definition 5.1.1) with respective sparsity patterns π_1, π_2 given by: $(1, 192, 48, 2), (2, 48, 192, 1)$ for the size $N \times N$, $(1, 768, 192, 2), (6, 64, 64, 1)$ for the size $4N \times N$, $(1, 768, 192, 2), (6, 64, 64, 1)$ for the size $4N \times N$. We focus on *batch-size-first* as it is the default convention in `PyTorch`[12]. *Credit goes to Léon for suggesting this protocol, that he set up with Pascal.*

Results. We denote by time(`fully-connected`) the inference time with dense matrices (and therefore, with the standard `PyTorch` implementation). Table 5.5 shows that time(`kernel`) < time(`bmm`) < time(`fully-connected`) over all the different submodules. **This concretely shows that using Kronecker-sparse matrices and the `kernel` implementation accelerates the inference of standard neural networks.**

Table 5.5: Acceleration of submodules of a ViT-S/16 using Kronecker-sparse matrices.

	$\frac{\text{time(bmm)}}{\text{time(fully-connected)}}$	$\frac{\text{time(kernel)}}{\text{time(fully-connected)}}$
Linear $N \times N$	0.82	**0.50**
Feed-forward network	0.91	**0.77**
Multi-head attention	0.87	**0.79**
Block	0.90	**0.78**
Kronecker-sparse ViT-S/16	0.89	**0.78**

[12]The insertion of Kronecker-sparse matrices in the *batch-size-last* memory layout would *a priori* require a careful implementation of the rest of the operations in *batch-size-last*, that are for now optimized in *batch-size-first* in `PyTorch`.

5.6 Conclusion

The *practical gain in time and energy* using Kronecker-sparse matrices was not clear before this thesis as there were too few reported results on the topic. This thesis aimed at closing this gap by extensively benchmarking existing GPU implementations of Kronecker-sparse matrix multiplication. Thanks to that, we identified that these implementations can spend up to 50% of their time on memory rewriting operations. This led us to propose a new CUDA kernel for Kronecker-sparse matrix multiplication that fuses some operations to reduce the memory transfers between the different levels of GPU memory. **In *float-precision*, the new kernel not only accelerates Kronecker-sparse matrix multiplication, but also decreases the energy consumption.** Moreover, we provide a simple heuristic to choose Kronecker sparsity patterns (a, b, c, d) that are particularly efficient for this implementation.

Perspectives. The heuristics we provided to decide which Kronecker sparsity patterns are time-efficient and energy-efficient pave the way to new research directions to design efficient Kronecker-sparse neural networks, as I will discuss in Chapter 7 (Section 7.4.1).

I will also discuss the challenges to obtain the same gains in *half-precision* as the ones we obtained with the new kernel in *float-precision* (Section 7.4.2).

This chapter has also demonstrated that some operations (the generic sparse matrix multiplication of `PyTorch`, and the new kernel, see Figure 5.10) are particularly performant in *batch-size-last*. This paves the way to revisit other common operations in neural networks within the *batch-size-last* memory layout.

Finally, translating our kernel into OpenCL could enable it to run on AMD hardware and other platforms. We also hope that our benchmark will serve as a baseline for comparing Kronecker-sparse implementations on other hardware, such as CPU, Intelligence Processing Unit, FPGA, etc.

Chapter 6

Approximation guarantees for quantized networks

This chapter focuses on the approximation power of neural networks, quantized or not, based on Gonon et al. [2023a]. It is disconnected from the theory on the path-lifting and path-activations developed in Chapters 2 to 4.

This is the first work I did during my PhD. I now have a different perspective on the results presented here, and I will comment on them as we go along. Because of that, this chapter is a mix of the original work and my current viewpoint. In Section 6.1, I give an high-level overview of the first half of the contribution in Gonon et al. [2023a] based on this new outlook. The other half of the paper Gonon et al. [2023a] is completely reproduced, starting at Section 6.2.1, but preceded by a new introduction, in Section 6.2, reflecting my updated perspective.

As already discussed in Chapter 1 and in the introduction of Chapter 5, the cost of machine learning at scale grows exponentially with years, and the time, energy and memory required to train and run neural networks is becoming a bottleneck for the deployment of AI in many applications. Besides sparsity, studied in Chapter 5, *quantization* is also an important technique widely used in practice to reduce the cost of neural networks. This is often done by decreasing the number of bits used to store the weights of the network, and more generally, to constrain them to a finite set. For instance, going from the IEEE standard 32-bit *float-precision* to the other IEEE standard 16-bit *half-precision* reduces the memory footprint of a neural network by a factor of two, makes training and inference faster, and reduces the energy consumption. However, the maximum value represented by a 16-bit *half-precision* ($\simeq 7e5$) is much smaller than the maximum value represented by a 32-bit *float-precision* ($\simeq 3e38$). This can lead to loss of accuracy in the approximation of the target function, and has therefore led to the development of alternative formats. For instance, for large language models, the *Brain Floating Point half-precision* is preferred over classical *half-precision*, for its wider range of values and its better empirical results in learning problems. Despite the development of these new formats, the quest for the best trade-off between the size of the formats and the accuracy provided by the neural network remains a crucial question in practice, and finding the right quantization scheme is for now a trial-and-error process. For this reason, I have been interested in investigating what could be said *in theory*. I found out that all the results on the approximation power of neural networks were based on the assumption that the weights of the network are *arbi-*

trary (unconstrained) real numbers, and that no result was available on the approximation power of neural networks with *quantized weights*.

This chapter addresses this challenge of better understanding the approximation power of quantized neural networks, e.g., by providing sufficient number of bits for quantized networks to have the same approximation rates as unquantized ones. This is done by combining existing results on the approximation power of neural networks with unconstrained real weights with new results on the quantization error of neural networks.

Another important question is to better understand non-trivial situations where neural networks, quantized or not, can be expected (or not) to have better approximation properties than the best known approximation families[1] such as polynomials or wavelets. **This chapter lays a framework to identify such situations.** This is done by introducing a property of the approximation family at hand, called ∞-*encodability*.

The outline is as follows.

- Section 6.1 gives an high-level overview of the results of Gonon et al. [2023a] on the approximation power of *quantized* neural networks. A practical type of quantization on which I will focus is quantization to the nearest-neighbour on a finite subset of a uniform grid (see, e.g., Theorem 6.1.1).
- Section 6.2 introduces the notion of ∞-encodability (Definition 6.2.2), a property of approximation families that I show to be useful to identify fundamental limits of neural network approximation (Theorem 6.2.1). This property has been implicitly used on a case-by-case basis in the literature to show that many classical approximation families share a fundamental limit in their approximation power. The new framework I introduce unifies and generalizes these results, hopefully providing a clearer perspective on such fundamental limits of neural network approximation.

6.1 Approximation by quantized neural networks

Given a metric d, an accuracy $\varepsilon > 0$ and a target function f, the problem of approximating f with a neural network is to find a DAG architecture G (Definition 2.2.2) such that there are parameters $\theta \in \mathbb{R}^G$ satisfying

$$d(f, R_\theta) \leqslant \varepsilon.$$

It has been shown in the literature that there exist such G and θ for many f, d, and ε, with explicit bounds on the size of G and the ℓ^q-norms of θ. These bounds have been provided for classifier functions f in L^2 [Petersen and Voigtländer, 2018], functions f in Hölder spaces [Ohn and Kim, 2019], Sobolev spaces [Gühring et al., 2019, Gribonval et al., 2022], and even more general Besov spaces [Suzuki, 2019, Gribonval et al., 2022].

However, these results use unconstrained real weights θ, while the parameters available on a machine are constrained to a finite set (e.g., floats). In this section, I investigate the effect of quantization on the approximation properties of neural networks: I call *quantization scheme* any function $Q : \mathbb{R}^G \mapsto \mathbb{R}^G$ with a finite image, and I am interested in

[1] An approximation family is any (often non-decreasing) sequence $(\Sigma_M)_{M=1,2,\dots}$ of subsets of a metric space $(\mathcal{F}, d)$.

the same question as above, but this time the goal is to have the approximation guarantee in terms of the quantized network $R_{Q(\theta)}$ instead of the unquantized one R_θ:

$$d(f, R_{Q(\theta)}) \leqslant \varepsilon.$$

A general quantization scheme as above corresponds to vector quantization. I will specifically focus on the special case of uniform scalar quantization, where Q acts coordinate-wise according to $Q(\theta)_i = \lfloor \theta_i/\eta \rfloor \eta$ for some step size $\eta > 0$. This is a simple and practical quantization scheme. The goal is to provide guarantees on the approximation power of LFCNs with quantized weights using this quantization scheme.

To derive guarantees, I will base myself on the simple triangle inequality:

$$d(f, R_{Q(\theta)}) \leqslant d(f, R_\theta) + d(R_\theta, R_{Q(\theta)}). \tag{6.1}$$

Given d, ε and f, existing results can be used to find an architecture G and (unquantized) parameters θ such that $d(f, R_\theta) \leqslant \varepsilon$ [Petersen and Voigtländer, 2018, Ohn and Kim, 2019, Gühring et al., 2019, Gribonval et al., 2022], hence controlling the first term of Inequality (6.1). Once we have such an architecture and parameters θ, I then bound the second term of (6.1) using the Lipschitz property of the network with respect to the parameters θ:

$$d(R_\theta, R_{Q(\theta)}) \leqslant \tilde{d}(\theta, Q(\theta))$$

for some metric $\tilde{d}$. The latter allows me to give a sufficient number of bits for the quantization scheme Q to guarantee the desired accuracy ε. Below, I describe in Section 6.1.1 how I control the quantization error $d(R_\theta, R_{Q(\theta)})$ in terms of the number of bits of the quantization scheme Q. Combined to existing approximation results for unconstrained real-weights networks, I can apply this to provide guarantees on the approximation power of LFCNs with quantized weights using the simple Equation (6.1). An example of application for Sobolev functions is given in Section 6.1.2. Section 6.1.3 concludes by establishing a sufficient number of bits for quantized networks to have the same polynomial asymptotic approximation rates.

6.1.1 Controlling the quantization error

Gonon et al. [2023a] controls the quantization error $d(R_\theta, R_{Q(\theta)})$ for *LFCNs* using Lipschitz bounds in θ of the same type as the one seen in Equation (3.7):

$$\|R_\theta(x) - R_{Q(\theta)}(x)\|_1 \leqslant (W\|x\|_\infty + 1) W L^2 R^{L-1} \|\theta - \theta'\|_\infty$$

where W is the width of the network, L is the number of affine layers, and R is a bound on some operator norm of each affine layer. The first contribution of Gonon et al. [2023a] is to generalize this type of Lipschitz property that were known in special cases [Berner et al., 2020, Theorem 2.6][Neyshabur et al., 2018, Lemma 2] to general L^p-norms on the function space and general operator norms on the affine layers, see Theorem III.1 in Gonon et al. [2023a]. *The details are tedious and I do not reproduce them here.* I then use this Lipschitz property to control the quantization error $\|R_\theta(x) - R_{Q(\theta)}(x)\|_{L^p}$ in terms of the number of bits of the quantization scheme Q. An example of what we can get is given in the following theorem.

Theorem 6.1.1. *Consider a LFCN with $L \geqslant 2$ affine layers and layers of neurons being of respective dimension $N_0, \dots, N_{L+1}$ (Definition 2.1.1). Denote $W = \max_{\ell=0,\dots,L} N_\ell$ the width of the architecture. Consider the space $\mathcal{F} = L^\infty([-D,D]^{d_{in}} \to (\mathbb{R}^{d_{out}}, \|\cdot\|_\infty), \mu)$ with μ the Lebesgue measure.*

Consider $\varepsilon \in (0, 1/2)$ and parameters θ. Let $k \geqslant 0$ be the smallest integer such that $\|\theta\|_\infty \leqslant \varepsilon^{-k}$ and $\max(W, L) \leqslant \varepsilon^{-k}$, i.e., $k = \lceil \log_2 \max(\|\theta\|_\infty, W, L) / \log_2(1/\varepsilon) \rceil$. For every integer $m \geqslant 2kL + k + 1 + \log_2(\lceil D \rceil)$, the weights of θ can be rounded up to a closest point in $\eta\mathbb{Z} \cap [-\varepsilon^{-k}, \varepsilon^{-k}]$ with $\eta := 2^{-m\lceil \log_2(\varepsilon^{-1}) \rceil} \leqslant \varepsilon^m$ to obtain $Q_\eta(\theta) \in (\eta\mathbb{Z})^{\#\text{params}}$ satisfying

$$\|Q_\eta(\theta)\|_\infty \leqslant \varepsilon^{-k} \text{ and } \|R_\theta - R_{Q_\eta(\theta)}\|_{L^\infty} \leqslant \varepsilon.$$

Theorem 6.1.1 gives an explicit number of bits that ensures that uniform quantization to the nearest neighbour yields a quantization error of at most ε. This condition improves on Lemma VI.8 in Elbrächter et al. [2021] essentially by a factor kL in the number of bits (in m).

However, I did that before being familiar with the path-lifting theory. Based on what we saw in Chapters 2 to 4, it is now clear that the tightest way to exploit a Lipschitz property expressed directly in terms of the raw parameters is often by first reducing to normalized parameters (Definition 3.2.1). Indeed, Lipschitz properties expressed in terms of raw θ are not invariant to rescaling symmetries, and the infimum of such a bound over the weak equivalence class of θ is often achieved by normalizing θ (see, e.g., Theorem 3.3.2). Therefore, it is expected that the results of Gonon et al. [2023a] can be both improved using the path-lifting, and generalized to arbitrary DAG networks since the path-lifting provides a Lipschitz property for any DAG network. *For this reason, Section 6.1.3 only presents the ideas of Gonon et al. [2023a], but not the proofs nor the details as they are tedious and likely to be improved.*

6.1.2 Application to Sobolev functions

I just showed how to control, in terms of the number of bits of the quantization scheme Q, the quantization error $\|R_\theta - R_{Q_\eta(\theta)}\|_{L^\infty}$, i.e., the first term of the right-hand side of Inequality (6.1):

$$d(f, R_{Q(\theta)}) \leqslant d(f, R_\theta) + d(R_\theta, R_{Q(\theta)}).$$

To get guarantees of approximation for quantized networks, it remains to control the first term of the right-hand side using existing results on the approximation power of neural networks with unconstrained real weights θ.

As an example, let's consider the case of functions f in an L^∞-Sobolev space. Given $\mathbf{n} := (n_1, \dots, n_d) \in \mathbb{N}_{>0}^d$, denote by $D^{\mathbf{n}} f$ the associated weak-derivative of f, if it exists. Consider $n \in \mathbb{N}_{>0}$ and denote by $\mathcal{W}^{n,\infty}([0,1]^d)$ the Sobolev space of real-valued functions on $[0,1]^d$ that are, with all their $\mathbf{n}$-weak derivatives up to order n ($\|\mathbf{n}\|_1 \leqslant n$), in L^∞. The norm on $\mathcal{W}^{n,\infty}([0,1]^d)$ is given by:

$$\|f\|_{\mathcal{W}^{n,\infty}([0,1]^d)} := \max_{\substack{\mathbf{n}:=(n_1,\dots,n_d)\in\mathbb{N}_{>0}^d \\ \sum_i n_i \leqslant n}} \operatorname{ess\,sup}_{x\in[0,1]^d} |D^{\mathbf{n}} f(x)|.$$

Theorem 6.1.2 ([Ding et al., 2019, Thm. 2]). *Let $\mathcal{C}_{n,d}$ be the unit ball of $\mathcal{W}^{n,\infty}([0,1]^d)$. There exists a constant $c>0$ depending only on n and d such that for every $\varepsilon \in (0,1)$, there exists $\eta > 0$ satisfying $\ln(1/\eta) \leqslant c\ln^2(1/\varepsilon)$ and a LFCN architecture (Definition 2.1.1) that can approximate every function $f \in \mathcal{C}_{n,d}$ within error $\varepsilon > 0$ in $L^\infty([0,1]^d)$ using weights in $\eta\mathbb{Z}$, with depth bounded by $c\ln(1/\varepsilon)$, a number of weights at most equal to $c\varepsilon^{-d/n}\ln(1/\varepsilon)$, and with a total number of bits (used to store the network weights) bounded by $c\varepsilon^{-d/n}\ln^3(1/\varepsilon)$.*

There is no particular difficulty so I skip the proof and invite the interested reader to refer directly to the paper [Gonon et al., 2023a].

Theorem 6.1.2 is an application of Theorem 6.1.1, recovering a special case of Theorem 2 in Ding et al. [2019] (the other cases can be recovered by combining this special case with Proposition 3 in Ding et al. [2019]). But the framework introduced here is more general as we have derived a general Lipschitz property in any L^p-space, so we could derive similar consequences not only for a function f in the unit ball of an L^∞-Sobolev space, but for every $f \in L^p$ $(1 \leqslant p \leqslant \infty)$ *as soon as it is known how to approximate f with unquantized ReLU networks, with explicit bounds on the growth of their depth, width and weight's magnitude.* For instance, such bounds are known for Hölder spaces Ohn and Kim [2019], classifier functions in L^2 Petersen and Voigtländer [2018] and Besov spaces Suzuki [2019]. The same argument also applies for networks with arbitrary Lipschitz activation (such as the sigmoid function) for which an analog Lipschitz bound on $\theta \mapsto R_\theta$ can be derived, and for which we know how to approximate "smooth" functions [Gühring et al., 2019, Table 1].

Another interesting question is, instead of fixing an accuracy ε and deriving bounds on the size of the architecture and the number of bits sufficient to meet this accuracy, to rather fix a sequence $\Sigma = (\Sigma_M)_{M\in\mathbb{N}_{>0}}$ of sets of network functions with growing architectures and growing numbers of bits[2], and study the polynomial rate at which $d(f,\Sigma_M)$ converges to zero as M grows. This is the object of Section 6.1.3.

6.1.3 Approximation speed of quantized neural networks

Given a subset $\mathcal{C}$ of a metric function space $(\mathcal{F},d)$ and an approximation family $\Sigma = (\Sigma_M)_{M\in\mathbb{N}_{>0}}$ in $\mathcal{F}$, this leads to the notion of *the polynomial asymptotic approximation speed $\gamma^{*approx}(\mathcal{C}|\Sigma)$ of $\mathcal{C}$ by Σ* [Elbrächter et al., 2021, Def. V.2, Def. VI.1], called simply approximation speed in what follows. This *is the best polynomial rate at which all functions of $\mathcal{C}$ are asymptotically approximated by Σ*:

$$\gamma^{*\text{approx}}(\mathcal{C}|\Sigma) := \sup\{\gamma\in\mathbb{R}, \sup_{f\in\mathcal{C}} \inf_{g\in\Sigma_M} d(f,g) = \mathcal{O}_{M\to\infty}\left(M^{-\gamma}\right)\},$$

with the convention $\gamma^{*\text{approx}}(\mathcal{C}|\Sigma) = -\infty$ if the supremum is over an empty set.

Consider a function $f\in L^p$ and a sequence of parameters $(\theta_M)_{M\in\mathbb{N}_{>0}}$. Can we design a sequence (Q_M) of quantization schemes such that the realizations of the networks with quantized parameters $(Q_M(\theta_M))_{M\in\mathbb{N}_{>0}}$ approximate the function f at the same asymptotic polynomial rate, with M, as the unquantized parameters $(\theta_M)_{M\in\mathbb{N}_{>0}}$? Using the triangle inequality $\|f - R_{Q_M(\theta_M)}\|_p \leqslant \|f - R_{\theta_M}\|_p + \|R_{\theta_M} - R_{Q_M(\theta_M)}\|_p$ for each integer

[2] Typically $\Sigma_M \subset \Sigma_{M+1}$.

M, it is sufficient to guarantee that $\|R_{\theta_M} - R_{Q_M(\theta_M)}\|_p$ decreases at the same polynomial asymptotic rate as $\|f - R_{\theta_M}\|_p$. This can be done using similar controls on the quantization error as the ones established in Section 6.1.1. I now give an *informal version of the results proved in Gonon et al. [2023a], and leave the details for the paper.*

In the following result, I exhibit a sufficient number of bits per coordinate that guarantees that nearest-neighbour uniform quantization preserves approximation rates of approximation families defined with LFCNs (Definition 2.1.1).

Informal Theorem 6.1.1 (see Theorem V.1 in Gonon et al. [2023a])**.** *Consider the approximation family $\Sigma = (\Sigma_M)_{M\in\mathbb{N}_{>0}}$ in an arbitrary L^p space, such that Σ_M is the set of functions realized by LFCNs (Definition 2.1.1) with at most $L_M \in \mathbb{N}_{>0}$ affine layers, with parameters having at most M non-zero coordinates and with Euclidean norm bounded by $r_M \geqslant 1$. Denote by $Lips(L, W, r)$ the Lipschitz constant of $\theta \mapsto R_\theta \in L^p$ on the set of LFCNs with L affine layers, width W and parameters of Euclidean norm at most equal to r.*

For $\gamma > 0$, consider the γ-uniformly quantized *sequence $\mathcal{Q}(\Sigma|\gamma) := (\mathcal{Q}_M(\Sigma_M|\gamma))_{M\in\mathbb{N}_{>0}}$, where $\mathcal{Q}_M(\Sigma_M|\gamma)$ is the set of functions realized by LFCNs as above, but with parameters uniformly quantized using the quantization scheme $Q_{\eta_M}(x) = \lfloor x/\eta_M \rfloor \eta_M$ for a step size $\eta_M = M^{-\gamma} Lips(M, L_M, r_M)$. Then, the* γ-uniformly quantized *sequence $\mathcal{Q}(\Sigma|\gamma)$ has, on every set $\mathcal{C} \subset L^p$, an approximation speed which is comparable to its unquantized version Σ:*

$$\begin{array}{ll} \gamma^{*approx}(\mathcal{C}|\mathcal{Q}(\Sigma|\gamma)) = \gamma^{*approx}(\mathcal{C}|\Sigma) & \text{if } \gamma \geqslant \gamma^{*approx}(\mathcal{C}|\Sigma), \\ \gamma^{*approx}(\mathcal{C}|\mathcal{Q}(\Sigma|\gamma)) \geqslant \gamma & \text{otherwise.} \end{array}$$

This theorem leads to explicit conditions on the number of bits per coordinate that guarantee quantized LFCNs to have the same approximation speeds as unquantized ones, see Example V.1 in Gonon et al. [2023a]. In the proof, approximation speeds are matched by (i) taking unquantized parameters that (almost) achieve the unquantized approximation speed and (ii) quantizing these parameters with a sufficiently large number of bits in order to preserve the approximation speed. Smarter (but computationally more challenging) quantization schemes can be envisioned, such as directly picking the best quantized parameters to approximate the function. If the budget for the number of bits per coordinate is larger than the one given in Informal Theorem 6.1.1, then even the smartest quantization scheme will not beat approach $(i) + (ii)$ in terms of polynomial approximation speed (but it can still have better constants/log-terms etc.). Indeed, $(i) + (ii)$ already yields the same approximation speeds for quantized networks as unquantized ones, and quantized networks cannot do better than unquantized ones. An open question is: what is the minimum number of bits per coordinate needed to keep the same approximation speeds? I provide a partial answer by providing an upper-bound in Informal Theorem 6.1.1.

This concludes the presentation of the results of Gonon et al. [2023a] about the approximation speed of quantized neural networks. I now turn to another result of Gonon et al. [2023a] about a fundamental limit of neural network approximation.

6.2 Fundamental limits of neural network approximation

Given an approximation family $\Sigma = (\Sigma_M)_{M\in\mathbb{N}_{>0}}$ and a target set $\mathcal{C}$ to approximate, I am interested in better understanding situations where it holds:

$$\gamma^{*\text{approx}}(\mathcal{C}|\Sigma) \leqslant \gamma^{*\text{encod}}(\mathcal{C}), \tag{6.2}$$

where $\gamma^{*\text{encod}}(\mathcal{C})$ is the *encoding speed* of $\mathcal{C}$, also called Kolmogorov-Donoho complexity [DeVore et al., 2021, Elbrächter et al., 2021]. The encoding speed is an *information-theoretic* quantity that measures how much we could compress a set when encoded at asymptotically small precision. This is done by looking at the rate at which the number of balls needed to cover the set grows as the radius of the balls decreases. Formally, consider the *metric entropy* $H(\mathcal{C}, d, \varepsilon) := \log_2(N(\mathcal{C}, d, \varepsilon))$ (recall the definition of covering numbers in Definition 4.3.1). The encoding speed of $\mathcal{C}$ is defined as [Elbrächter et al., 2021, Def. IV.1]:

$$\gamma^{*\text{encod}}(\mathcal{C}) := \sup\left\{\gamma > 0, H(\mathcal{C}, d, \varepsilon) = \mathcal{O}_{\varepsilon\to 0}(\varepsilon^{-1/\gamma})\right\}, \tag{6.3}$$

with the convention that $\gamma^{*\text{encod}}(\mathcal{C}) = 0$ if the supremum is over an empty set.

Inequality (6.2) relates an approximation quantity, $\gamma^{*\text{approx}}(\mathcal{C}|\Sigma)$, to an information-theoretic quantity $\gamma^{*\text{encod}}(\mathcal{C})$. This inequality is important for several reasons.

- **It is challenging to derive an upper-bound on the approximation speed.** Deriving an upper bound on the approximation speed $\gamma^{*\text{approx}}(\mathcal{C}|\Sigma)$ is in general more challenging than deriving a lower-bound. For each function $f \in \mathcal{C}$, a lower bound is obtained as soon as we can exhibit *one* sequence of functions $(g_M)_{M\in\mathbb{N}_{>0}}$ in Σ such that $d(f, g_M)$ converges to zero at a polynomial rate. Such constructions are well-known in many cases of interests, such as when f is smooth and Σ_M is the set of polynomials of degree M (Taylor-like approximation), or when f is in a Sobolev space and Σ_M is a set of wavelets, or when f is a polynomial and Σ_M is a set of neural networks [Gribonval et al., 2022]. However, deriving an upper bound on the approximation speed $\gamma^{*\text{approx}}(\mathcal{C}|\Sigma)$ requires to prove that there is at least one f such that *any sequence* of functions $(g_M)_{M\in\mathbb{N}_{>0}}$ in Σ cannot converge to f at a given polynomial rate. The fact that all the sequences have to be ruled out makes this task more challenging.

- **The encoding speed is known for many $\mathcal{C}$'s**, see [Elbrächter et al., 2021, Table 1].

- **Inequality (6.2) is often sharp** as it is often an equality, see [Elbrächter et al., 2021, Table 1].

- **Inequality (6.2) is a reasonable fundamental limit to the approximation capability of an approximation family.** We will see that in order for this inequality to *not hold*, this requires making very strong growth assumptions on the sets of the approximation family $\Sigma = (\Sigma_M)_M$. This means that all *reasonable* approximation families Σ must satisfy this inequality, which can be interpreted as a fundamental limit to how well a given set $\mathcal{C}$ can be approximated by any (reasonable) given approximation family Σ.

Inequality (6.2) has been established on a case-by-case basis in the literature, for instance when Σ is defined with dictionaries [Elbrächter et al., 2021, Thm. V.3][Grohs, 2015, Thm. 5.24] or LFCNs (Definition 2.1.1) [Elbrächter et al., 2021, Thm. VI.4]. In Section 6.2.1, I introduce the notion of γ-encodability for approximation families Σ, that measures how well an approximation family can be covered by balls. Section 6.2.2 shows that Inequality (6.2) can be understood as a consequence of this new notion of γ-encodability. *What I like about γ-encodability is that it is a property of Σ alone, regardless of the target set $\mathcal{C}$, and it implies Inequality (6.2) for* all *target sets $\mathcal{C}$. I find this elegant as it means that Inequality (6.2) describes a fundamental limit to the approximation capability that is inherently encoded in Σ. In particular, it does not depend on how well this approximation family is adapted to a given $\mathcal{C}$.* Section 6.2.3 then shows that many approximation families are ∞-encodable, recovering in particular previous known cases where Inequality (6.2) holds [Elbrächter et al., 2021, Thm. V.3][Grohs, 2015, Thm. 5.24][Elbrächter et al., 2021, Thm. VI.4].

6.2.1 Notion of γ-encodability

Let $\Sigma := (\Sigma_M)_{M\in\mathbb{N}_{>0}}$ be a sequence of non-empty subsets of a metric space $(\mathcal{F}, d)$. Consider $\mathcal{C} \subset \mathcal{F}$ and $\varepsilon > 0$. If $\gamma^{*\text{approx}}(\mathcal{C}|\Sigma) > 0$, since Σ approximates $\mathcal{C}$ at speed $\gamma^{*\text{approx}}(\mathcal{C}|\Sigma)$, there exists a positive integer M large enough such that every element $f \in \mathcal{C}$ can be ε-approximated (with respect to the metric d) by an element of Σ_M. Since Σ_M can be ε-covered (with respect to d) with $N(\Sigma_M, d, \varepsilon)$ elements, $\mathcal{C}$ can be 2ε-covered with $N(\Sigma_M, d, \varepsilon)$ elements. Instances of this simple reasoning can be found in [Elbrächter et al., 2021, Thm. V.3, Thm. VI.4][Grohs, 2015, Thm. 5.24][Kerkyacharian and Picard, 2004, Prop. 11]. This suggests the existence of a relation between the approximation speed $\gamma^{*\text{approx}}(\mathcal{C}|\Sigma)$ and the encoding speed $\gamma^{*\text{encod}}(\mathcal{C})$ that depends on the growth with M of the covering numbers of Σ_M.

I claim that a "reasonable" growth of the covering numbers of Σ_M consists in a situation where, for some $\gamma > 0$, the set Σ_M can be $M^{-\gamma}$-covered with "roughly" $2^{M \log M}$ elements. Indeed, this covers the case where each element of Σ_M can be described by M parameters that can be stored with a number of bits per parameter that grows logarithmically in M. For instance if Σ_M is a bounded set in dimension M then it can be uniformly quantized along each dimension with a size step of order $M^{-\gamma}$, so that $\log M$ bits is roughly enough to encode each of the M coordinates. This "reasonable" growth for the covering numbers of Σ_M is formalized in Definition 6.2.2, and yields the simple relation $\min(\gamma^{*\text{approx}}(\mathcal{C}|\Sigma), \gamma) \leqslant \gamma^{*\text{encod}}(\mathcal{C})$ for every set $\mathcal{C} \subset \mathcal{F}$, as shown in Theorem 6.2.1.

Definition 6.2.1 ((γ, h)-encoding)**.** *Consider a metric space $(\mathcal{F}, d)$, an arbitrary sequence $\Sigma := (\Sigma_M)_{M\in\mathbb{N}_{>0}}$ of (non-empty) subsets of $\mathcal{F}$, and $\gamma > 0$ and $h > 0$. A sequence $(\Sigma(\gamma, h)_M)_{M\in\mathbb{N}_{>0}}$ is said to be a* (γ, h)-encoding *of Σ if there exist constants $c_1, c_2 > 0$ such that for every $M \in \mathbb{N}_{>0}$, the set $\Sigma(\gamma, h)_M$ is a $c_1 M^{-\gamma}$-covering of Σ_M (recall Definition 4.3.1, in particular $\Sigma(\gamma, h)_M$ must be a subset of Σ_M) of size satisfying $\log_2(|\Sigma(\gamma, h)_M|) \leqslant c_2 M^{1+h}$.*

The following definition captures a "reasonable" growth with M of the covering numbers of Σ_M.

Definition 6.2.2 (γ-encodable Σ in $(\mathcal{F}, d)$)**.** *Let $(\mathcal{F}, d)$ be a metric space. Let $\Sigma := (\Sigma_M)_{M\in\mathbb{N}_{>0}}$ be an arbitrary sequence of (by default, non-empty) subsets of $\mathcal{F}$. Consider $\gamma > 0$. We say that Σ is* γ-encodable *in $(\mathcal{F}, d)$ if for every $h > 0$, there exists a (γ, h)-encoding of Σ. We say that Σ is* ∞-encodable *in $(\mathcal{F}, d)$ if it is γ-encodable in $(\mathcal{F}, d)$ for all $\gamma > 0$. When the context is clear, we will omit the mention to $(\mathcal{F}, d)$.*

Note that if Σ is γ-encodable then it is γ'-encodable for every $\gamma' \leqslant \gamma$. Several examples of ∞-encodable sequences are given in Section 6.2.3, including classical approximation families defined with dictionaries or LFCNs (Definition 2.1.1).

6.2.2 The encoding speed as a universal upper bound for approximation speeds

It is known that $\gamma^{*\text{approx}}(\mathcal{C}|\Sigma) \leqslant \gamma^{*\text{encod}}(\mathcal{C})$ for various sets $\mathcal{C}$ when Σ is defined with neural networks [Elbrächter et al., 2021, Thm. VI.4] or dictionaries [Elbrächter et al., 2021, Thm. V.3][Grohs, 2015, Thm. 5.24]. The following proposition shows that ∞-encodability implies $\gamma^{*\text{approx}}(\mathcal{C}|\Sigma) \leqslant \gamma^{*\text{encod}}(\mathcal{C})$. This settles a unified and generalized framework for the aforementioned known cases that implicitly use, one way or another, the ∞-encodability property, as I will detail in section 6.2.3.

Theorem 6.2.1 ([Gonon et al., 2023a])**.** *Consider $(\mathcal{F}, d)$ a metric space and $\Sigma := (\Sigma_M)_{M\in\mathbb{N}_{>0}}$ an arbitrary sequence of (non-empty) subsets of $\mathcal{F}$ which is γ-encodable in $(\mathcal{F}, d)$, with $\gamma \in (0, \infty]$. Then for every (non-empty) $\mathcal{C} \subset \mathcal{F}$:*

$$\min(\gamma^{*approx}(\mathcal{C}|\Sigma), \gamma) \leqslant \gamma^{*encod}(\mathcal{C}).$$

Proof of Theorem 6.2.1. If $\gamma^{*\text{approx}}(\mathcal{C}|\Sigma) \leqslant 0$ then the result is trivial since we always have $\gamma^{*\text{encod}}(\mathcal{C}) \geqslant 0$. In the rest of the proof we assume $\gamma^{*\text{approx}}(\mathcal{C}|\Sigma) > 0$. Fix $0 < \gamma' < \min(\gamma^{*\text{approx}}(\mathcal{C}|\Sigma), \gamma)$ and $h > 0$. First, Σ is γ-encodable so there exists a (γ, h)-encoding of Σ that we denote $\Sigma(\gamma, h)$. This means that there exist constants $c_1', c_2' > 0$ such that for every $M \in \mathbb{N}$, the set $\Sigma(\gamma, h)_M$ is a $c_1' M^{-\gamma}$-covering of Σ_M of size $|\Sigma(\gamma, h)_M| \leqslant 2^{c_2' M^{1+h}}$. Second, since $0 < \gamma' < \min(\gamma^{*\text{approx}}(\mathcal{C}|\Sigma), \gamma)$, the definition of the approximation speed guarantees that there exists a constant $c_3' > 0$ such that for every $f \in \mathcal{C}$ and every $M \in \mathbb{N}$, there exists a function $\Phi_M(f) \in \Sigma_M$ that satisfies:

$$d\left(f, \Phi_M(f)\right) \leqslant c_3' M^{-\gamma'}.$$

Since $0 < \gamma' < \gamma$, note that for every $M \in \mathbb{N}$, it holds $c_1' M^{-\gamma} + c_3' M^{-\gamma'} \leqslant (c_1' + c_3') M^{-\gamma'}$. Define $c_1 = c_1' + c_3'$ and $c_2 = c_2'$. We deduce that for every $M \in \mathbb{N}$, the set $\Sigma(\gamma, h)_M$ is a $c_1 M^{-\gamma'}$-covering of $\mathcal{C}$ of size $|\Sigma(\gamma, h)_M| \leqslant 2^{c_2 M^{1+h}}$. Now, for every $\varepsilon > 0$, the integer $M_\varepsilon := \left\lceil \left(\frac{c_1}{\varepsilon}\right)^{1/\gamma'} \right\rceil$ satisfies $\varepsilon \geqslant c_1 M_\varepsilon^{-\gamma'}$. By monotonicity of the metric entropy $H(\mathcal{C}, d, \cdot)$ we get $H(\mathcal{C}, d, \varepsilon) \leqslant H(\mathcal{C}, d, c_1 M_\varepsilon^{-\gamma'}) \leqslant c_2 M_\varepsilon^{1+h}$. Note that for $0 < \varepsilon < c_1$, denoting by $c = (2c_1^{1/\gamma'})^{1+h}$ it holds $M_\varepsilon^{1+h} \leqslant \left(1 + \left(\frac{c_1}{\varepsilon}\right)^{1/\gamma'}\right)^{1+h} = \left(\frac{c_1}{\varepsilon}\right)^{(1+h)/\gamma'} \left(1 + \left(\frac{\varepsilon}{c_1}\right)^{1/\gamma'}\right)^{1+h} \leqslant c\varepsilon^{-(1+h)/\gamma'}$. Finally for every $0 < \varepsilon < c_1$, it holds

$$H(\mathcal{C}, d, \varepsilon) \leqslant c\varepsilon^{-(1+h)/\gamma'},$$

As a direct consequence of Equation (6.3), this implies $\gamma^{*\text{encod}}(\mathcal{C}) \geqslant \frac{\gamma'}{1+h}$ for every $h > 0$ and every $0 < \gamma' < \min(\gamma^{*\text{approx}}(\mathcal{C}|\Sigma), \gamma)$, hence the desired result. □

I derive from Theorem 6.2.1 a generic lower bound on the encoding speed of the set of functions uniformly approximated at a given speed.

Corollary 6.2.1 ([Gonon et al., 2023a])**.** *Let $(\mathcal{F}, d)$ be a metric space. Consider $\gamma \in (0, \infty]$ and $\Sigma := (\Sigma_M)_{M\in\mathbb{N}_{>0}}$ an arbitrary sequence of (non-empty) subsets of $\mathcal{F}$ which is γ-encodable in $(\mathcal{F}, d)$. Consider $\alpha, \beta > 0$ and $\mathcal{A}^\alpha(\mathcal{F}, \Sigma, \beta)$ the set of all $f \in \mathcal{F}$ such that $\sup_{M\geqslant 1} M^\alpha d(f, \Sigma_M) \leqslant \beta$. This set satisfies*

$$\gamma^{*encod}(\mathcal{A}^\alpha(\mathcal{F}, \Sigma, \beta)) \geqslant \min(\alpha, \gamma).$$

Proof. By the very definition of $\mathcal{A}^\alpha(\mathcal{F}, \Sigma, \beta)$, it holds $\gamma^{*\text{approx}}(\mathcal{A}^\alpha(\mathcal{F}, \Sigma, \beta)|\Sigma) \geqslant \alpha$. Theorem 6.2.1 then gives the result. □

The reader may wonder about the role of β in the above result, and whether a similar result can be achieved with $\mathcal{A}^\alpha(\mathcal{F}, \Sigma) := \cup_{\beta>0}\mathcal{A}^\alpha(\mathcal{F}, \Sigma, \beta)$. While this is left open, a related discussion after Corollary 6.2.4 suggests this may not be possible without additional assumptions on Σ.

As an immediate corollary of Theorem 6.2.1 we also obtain the following result.

Corollary 6.2.2 ([Gonon et al., 2023a])**.** *Consider $\Sigma := (\Sigma_M)_{M\in\mathbb{N}_{>0}}$ an arbitrary sequence of (non-empty) subsets of a metric space $\mathcal{F}$ and a (non-empty) set $\mathcal{C} \subset \mathcal{F}$. If Σ is γ-encodable for every $\gamma < \gamma^{*approx}(\mathcal{C}|\Sigma)$ then:*

$$\gamma^{*approx}(\mathcal{C}|\Sigma) \leqslant \gamma^{*encod}(\mathcal{C}).$$

Proof. For every $\gamma < \gamma^{*\text{approx}}(\mathcal{C}|\Sigma)$, since Σ is γ-encodable, we have $\gamma = \min(\gamma^{*\text{approx}}(\mathcal{C}|\Sigma), \gamma) \leqslant \gamma^{*\text{encod}}(\mathcal{C})$ by Proposition 6.2.1. Taking the supremum of such γ, we get the inequality. □

As I will show in Section 6.2.3, applying Corollary 6.2.2 to specific ∞-encodable sequences allows one to unify and generalize different cases where $\gamma^{*\text{approx}}(\mathcal{C}|\Sigma) \leqslant \gamma^{*\text{encod}}(\mathcal{C})$ is known to hold [Elbrächter et al., 2021, Thm. V.3, Thm. VI.4][Grohs, 2015, Thm. 5.24].

Note that the quantity $\gamma^{*\text{encod}}(\mathcal{C})$ is known in several cases, see [Elbrächter et al., 2021, Table 1]. In the next section, I discuss concrete examples of ∞-encodable sequences Σ. For such a sequence Σ and an arbitrary set $\mathcal{C}$, independently of the adequation of Σ and $\mathcal{C}$, Corollary 6.2.2 automatically yields an upper bound for the approximation speed of $\mathcal{C}$ by Σ.

In some situations, the converse of Corollary 6.2.2 can be established.

Theorem 6.2.2. *Let $\mathcal{C}$ be a (non-empty) subset of a metric space $(\mathcal{F}, d)$ and $\Sigma := (\Sigma_M)_{M\in\mathbb{N}_{>0}}$ a sequence of (non-empty) subsets of $\mathcal{F}$ such that $\Sigma_M \subset \mathcal{C}$ for every M large enough. If $\min(\gamma^{*approx}(\mathcal{C}|\Sigma), \gamma^{*encod}(\mathcal{C})) > 0$ then the sequence Σ is γ-encodable for each γ, $0 < \gamma < \min(\gamma^{*approx}(\mathcal{C}|\Sigma), \gamma^{*encod}(\mathcal{C}))$. In particular, if $\gamma^{*approx}(\mathcal{C}|\Sigma) \leqslant \gamma^{*encod}(\mathcal{C})$ then Σ is γ-encodable for every $0 < \gamma < \gamma^{*approx}(\mathcal{C}|\Sigma)$.*

Proof. Fix $0 < \gamma < \min(\gamma^{*\text{approx}}(\mathcal{C}|\Sigma), \gamma^{*\text{encod}}(\mathcal{C}))$. By definition of $\gamma^{*\text{approx}}(\mathcal{C}|\Sigma)$, there exists a constant $c > 0$ such that for every $f \in \mathcal{C}$ and every $M \in \mathbb{N}_{>0}$, there exists $g_M(f) \in \Sigma_M$ such that $d(f, g_M(f)) \leqslant cM^{-\gamma}$. Consider $\gamma' > 0$ such that $\gamma < \gamma' < \min(\gamma^{*\text{approx}}(\mathcal{C}|\Sigma), \gamma^{*\text{encod}}(\mathcal{C}))$. For $M \in \mathbb{N}_{>0}$, define $\varepsilon_M := M^{-\gamma}$. By definition of $\gamma^{*\text{encod}}(\mathcal{C})$, there is a constant $c' > 0$ such that for every $\varepsilon > 0$, there exists an ε-covering $\mathcal{C}_\varepsilon$ of $\mathcal{C}$ of size satisfying $\log_2(|\mathcal{C}_\varepsilon|) \leqslant c'\varepsilon^{-1/\gamma'}$. For M large enough, $\Sigma_M \subset \mathcal{C}$, hence for

every such M and every $f \in \Sigma_M$, there exists $f_{\varepsilon_M} \in \mathcal{C}_{\varepsilon_M}$ such that $d(f, f_{\varepsilon_M}) \leqslant \varepsilon_M$. Using the triangle inequality, we obtain that for every M large enough and every $f \in \Sigma_M$: $d(f, g_M(f_{\varepsilon_M})) \leqslant (1+c)M^{-\gamma}$. This shows that $g_M(\mathcal{C}_{\varepsilon_M})$ is a $(1+c)M^{-\gamma}$-covering of Σ_M of size satisfying $\log_2(|g_M(\mathcal{C}_{\varepsilon_M})|) \leqslant c' M^{\gamma/\gamma'}$, with $\gamma/\gamma' < 1$. This shows that Σ is γ-encodable. The rest of the claim follows. □

6.2.3 Examples of ∞-encodable approximation families

I now give several examples of ∞-encodable sequences Σ. I start with a gentle warmup, proving that some sequences of balls (in the sense of the metric space $\mathcal{F}$) of increasing radius and dimension are ∞-encodable. Quite naturally, ∞-encodability is preserved under some Lipschitz transformation, as shown in Theorem 6.2.3 in the specific case of ∞-encodable sequences of balls (this can be generalized to other ∞-encodable sequences, but this is not useful here). Then, I give examples of ∞-encodable sequences in the context of approximations with dictionaries (Theorem 6.2.4, Theorem 6.2.5), showing that Theorem 6.2.1 unifies and generalizes Theorem V.3 in Elbrächter et al. [2021] and Theorem 5.24 in Grohs [2015]. Finally, I give an example of an ∞-encodable approximation family defined with LFCNs (Theorem 6.2.6). Once again, Theorem 6.2.1 applied to this ∞-encodable sequence recovers a known result, see Example 6.2.1.

First examples of ∞-encodable sequences

I start with a gentle warmup, giving basic examples of ∞-encodable sequences in order to manipulate the new notion of encodability (Definition 6.2.2). Let $(\mathcal{F}, d)$ be a metric space and π be a positive function that grows at most polynomially. Let $\Sigma := (\Sigma_M)_{M \in \mathbb{N}_{>0}}$ be a sequence of sets $\Sigma_M \subset \mathcal{F}$ that can be *covered* with $N_M = \mathcal{O}_{M\to\infty}(2^{M\pi(\log M)})$ balls (with respect to the ambient metric space) centered in Σ_M of radius $\varepsilon_M = \mathcal{O}_{M\to\infty}(M^{-\gamma})$. Since $\mathcal{O}_{M\to\infty}(2^{M\pi(\log M)}) = \mathcal{O}_{M\to\infty}(2^{M^{1+h}})$ for every $h > 0$, it is clear from the definition that Σ is ∞-encodable. This is trivially the case when $\Sigma := (\Sigma_M)_{M \in \mathbb{N}_{>0}}$ is a sequence of finite sets $\Sigma_M \subset \mathcal{F}$ with at most $2^{M\pi(\log M)}$ elements since each Σ_M is an exact covering of itself. Another example consists of some sequences of balls (in the sense of the metric space $\mathcal{F}$) of increasing radius and dimension as described in the next lemma.

Lemma 6.2.1 ([Gonon et al., 2023a])**.** *Consider $q \in [1, \infty]$, $(d_M)_{M \in \mathbb{N}_{>0}} \in ()\mathbb{N}_{>0}^{\mathbb{N}_{>0}}$, $(r_M)_{M \in \mathbb{N}_{>0}}$ a sequence of real numbers satisfying $r_M \geqslant 1$ and define $\Sigma := (\Sigma_M)_{M \in \mathbb{N}_{>0}}$, with $\Sigma_M := B_{d_M, \|\cdot\|_q}(0, r_M)$ being the set of sequences of $\ell^q(\mathbb{N}_{>0})$ bounded by r_M and supported in the first d_M coordinates. Then, Σ is either ∞-encodable in $\ell^q(\mathbb{N}_{>0})$ or it is never γ-encodable in $\ell^q(\mathbb{N}_{>0})$, whatever $\gamma > 0$ is. Moreover, it is ∞-encodable if, and only if,*

$$d_M \left(\log_2(r_M) + 1\right) = \mathcal{O}_{M\to\infty}(M^{1+h}), \qquad \forall h > 0.$$

Proof of Lemma 6.2.1. Each Σ_M can be identified with the closed ball of radius r_M in dimension d_M with respect to the q-th norm, so that standard bounds on covering numbers [Wainwright, 2019, Eq. (5.9)] yield for every $0 < \varepsilon \leqslant r_M$:

$$d_M \log_2\left(\frac{r_M}{\varepsilon}\right) \leqslant H(\Sigma_M, \|\cdot\|_q, \varepsilon) \leqslant d_M \log_2\left(\frac{3r_M}{\varepsilon}\right). \tag{6.4}$$

For $\varepsilon = M^{-\gamma} (\leqslant 1 \leqslant r_M)$, we get:

$$d_M(\log_2(r_M) + \gamma \log_2(M)) \\ \leqslant H(\Sigma_M, \|\cdot\|_q, \varepsilon) \leqslant d_M(\log_2(3r_M) + \gamma \log_2(M)).$$

Everything is non-negative, so if the right hand-side is $\mathcal{O}_{M\to\infty}(M^{1+h})$, for every $h > 0$, then so is the left hand-side. The converse is also true since both sides only differ by $\log_2(3) d_M = \mathcal{O}_{M\to\infty}(d_M \log M)$. The non-negativity of the quantities also implies that the condition $d_M[\log_2(r_M)+\gamma \log_2(M)] = \mathcal{O}_{M\to\infty}(M^{1+h})$, for every $h > 0$, does not depend on γ. As a consequence, either Σ is ∞-encodable or it is never γ-encodable, whatever $\gamma > 0$ is. Finally, note that for every $h > 0$, $d_M(\log_2(r_M) + \log_2(M)) = \mathcal{O}_{M\to\infty}(M^{1+h})$ if and only if $d_M(\log_2(r_M) + 1) = \mathcal{O}_{M\to\infty}(M^{1+h})$. The "only if" part is clear since for $M \geqslant 2$, it holds $0 \leqslant d_M(\log_2(r_M) + 1) \leqslant d_M(\log_2(r_M) + \log_2(M))$. For the "if" part, use that $r_M \geqslant 1$ and the assumption to get $0 \leqslant d_M \leqslant d_M(\log_2(r_M) + 1) = \mathcal{O}_{M\to\infty}(M^{1+h})$ so that $d_M \log_2(M) = \mathcal{O}_{M\to\infty}(M^{1+h} \log_2(M)) = \mathcal{O}_{M\to\infty}(M^{1+h})$. □

Quite naturally, ∞-encodability can be preserved under Lipschitz maps as shown in the following theorem. There is no particular difficulty in the proof: this is simply a matter of relating the covering numbers of the image to the pre-image using the Lipschitz continuity. I leave the details to the paper Gonon et al. [2023a].

Theorem 6.2.3 ([Gonon et al., 2023a])**.** *Consider the same setting as in Lemma 6.2.1. Consider also a sequence $\varphi := (\varphi_M)_{M\in\mathbb{N}_{>0}}$ of maps $\varphi_M : (\Sigma_M, \|\cdot\|_q) \to (\mathcal{F}, d)$ that are $Lips(\varphi_M)$-Lipschitz for some constants $Lips(\varphi_M) \geqslant 1$. Define $\varphi(\Sigma) := (\varphi_M(\Sigma_M))_{M\in\mathbb{N}_{>0}}$. Assume that for every $h > 0$:*

$$d_M(\log_2(r_M) + \log_2(Lips(\varphi_M)) + 1) = \mathcal{O}_{M\to\infty}(M^{1+h}). \tag{6.5}$$

Then $\varphi(\Sigma)$ is ∞-encodable.

The case of dictionaries

I now consider sequences Σ defined with dictionaries. As detailed below, results of the literature [Grohs, 2015, Thm. 5.24][Kerkyacharian and Picard, 2004, Prop. 11] use arguments that implicitly prove γ-encodability. Let me start with the case of approximation in Banach spaces as in Kerkyacharian and Picard [2004]. I only explicit the sequence used in Kerkyacharian and Picard [2004] which is γ-encodable and I do not delve into more details as results of Kerkyacharian and Picard [2004] are out of scope of this chapter. A part of the proof of [Kerkyacharian and Picard, 2004, Prop. 11] consists of implicitly showing that some specific sequence Σ^q is s-encodable, for q and s as described below in Theorem 6.2.4. In particular, the setup of Theorem 6.2.4 applies when $\mathcal{F}$ is the L^p space on $\mathbb{R}^d$ or $[0,1]^d$, $1 < p < \infty$, and the basis B is a compactly supported wavelet basis or associated wavelet-tensor product basis.

Theorem 6.2.4. *Let $\mathcal{F}$ be a Banach space with a basis $B = (e_i)_{i\in\mathbb{N}_{>0}}$ satisfying $\sup_{i\in\mathbb{N}_{>0}} \|e_i\|_{\mathcal{F}} < \infty$. Consider $p \in (0, \infty)$ and assume that B satisfies the so-called p-Telmyakov property [Kerkyacharian and Picard, 2004, Def. 2], i.e., assume that there exists $c > 0$ such that for every finite subset I of $\mathbb{N}_{>0}$ and every $(c_i)_{i\in I} \in \mathbb{R}^I$:*

$$\frac{1}{c}|I|^{1/p} \min_{i\in I} |c_i| \leqslant \|\sum_{i\in I} c_i e_i\|_{\mathcal{F}} \leqslant c|I|^{1/p} \max_{i\in I} |c_i|. \tag{6.6}$$

Consider $0 < q < p$. *For every* $M \in \mathbb{N}_{>0}$, *define*[3]:

$$\Sigma_M^q := \left\{ \sum_{i=1}^{M} c_i e_i, c_i \in \mathbb{R}, \sup_{0<\lambda<\infty} \lambda |\{i, |c_i| \geqslant \lambda\}|^{1/q} \leqslant 1 \right\}.$$

Define $s = \frac{1}{q} - \frac{1}{p}$. *Then the sequence* $\Sigma^q := (\Sigma_M^q)_{M \in \mathbb{N}_{>0}}$ *is* s*-encodable in* $\mathcal{F}$.

Proof of Theorem 6.2.4. Fix $M \in \mathbb{N}$ and $f = \sum_{i=1}^{M} c_i e_i \in \Sigma_M^q$. Consider $0 < \lambda < 1$. Define $Q_\lambda(f) := \sum_{i=1}^{M} \operatorname{sign}(c_i) \left\lfloor \frac{c_i}{\lambda} \right\rfloor \lambda e_i$ with $\operatorname{sign}(x) = 1$ if $x \geqslant 0$, -1 otherwise. It is proven in [Kerkyacharian and Picard, 2004, Prop. 6] that there exists a constant $c(p,q) > 0$ that only depends on p and q such that:

$$\|f - Q_\lambda(f)\|_{\mathcal{F}} \leqslant c(p,q) \lambda^{1-q/p} \sup_{i \in \mathbb{N}} \|e_i\|_{\mathcal{F}}.$$

Moreover, it is proven in [Kerkyacharian and Picard, 2004, Lem. 4 and proof of Prop. 11] that the family $(Q_\lambda(f))_{f \in \Sigma_M^q}$ has at most $2^{\lambda^{-q}(1-\log_2(\lambda)+\log_2(M))}$ elements. Setting $\varepsilon = \lambda^{1-q/p}$, and observing that $\lambda^{-q} = \varepsilon^{-1/s}$, this proves that the family $(Q_\lambda(f))_{f \in \Sigma_M^q}$ is a $\mathcal{O}_{\varepsilon \to 0}(\varepsilon)$-covering of Σ_M^q of size $\mathcal{O}_{\varepsilon \to 0}(2^{\varepsilon^{-1/s}(\log_2 1/\varepsilon + \log_2 M)})$, with constants independent of M. For every $M \in \mathbb{N}$, using the above result with $\varepsilon = M^{-s}$ proves that Σ^q is s-encodable. □

In the case of Hilbert spaces, much more generic sequences than Σ^q above are in fact ∞-encodable, as I now discuss. The ∞-encodability can be used to recover [Grohs, 2015, Thm. 5.24] (see Corollary 6.2.3), and to generalize Corollary 6.2.1 (see Corollary 6.2.4). Let $\mathcal{F}$ be a Hilbert space and d be the metric associated to the norm on $\mathcal{F}$. A dictionary is, by definition [Grohs, 2015, Def. 5.19], a subset $\mathcal{D} = (\varphi_i)_{i \in \mathbb{N}_{>0}}$ of $\mathcal{F}$ indexed by a countable set, which I assume to be $\mathbb{N}_{>0}$ without loss of generality. The dictionary $\mathcal{D}$ can be used to approach elements of $\mathcal{F}$ by linear combinations of a growing number M of its elements.

Theorem 6.2.5 ([Gonon et al., 2023a])**.** *Let* $\mathcal{F}$ *be a Hilbert space. Let* $\mathcal{D} = (\varphi_i)_{i \in \mathbb{N}_{>0}}$ *be a dictionary in* $\mathcal{F}$, *and* $\pi : \mathbb{N}_{>0} \to \mathbb{N}_{>0}$ *be a function with at most polynomial growth. For every* $I \subset \mathbb{N}_{>0}$, *define* $(\tilde{\varphi}_i^I)_{i \in I}$ *as any orthonormalization of* $(\varphi_i)_{i \in I}$ *(for instance we may consider the Gram-Schmidt orthonormalization). Define for every* $M \in \mathbb{N}_{>0}$ *and* $c > 0$ *(denoting* $[n] = \{1, \dots, n\}$ *for* $n \in \mathbb{N}_{>0}$*):*

$$\Sigma_M^\pi := \left\{ \sum_{i \in I} c_i \varphi_i, I \subset [\pi(M)], |I| \leqslant M, (c_i)_{i \in I} \in \mathbb{R}^I \right\},$$
$$\tilde{\Sigma}_M^{\pi,c} := \left\{ \sum_{i \in I} \tilde{c}_i \tilde{\varphi}_i^I, I \subset [\pi(M)], |I| \leqslant M, (\tilde{c}_i)_{i \in I} \in [-c,c]^I \right\}.$$

The sequence $\tilde{\Sigma}^{\pi,c} := (\tilde{\Sigma}_M^{\pi,c})_{M \in \mathbb{N}_{>0}}$ *is* ∞*-encodable in* $(\mathcal{F}, d)$, *and for every bounded set* $\mathcal{C} \subset \mathcal{F}$, *it holds:*

$$\gamma^{*approx}(\mathcal{C}|\Sigma^\pi) = \max_{c>0} \gamma^{*approx}(\mathcal{C}|\tilde{\Sigma}^{\pi,c}). \tag{6.7}$$

[3] In terms of weak-ℓ^q-space, the set Σ_M^q is simply the set of linear combinations of elements of B given by sequences $(c_i)_{i \in \mathbb{N}_{>0}}$ in the closed unit ball of $\ell^{q,\infty}(\mathbb{N}_{>0})$ with zero coordinates outside the first M ones.

In order to prove Theorem 6.2.5, U use that the set $\tilde{\Sigma}^{\pi,c}(I)$ is the image of $\varphi_{M,I} : (\tilde{c}_i)_{i\in I} \in ([-c,c]^I, \|\cdot\|_2) \mapsto \sum_{i\in I} \tilde{c}_i \tilde{\varphi}_i^I \in \mathcal{F}$, and this map is 1-Lipschitz (since $(\tilde{\varphi}_i^I)_{i\in I}$ is orthonormal). Thanks to that, I can therefore control the covering numbers of $\tilde{\Sigma}^{\pi,c}(I)$ by the ones of its pre-image in $([-c,c]^I, \|\cdot\|_2)$. The rest of the proof is purely technical, and I leave the details to the paper Gonon et al. [2023a]. As a consequence of Theorem 6.2.5, we can recover [Grohs, 2015, Thm. 5.24] as I now describe.

Corollary 6.2.3 ([Grohs, 2015, Thm. 5.24])**.** *Consider a Hilbert space $(\mathcal{F}, d)$ and $\mathcal{C} \subset \mathcal{F}$. Under the assumptions of Theorem 6.2.5, the sequence $\Sigma^\pi = (\Sigma_M^\pi)_{M\in\mathbb{N}_{>0}}$ satisfies for every relatively compact[4] set $\mathcal{C}$:*

$$\gamma^{*approx}(\mathcal{C}|\Sigma^\pi) \leqslant \gamma^{*encod}(\mathcal{C}).$$

Actually, instead of stating the previous result with the approximation speed $\gamma^{*\text{approx}}(\mathcal{C}|\Sigma)$, Theorem 5.24 in Grohs [2015] considers the following quantity [Grohs, 2015, Def. 5.23]:

$$\gamma^*(\mathcal{C}|\Sigma) := \sup\{\gamma \in \mathbb{R}, \forall f \in \mathcal{C}, \exists c > 0, \forall M \in \mathbb{N}_{>0}, d(f, \Sigma_M) \leqslant cM^{-\gamma}\},$$

which satisfies $\gamma^*(\mathcal{C}|\Sigma) \geqslant \gamma^{*\text{approx}}(\mathcal{C}|\Sigma)$ but generally differs from $\gamma^{*\text{approx}}(\mathcal{C}|\Sigma)$ since in the definition of $\gamma^{*\text{approx}}(\mathcal{C}|\Sigma)$, the implicit constant $c > 0$ is not allowed to depend on $f \in \mathcal{C}$. However, when $\mathcal{C}$ is relatively compact (that is, its closure is compact), then $c > 0$ can be chosen independently of f [Grohs, 2015, Proof of Thm 5.24] so that the two quantities coincide. The proof of Corollary 6.2.3 that can be found below is essentially a rewriting in the formalism of section 6.2 of the original proof of Theorem 5.24 in Grohs [2015]. The rewriting makes explicit the use of equality (6.7) and the ∞-encodability of the sequences $\tilde{\Sigma}^{\pi,c}$ for $c > 0$, which are only implicitly used in the original proof.

Proof. Since $\mathcal{C}$ is relatively compact, it must be bounded so Equation (6.7) of Theorem 6.2.5 holds. For every $c > 0$, Theorem 6.2.1 applied to $\tilde{\Sigma}^{\pi,c}$ of Theorem 6.2.5, which is ∞-encodable, shows that the right hand-side of Equation (6.7) is bounded from above by $\gamma^{*\text{encod}}(\mathcal{C})$. This yields the result. □

We also obtain a generic lower bound on the encoding speed of balls of approximation spaces [DeVore and Lorentz, 1993, Sec. 7.9] (also called maxisets Kerkyacharian and Picard [2000]) with general dictionaries.

Corollary 6.2.4. *Let $(\mathcal{F}, d)$ be a Hilbert space. Under the assumptions of Theorem 6.2.5, consider $\alpha, \beta > 0$ and the set[5][6] $\mathcal{A}^\alpha(\mathcal{F}, \Sigma^\pi, \beta)$ of all $f \in \mathcal{F}$ such that $\|f\| \leqslant \beta$ and $\sup_{M\geqslant 1} M^\alpha d(f, \Sigma_M) \leqslant \beta$. This set satisfies*

$$\gamma^{*encod}(\mathcal{A}^\alpha(\mathcal{F}, \Sigma^\pi, \beta)) \geqslant \alpha.$$

Corollary 6.2.4 cannot be generalized to $\mathcal{A}^\alpha(\mathcal{F}, \Sigma^\pi) := \bigcup_{\beta>0} \mathcal{A}^\alpha(\mathcal{F}, \Sigma^\pi, \beta)$: this set is homogeneous (stable by multiplication by any scalar), thus it cannot be encoded at

[4]Recall that a set is relatively compact if its closure is compact. In particular, it must be totally bounded, and in particular bounded.

[5]This is the ball of radius β of an approximation space [DeVore and Lorentz, 1993, Sec. 7.9]/maxisetKerkyacharian and Picard [2000].

[6]Note that compared to the set in Corollary 6.2.1, we additionally require that $\|f\| \leqslant \beta$ so that $\mathcal{A}^\alpha(\mathcal{F}, \Sigma^\pi, \beta)$ is a bounded set and Equation (6.7) of Theorem 6.2.5 holds.

any positive rate. Indeed, a positive encoding rate implies total boundedness of a set, whereas homogeneity implies that the set cannot be totally bounded (at least under the assumption that the metric is induced by a norm; there should, in general, be metrics with respect to which a homogeneous set may be totally bounded).

In some situations, the converse inequality $\gamma^{*\text{encod}}(\mathcal{A}^\alpha(\mathcal{F}, \Sigma^\pi, \beta)) \leqslant \alpha$ can typically be proven by studying the existence of large enough packing sets of $\mathcal{A}^\alpha(\mathcal{F}, \Sigma^\pi, \beta)$, but this falls out of the scope of this chapter. The reader can refer to [Kerkyacharian and Picard, 2004, Sec. 4] for an example.

Proof of Corollary 6.2.4. By the very definition of $\mathcal{C} := \mathcal{A}^\alpha(\mathcal{F}, \Sigma^\pi, \beta)$, this is a bounded set so Equation (6.7) of Theorem 6.2.5 holds. For every $c > 0$, Theorem 6.2.1 applied to $\tilde{\Sigma}^{\pi,c}$ of Theorem 6.2.5, which is ∞-encodable, shows that the right hand-side of Equation (6.7) is bounded from above by $\gamma^{*\text{encod}}(\mathcal{C})$, so that

$$\gamma^{*\text{encod}}(\mathcal{C}) \geqslant \max_{c>0} \gamma^{*\text{approx}}(\mathcal{C}|\tilde{\Sigma}^{\pi,c}) = \gamma^{*\text{approx}}(\mathcal{C}|\Sigma^\pi).$$

Finally, again by definition of $\mathcal{C} := \mathcal{A}^\alpha(\mathcal{F}, \Sigma^\pi, \beta)$, we have $\gamma^{*\text{approx}}(\mathcal{C}|\Sigma^\pi) \geqslant \alpha$. □

Note that if Σ^π was γ-encodable for some $\gamma > 0$ large enough then Corollary 6.2.3 would be a special case of Corollary 6.2.2 whereas Corollary 6.2.4 would be a special case of Corollary 6.2.1. But in this situation, Σ^π has no reason to be γ-encodable, whatever $\gamma > 0$ is (since the dictionary is arbitrary and the coefficients of the linear combinations are not bounded). This shows that Corollary 6.2.2 and Corollary 6.2.1 actually holds more generally for some sequences Σ that are not γ-encodable, whatever $\gamma > 0$ is, as soon as Σ can be recovered as a limit of non-decreasing sequences Σ^c, $c > 0$, that are γ-encodable, in the sense that for every $M \in \mathbb{N}_{>0}$, if $0 < c \leqslant c'$ then $\Sigma_M^c \subseteq \Sigma_M^{c'}$ and $\Sigma_M = \cup_{c>0}\Sigma_M^c$.

The case of ReLU networks

When Σ is defined with LFCNs (Definition 2.1.1), I now explicitly study how the property of ∞-encodability depends on (bounds on) the neural network sparsity, depth, and weights. In particular, Theorem 6.2.6 establishes a "simple" explicit condition under which Theorem 6.2.1 generalizes Theorem VI.4 in Elbrächter et al. [2021] to other type of constraints.

Theorem 6.2.6. *Consider an approximation family $\mathcal{N} = (\mathcal{N}_M)_{M\in\mathbb{N}_{>0}}$ where $\mathcal{N}_M$ is a set of functions realized by LFCNs (Definition 2.1.1) with at most L_M affine layers, parameters of Euclidean norm bounded by $r_M \geqslant 1$ and with at most M nonzero coordinates. Assume that for every $h > 0$, it holds:*

$$L_M M\,(1 + \log_2(r_M)) = \mathcal{O}_{M\to\infty}(M^{1+h}). \tag{6.8}$$

Then the approximation family $\mathcal{N}$ defined with such LFCNs is ∞-encodable.

The proof of Theorem 6.2.6 relies on controlling the covering numbers by taking the pre-image of the set $nnclass_M$ by $param \mapsto R_\theta$, and using that this map is Lipschitz. The details are left to the paper Gonon et al. [2023a]. As a consequence of Theorem 6.2.6, we can recover Theorem VI.4 in Elbrächter et al. [2021] as I now describe.

Example 6.2.1 (∞-encodable sequences of *sparse* neural networks - [Elbrächter et al., 2021, Thm. VI.4])**.** *Let π be a positive polynomial and consider, as in Definition VI.2 of Elbrächter et al. [2021], $\mathcal{N}_M^\pi$ the set of functions parameterized by a LFCN with weights' amplitude bounded by $\pi(M)$, depth bounded by $\pi(\log M)$ and at most M non-zero parameters. Assumption (6.8) holds since this corresponds to the case where $L_M \leqslant \pi(\log(M))$ and $1 \leqslant r_M \leqslant \max(1, \pi(M))$. Then, Theorem 6.2.6 guarantees that $\mathcal{N}^\pi := (\mathcal{N}_M^\pi)_{M\in\mathbb{N}_{>0}}$ is ∞-encodable. Given Theorem 6.2.1, the fact that $\mathcal{N}^\pi$ is ∞-encodable gives $\gamma^{*approx}(\mathcal{C}|\mathcal{N}^\pi) \leqslant \gamma^{*encod}(\mathcal{C})$ for arbitrary $p \in [1,\infty]$ and arbitrary $\mathcal{C} \subset L^p$. This is exactly Theorem VI.4 in Elbrächter et al. [2021].*

6.3 Conclusion

I now summarize the different contributions and discuss perspectives.

Approximation with quantized ReLU networks. I characterized the error of simple uniform quantization scheme Q_η that acts coordinatewise as $Q_\eta(x) = \lfloor x/\eta \rfloor \eta$. I proved in Theorem 6.1.1 that it is sufficient for the number of bits per coordinate to grow linearly with the number of layers L in order to provide ε-error in $L^\infty([-D,D]^d)$. Theorem IV.2 in Gonon et al. [2023a] shows that this number of bits is necessary for some parameters. The proof exploits a new lower-bound on the Lipschitz constant of the parameterization of LFCNs (Definition 2.1.1) established in Theorem III.1 f Gonon et al. [2023a]. The same theorem also shows a generic upper-bound for this Lipschitz constant, which generalizes upper-bounds known in specific situations. As a consequence, I gave explicit conditions on the number of bits per coordinate that guarantees quantized LFCNs to have the same approximation speeds as unquantized ones in generic L^p spaces, see Informal Theorem 6.1.1.

Notion of γ-encodability. I introduced in Definition 6.2.2 a new property of approximation families: being γ-encodable. As soon as Σ is γ-encodable in a metric space $(\mathcal{F}, d)$, Theorem 6.2.1 shows that there is a simple relation between the approximation speed of every set $\mathcal{C} \subset \mathcal{F}$ and its encoding speed:

$$\min(\gamma^{*\text{approx}}(\mathcal{C}|\Sigma), \gamma) \leqslant \gamma^{*\text{encod}}(\mathcal{C}). \tag{6.9}$$

As seen in Section 6.2.3, several classical approximation families Σ are γ-encodable for some $\gamma > 0$, including classical families defined with dictionaries or LFCNs. As a consequence, γ-encodability lays a generic framework that unifies several situations where Inequality (6.9) is known, such as when doing approximation with dictionaries [Grohs, 2015, Thm. 5.24][Kerkyacharian and Picard, 2004, Prop. 11] or LFCNs [Elbrächter et al., 2021, Thm. VI.4].

Perspectives. I discuss in Chapter 7 the perspective of considering functionally equivalent parameters when designing a quantization scheme. Functionally equivalent parameters θ, θ' are such that $R_\theta = R_{\theta'}$ (or with equality only on a given dataset). Due to the positive homogeneity of the ReLU function, we saw in Lemma 2.4.1 that there are

uncountably many equivalent parameters to θ that can be obtained by rescaling the coordinates of θ (but these are not the only ones since permuting coordinates can also lead to functionally equivalent parameters). When quantizing θ, it would be interesting to take these equivalent parameters into account. A first work that goes in that direction is given by Gribonval et al. [2023].

Chapter 7

Perspectives

In this chapter, I discuss the perspectives opened by this thesis. Let me present the outline, where I reuse some material from the conclusions of the chapters.

- **Extending to DAG networks results for LFCNs based on the path-lifting.** (Section 7.1) In this thesis, I started by extending the path-lifting and path-activations to generic ReLU networks (Chapter 2). Thanks to that, I was able to extend and prove new results for such *general ReLU networks*: Lipschitz properties (Chapter 3), pruning strategies (Chapter 3) and generalization bounds (Chapter 4). Many more can be envisioned for the future. Section 7.1 discusses two examples: the extension to DAG networks of existing results on *identifiability* [Stock and Gribonval, 2023, Bona-Pellissier et al., 2022] and on *training dynamics* [Marcotte et al., 2023], currently established only for *LFCNs*.

- **Assessing the promises of path-norm-based bounds in practice.** (Section 7.2) I have extended to *generic DAG networks* the path-norm-based generalization bounds, which is for the first time valid for widely used networks such as ResNets, VGGs etc. It is not time to assess the promises of the path-norm as a complexity measure for generalization by numerically evaluating these generalization bounds on ResNets trained on ImageNet. I do that in Section 7.2, and shed the light on remaining challenges to improve these bounds, notably by investigating average-case measures based on the path-lifting, instead of worst-case ones.

- **Adapting the Rademacher approach to practice.** (Section 7.3) I already evoked in Chapter 4 that it is not clear how the theory based on the Rademacher complexity, which is the dominant approach to statistical learning, can be strictly applied to neural networks trained in practice, because of subtle quantifier issues. In Section 7.3, I further discuss this challenge of adapting this theoretical framework to practice. And I also discuss the challenge of theoretically accounting for subtle phenomena between the training set and the generalization of the network.

- **Capitalizing on the success of the new kernel for Kronecker-sparse matrices.** (Section 7.4) We have provided with our new kernel a heuristic to decide what Kronecker sparsity pattern is efficient in practice. This paves the way to design efficient Kronecker-sparse neural networks, as discussed in Section 7.4.1. I also

discuss in Section 7.4.2 the challenge of obtaining the same gains in *half-precision* as the ones obtained with the new kernel in *float-precision*. Finally, I mention in Section 7.4 two other challenges: exploring the extension of the kernel to other hardwares, and exploring the potential benefit of the *batch-size-last* memory layout for other operations than the multiplication with a Kronecker-sparse matrix.

- **Taking into account the symmetries for quantization.** (Section 7.5) I conclude this thesis by discussing how the symmetries, and potentially the path-lifting, could be used both in practice and theory to design efficient quantization schemes.

7.1 On extending to DAG networks results for LFCNs based on the path-lifting

The extended framework maintains the key properties of path-lifting and path-activations that have proved useful in the literature and in this thesis for analyzing ReLU neural networks. These properties include capturing the piecewise affine structure of the network and being invariant to rescaling symmetries, which are essential for understanding and manipulating the behavior of ReLU neural networks. Thanks to that, I was for instance able to extend the path-norm-based generalization bounds to these general DAG networks in Chapter 4.

In this thesis, I have extended the framework of the path-lifting and the path-activations to general DAG networks, and thanks to that, I was for instance able to extend the path-norm-based generalization bounds to these general DAG networks in Chapter 4. Many other results have been established in the literature for simple LFCNs, using the path-lifting. I expect that the general framework laid by this thesis could also be used to extend these results to these more general networks. In this section, I discuss two examples: identifiability in Section 7.1.1 and training dynamics in Section 7.1.2.

7.1.1 Identifiability

The problem of identifiability is to recover the parameters of a neural network from a black-box access to the network, i.e., by querying the network with inputs and observing the outputs. This is a key challenge in the study of neural networks, with practical implications for security, privacy, intellectual property, and understanding the behavior of black-box models. For instance, if this was feasible we could reverse-engineer chatGPT, recovering its parameters, and this could pose an economic threat to such commercial models for which enormous resources have been invested in training.

There are three obvious obstructions to the identifiability of the parameters θ of the network from the mere observation of a finite number of input/output pairs $(x_i, R_\theta(x_i))_{i=1,\dots,n}$ [Stock and Gribonval, 2023], all of which are due to symmetries of the network. The first obstruction is due to rescaling symmetries: I proved in Lemma 2.4.1 that if θ is rescaled into $\lambda \diamond \theta$, then they are functionally equivalent in the sense that $R_\theta = R_{\lambda \diamond \theta}$ although they are distinct parameters for a general $\lambda \in \mathbb{R}^H_{>0}$. This means that the parameters θ can only be uniquely identified from the outputs $R_\theta(x_i)$ **up to rescaling symmetries**. The second obstruction is due to input-dead and output-dead neurons (Definition 2.4.2), for which we have the same kind of functional equivalence

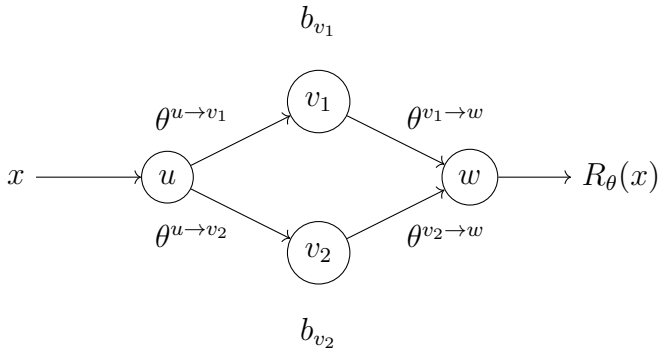


Figure 7.1: Two hidden neuron network

with distinct parameters. The third obvious obstruction is due to so-called *twin* neurons, for which we can redistribute the weights between them without affecting the function R_θ. Let me give an example of *twin* neurons (and see Stock and Gribonval [2023] for a formal definition). Consider a two-hidden-neurons network, corresponding to Figure 7.1 with associated function

$$R_\theta(x) = \sum_{i=1}^{2} \mathrm{ReLU}(x\theta^{u\to v_i} + b_{v_i})\theta^{v_i\to w}.$$

The neurons v_1 and v_2 are "twins" if $\theta^{u\to v_1} = \lambda\theta^{u\to v_2}$ and $b_{v_1} = \lambda b_{v_2}$ with $\lambda > 0$. In such a situation, neurons v_1 and v_2 have always the same activation, and it is possible to redistribute the weights between v_1 and v_2 without affecting the function R_θ. For instance, one could divide by 2 the incoming weights and bias of v_1, and multiply by 2 the ones of v_2. This would result in the same function R_θ but with different parameters θ.

Remarkably, Theorem 3 in Stock and Gribonval [2023] shows that for *LFCNs*, **these obvious obstructions are the only ones** that prevent from the identifiability from a finite set of inputs[1]. This theorem is based on the path-lifting $\Phi(\theta)$ and the simple formula $R_\theta(x) = \left\langle \Phi(\theta), A(\theta, x)\begin{pmatrix} x \\ 1 \end{pmatrix} \right\rangle$ in the scalar-valued case, and more generally on the similar formula proved in Theorem 2.4.2. Bona-Pellissier et al. [2022] further gives practical test conditions for checking whether samples $(x_i)_i$ can lead to identifiability of θ from the outputs $R_\theta(x_i)$. It is also based on the path-lifting and the path-activations. *However, both Stock and Gribonval [2023] and Bona-Pellissier et al. [2022] are limited to simple LFCNs. I expect that the results of this thesis can be used to extend these results to general DAG networks, as the main properties of the path-lifting and path-activations used to establish these results are preserved in the extension. This would pave the way for practical tests of identifiability for widely used architectures.* I now explain why this is expected to be the case.

Important properties provided by the path-lifting and path-activations for identifiability. The path-lifting $\Phi(\theta)$ complemented with the sign vectors of θ can be

[1]However, Theorem 3 in Stock and Gribonval [2023] is an existence theorem: there exists a finite set of inputs x_i such that the parameters θ can be uniquely identified from the outputs $R_\theta(x_i)$, up to the discussed obstructions.

used as a representation of the whole rescaling equivalence class of θ [Stock and Gribonval, 2023, Theorem 1], i.e., all the parameters that can be obtained from θ by neuron-wise rescaling symmetries (Definition 2.4.1). I even proved that $\Phi(\theta)$ and R_θ are invariant under symmetries due to input-dead neurons, and evoked (but did not prove) that this is also the case for output-dead neurons. This means that $(\Phi(\theta), \mathrm{sgn}(\theta))$ is also expected to be a representation of the parameters that absorbs the first two obstructions: rescaling symmetries and dead neurons.

Moreover, the formula

$$R_\theta(x) = \left\langle \Phi(\theta), A(\theta, x) \begin{pmatrix} x \\ 1 \end{pmatrix} \right\rangle$$

shows that if there are enough points x with different activations $A(\theta, x)$, then the vector $\Phi(\theta)$ can be uniquely identified from the output $R_\theta(x)$ [Stock and Gribonval, 2023]. This is the logic behind Theorem 3 in Stock and Gribonval [2023]: first prove that $\Phi(\theta)$ is identifiable from the outputs $R_\theta(x_i)$ by choosing points x_i with different activations $A(\theta, x_i)$, and then prove that θ is identifiable from $\Phi(\theta)$ up to the obstructions above.

Open research directions provided by this thesis. I expect that the results from Stock and Gribonval [2023] and Bona-Pellissier et al. [2022] could be generalized to general DAG networks (Definition 2.2.2) by using the extended framework on path-lifting and path-activations presented in this thesis.

7.1.2 Training dynamics

In many cases, training is done in overparameterized settings, corresponding to many more parameters than training samples[2]. For example, the smallest ResNet architecture has 11.5M parameters [He et al., 2016], and ImageNet has 1.2M images. In overparameterized settings, the optimization algorithm often reaches one of the many zero training error solutions [Zhang et al., 2021]. However, the generalization error can be very different between these solutions [Zhang et al., 2021]. To explain the success of neural networks, it is believed that the training algorithm has a bias towards some of these solutions that generalize well, called "implicit bias". A better understanding of the training dynamics is key to understanding the implicit bias of the training algorithm.

In practice, the most usual way to train the parameters is to initialize them randomly $\theta(0) = \theta_{\text{init}}$, and update them using gradient descent:

$$\theta(t+1) = \theta(t) - \eta \nabla_\theta L(\theta(t)),$$

where $\eta > 0$ is a so-called learning rate and $L(\theta)$ is the training error introduced in Section 4.1:

$$L(\theta) = \frac{1}{n} \sum_{i=1}^{n} \ell(R_\theta(x_i), y_i)$$

for a training set $(x_i, y_i)_{i=1,\dots,n}$ and a loss function ℓ. The trajectory $t \mapsto \theta(t)$ is called the training dynamic. Since the training error $L(\theta)$ is often non-convex in the parameters θ, this makes the trajectory $t \mapsto \theta(t)$ challenging to analyze.

[2]This is not the case for large language models: GPT-3 has 175B parameters and was trained on 300 Billion tokens [Chuan Li, 2021].

A form of "weak" implicit bias consists of the conserved quantities of the training dynamics: they trap the trajectory $t \mapsto \theta(t)$ in a low-dimensional manifold. Emmy Noether's famous theorem states that every symmetry of a physical system leads to a conservation law [Noether, 1918]. In the context of neural networks, the loss function $\theta \mapsto L(\theta)$ governs the training dynamic. It has been shown that any one-parameter transformation group that leaves the training error invariant leads to a conserved quantity [Kunin et al., 2021, Theorem 1]. Because of the definition of the training error, we see that if parameters θ and θ' are functionally equivalent ($R_\theta = R_{\theta'}$), then $L(\theta) = L(\theta')$. Therefore, the training error inherits from the same symmetries of the network R_θ. Thus, the symmetries of the network R_θ lead to conserved quantities of the training dynamic.

Important properties provided by the path-lifting for training dynamics. For *simple LFCNs*, Theorem 1 in Stock and Gribonval [2023] shows that the vector $\Phi(\theta)$ locally captures the neuron-wise rescaling symmetries of R_θ, meaning that for every[3] θ, there is a neighborhood Ω of θ such that for every $\theta' \in \Omega$, if $\Phi(\theta') = \Phi(\theta)$ then θ' can be deduced from θ by neuron-wise rescaling symmetries (we have already mentioned that the converse is always true, see Theorem 2.4.1). This shows that locally, $\Phi(\theta)$ is a representative of all parameters that are rescaling equivalent to θ. Therefore, it is expected that the local image of Φ is related to the conserved quantities of the training dynamics. To go in this direction, Theorem 2.14 of Marcotte et al. [2023] shows for one-hidden-layer LFCNs and linear networks (corresponding to LFCNs with the ReLU replaced by the identity) that a function $h : \theta \in \mathbb{R}^d \mapsto \mathbb{R}$ is locally conserved[4] around θ by the gradient flow (the time-continuous version of the gradient descent algorithm) if and only if $J\Phi(\theta)\nabla h(\theta) = 0$ where $J\Phi(\theta)$ is the Jacobian matrix of the path-lifting Φ at θ.

This basically follows from the simple formula $R_\theta(x) = \left\langle \Phi(\theta), A(\theta, x) \begin{pmatrix} x \\ 1 \end{pmatrix} \right\rangle$ in the scalar-valued case, and more generally from the similar formula proved in Theorem 2.4.2, in addition to the fact that the gradient flow is given by $\dot{\theta} = -\nabla_\theta L(\theta)$. Indeed, at least informally (skipping differentiability assumptions etc.), if h is conserved then by differentiating $h(\theta(t)) = h(\theta(0))$, we get $\left\langle \nabla h(\theta(t)), \dot{\theta}(t) \right\rangle = 0$ for every t. Denote by $S = (x_i, y_i)_{i=1}^n$ the training samples, and consider the function f defined by $f(\varphi, S) = \frac{1}{n}\sum_{i=1}^n \ell(\left\langle \varphi, A(\theta, x_i) \begin{pmatrix} x_i \\ 1 \end{pmatrix} \right\rangle, y_i)$ in the scalar-valued case (the same reasoning can be done in general using similar quantities building upon Theorem 2.4.2). We have $L(\theta) = f(\Phi(\theta), S)$, and therefore $\nabla_\theta \ell(\theta) = J\Phi(\theta)^T \nabla_\varphi f(\Phi(\theta), S)$. Since $\dot{\theta}(t) = -\nabla_\theta \ell(\theta(t)) = -J\Phi(\theta)^T \nabla_\varphi f(\Phi(\theta), S)$ we deduce that

$$\left\langle \nabla h(\theta(t)), J\Phi(\theta(t))^T \nabla_\varphi f(\Phi(\theta(t)), S) \right\rangle = 0, \forall t.$$

This is equivalent to

$$\langle J\Phi(\theta(t))\nabla h(\theta(t)), \nabla_\varphi f(\Phi(\theta(t)), S) \rangle = 0, \forall t.$$

[3] Except for a Lebesgue negligeable set.

[4] Meaning: there is a neighborhood of θ such that for any initialization in this neighborhood, any training set $(x_i, y_i)_i$, the maximal solution $t \mapsto \theta(t)$ given by the Cauchy-Lipschitz theorem is such that $h(\theta(t)) = h(\theta(0))$ for any t where this maximal solution is defined. Simply said, h is conserved by the gradient flow for any training datasets and any initialization around θ.

To get Theorem 2.14 in Marcotte et al. [2023], it is sufficient to check that the span of $\{\nabla_\varphi f(\Phi(\theta(t)), S), t\}$ is the whole ambient space. Indeed, this would imply $J\Phi(\theta(t))\nabla h(\theta(t)) = 0$ for every t. The span condition is checked at hand in Marcotte et al. [2023] for one-hidden-layer LFCNs and linear networks, with usual loss functions.

This further allows Marcotte et al. [2023] to count the number of (linearly independent) conserved quantities, by studying the rank of the matrix $J\Phi(\theta)$. They also give a code to explicitly compute the polynomial conserved quantities. This code explicitly manipulates $\Phi(\theta)$.

Open research directions provided by this thesis. I expect that the results of Marcotte et al. [2023] could be further extended to general DAG networks by using the extended framework on path-lifting and path-activations presented in this thesis. This would pave the way to practical computations of conserved quantities for widely used ReLU networks.

I now turn to concrete numerical evaluations of the path-norm-based bounds established in this thesis on ResNets trained on ImageNet.

7.2 On challenging the promises of path-norm-based bounds in practice

This thesis made available the first generalization guarantees based on the path-norm for general DAG networks, including ResNets, VGGs, U-nets, ReLU MobileNets, Inception nets, AlexNet, AlphaGo, and AlphaZero. This is the opportunity to assess the current state of the gap between theory and practice, and to diagnose without concession possible room for improvements. I released a code that allows anyone to easily experiment on their favorite network and see if the promises of the path-norm-based bounds are held. I recall that the code is available at `github.com/agonon/pathnorm_toolkit`.

As a concrete example, Section 7.2.1 demonstrates that on ResNet18 trained on ImageNet: 1) the generalization bound provided by Theorem 4.4.1 can be numerically computed; 2) for a (dense) ResNet18 trained in a standard way, roughly *30 orders of magnitude* would need to be gained for this path-norm-based bound to match practically observed generalization gap; 3) the same bound evaluated on a *sparse* ResNet18 (trained with standard sparsification techniques) is decreased by up to 13 orders of magnitude. Section 7.2.2 will discuss promising leads to reduce this gap, in particular going beyond worst-case measures to average-case measures based on the path-lifting.

7.2.1 Experiments

When would a generalization bound be informative? For ResNets trained on ImageNet, the training error associated with cross-entropy is typically between 1 and 2, and the top-1 training error is typically less than 0.30. The same orders of magnitude apply to the empirical generalization gap. To ensure that the expected risk (either for cross-entropy or top-1 accuracy) is of the same order as the training error, *a bound on the generalization gap should basically be of order at most* 1.

For parameters θ learned from training data, Theorem 4.4.1 and Theorem 4.1.1 allow us to bound the expected risk in terms of a performance measure (that depends on a free choice of $\gamma > 0$ for the top-1 accuracy, see Theorem 4.1.2) on the training data, plus a term depending on the Rademacher complexity, this term being bounded by $\frac{4\sigma}{n} C \times L \times \|\Phi(\theta)\|_1$. The Lipschitz constant L is $\sqrt{2}$ for cross-entropy, and $2/\gamma$ for the top-1 accuracy.

Evaluation of $\frac{4\sigma}{n}C$ for ResNets on ImageNet. I further bound σ/n by $B/\sqrt{n}$, where $B \simeq 2.6$ is the maximum L^∞-norm of the images of ImageNet normalized for inference. We at most lose a factor B compared to the bound directly involving σ since it also holds $\sigma/n \geqslant 1/\sqrt{n}$ by definition of σ. I train on 99% of ImageNet so that $n = 1268355$. Moreover, recall that $C = (D \log((3+2P)K) + \log(\frac{3+2P}{1+P}(d_{\text{in}}+1)d_{\text{out}}))^{1/2}$. For ResNets, $P = 1$ (as there are only classical max-pooling neurons, corresponding to k-max-pooling with $k = 1$), the kernel size is $K = 9$, $d_{\text{in}} = 224 \times 224 \times 3$, $d_{\text{out}} = 1000$, and the depth is $D = 2 + \#$ basic blocks $\times$ # conv per basic block, with the different values available in Appendix E.1. The values for $4BC/\sqrt{n}$ are reported in Table 7.1. Given these results and the values of the Lipschitz constant L, on ResNet18, *the bound would be informative only when* $\|\Phi(\theta)\|_1 \lesssim 10$ *or* $\|\Phi(\theta)\|_1/\gamma \lesssim 10$ respectively for the cross-entropy and the top-1 accuracy.

Table 7.1: Numerical evaluations on ResNets and ImageNet1k with 2 significant digits. Multiplying by the Lipschitz constant L of the loss and the path-norm gives the bound in Theorem 4.4.1. The second line reports the values when the analysis is sharpened for max-pooling neurons, see Remark 4.4.1.

ResNet	18	34	50	101	152
$\frac{4}{\sqrt{n}}CB =$	0.088	0.11	0.14	0.19	0.23
$\frac{4}{\sqrt{n}}C_{\text{sharpened}}B =$	0.060	0.072	0.082	0.11	0.13

I now compute the path-norms of trained ResNets, both dense and sparse, using the simple formula proved in Theorem 3.1.1.

The ℓ^1-path-norm of pretrained ResNets is 30 orders of magnitude too large. Table 7.2 shows that the ℓ^1-path-norm is 30 orders of magnitude too large to make the bound informative for the cross-entropy loss. The choice of γ is discussed in Appendix E.1, where I observe that there is no possible choice that leads to an informative bound for top-1 accuracy in this situation.

Table 7.2: Path-norms of pretrained ResNets available on PyTorch, computed in float32.

ResNet	18	34	50	101	152
$\|\Phi(\theta)\|_1$	1.3×10^{30}	overflow	overflow	overflow	overflow
$\|\Phi(\theta)\|_2$	2.5×10^{2}	1.1×10^{2}	2.0×10^{8}	2.9×10^{9}	8.9×10^{10}
$\|\Phi(\theta)\|_4$	7.2×10^{-6}	4.9×10^{-6}	6.7×10^{-4}	3.0×10^{-4}	1.5×10^{-4}

Sparse ResNets can decrease the bounds by 13 orders of magnitude. I just showed that pretrained ResNets have very large ℓ^1-path-norm. Does every network with a good test top-1 accuracy (i.e., expected risk) have such a large ℓ^1-path-norm? Since any zero in the parameters θ leads to many zero coordinates in $\Phi(\theta)$, I now investigate whether sparse versions of ResNet18 trained on ImageNet have a smaller path-norm. Sparse networks are obtained with iterative magnitude pruning plus rewinding, with hyperparameters similar to the one in Frankle et al. [2021, Appendix A.3]. Results show that the ℓ^1-path-norm decreases from $\simeq 10^{30}$ for the dense network to $\simeq 10^{17}$ after 19 pruning iterations, basically losing between a half and one order of magnitude per pruning iteration. Moreover, the test top-1 accuracy is better than with the dense network for the first 11 pruning iterations, and after 19 iterations, the test top-1 accuracy is still way better than what would be obtained by guessing at random, so this is still a non-trivial matter to bound the generalization error for the last iteration. Details are in Appendix E.1. This shows that there are indeed practically trainable networks with much smaller ℓ^1-path-norm that perform well. It remains open whether alternative training techniques, possibly with path-norm regularization, could lead to networks combining good performance and informative generalization bounds.

Additional observations: increased depth and train size. In practice, increasing the size of the network (that is, the number of parameters) or the number of training samples can improve generalization. I can, again, assess for the first time whether the bounds based on path-norms follow the same trend for standard modern networks. Table 7.2 shows that path-norms of pretrained ResNets available on PyTorch roughly increase with depth. This complements Dziugaite et al. [2020, Figure 1], which shows that for simple *LFCNs*, path-norm, despite being one of the best *overall* weight-based measures for correlating with the generalization gap, struggles the most to maintain positive correlation with *depth* variation, compared to other hyperparameters (width, weight decay, etc.). For increasing training sizes, I did not observe a clear trend for the ℓ^1-path-norm, which seems to mildly evolve with the number of epochs rather than with the train size, see Appendix E.1 for details.

7.2.2 Open research directions provided by this thesis

Theorem 4.4.1 is the first generalization bound valid for such networks based on path-norm. This bound recovers or beats the sharpest known ones of the same type. Its ease of computation led in Section 7.2.1 to the first experiments on widely used networks. However, in spite of all the reasons to consider the path-norm as a complexity measure for generalization, we observed a gap between theory and practice of roughly 30 orders of magnitude for a dense version of ResNet18 trained with standard tools on ImageNet. Building on the theoretical and experimental results of this thesis, I now dress a list of promising lines of research to close this gap.

- **The most promising lead is probably to go beyond worst-case, investigating average-case complexity measures based on the path-lifting.** In this thesis, I have extended the *path-lifting* to general DAG networks, while ensuring that it fulfills several of its promises that I anticipated as the result of previous works

in the simple cases of LFCNs: it remains invariant to rescaling symmetries, it captures with the path-activations the piecewise structure of the network, and its mixed norms are easy to compute. I further showed that it is indeed possible to use these key properties to establish generalization bounds based on the path-lifting for such general DAG networks. Remember that before this, the analyses using the path-lifting were limited to scalar-valued LFCNs. Thanks to all this development, I was able to challenge the current best bound based on the path-norm (Theorem 4.4.1), and we now know that the ℓ^1-path-norm is not yet the good complexity measure to explain generalization in over-parameterized training settings, where there are too few training samples compared to the expressivity of the network's architecture. **In short, there are good reasons to believe that the path-lifting provides a good representation of the parameters to analyze the generalization properties of neural networks, but its ℓ^1-norm is not the ultimate way to measure the complexity of the network.** The next exciting challenge is to find a more refined complexity measure based on the path-lifting that would remain computable, while being informative even in over-parameterized regimes. For that, the most promising lead is probably to go beyond the worst-case measure $\|\Phi(\theta)\|_1$ and to investigate average-case complexity measures based on the path-lifting. First, let me detail why $\|\Phi(\theta)\|_1$ is worst-case.

The worst-case analysis of the path-activations. Let me recall the discussion we had in Section 2.4.2 about the formula

$$R_\theta(x) = \left\langle A(\theta, x)^T \Phi(\theta), \begin{pmatrix} x \\ 1 \end{pmatrix} \right\rangle$$

for the scalar-valued case, and the similar formula in the general case (Equation (2.6)). On each of the regions in (θ, x) where the path-activations $A(\theta, x)$ are constant, this formula says that the output of the network is affine in x, with the affine coefficients given by $A(\theta, x)^T \Phi(\theta)$: the matrix A filters the affine coefficients stored in $\Phi(\theta)$ to only keep the ones relevant on the current region. Conceptually, the ℓ^1-path-norm arises in the generalization bounds because of a type of control as follows:

$$\begin{aligned} R_\theta(x) &\underset{\text{Theorem 2.4.2}}{=} \left\langle A(\theta, x)^T \Phi(\theta), \begin{pmatrix} x \\ 1 \end{pmatrix} \right\rangle \\ &\underset{\text{Hölder}}{\leqslant} \max(\|x\|_\infty, 1) \|A(\theta, x)^T \Phi(\theta)\|_1 \\ &\underset{\text{Lemma 3.3.1}}{\leqslant} \max(\|x\|_\infty, 1) \|\Phi(\theta)\|_1. \end{aligned}$$

For $\|A(\theta, x)^T \Phi(\theta)\|_1 \leqslant \|\Phi(\theta)\|_1$, I used the *worst-case analysis that says that all the paths are activated simultaneously* (Lemma 3.3.1). Let me go step by step to the last inequality involving the ℓ^1-path-norm to understand what is missing in this complexity measure. Already for $\|A(\theta, x)^T \Phi(\theta)\|_1$, the affine coefficients (coordinates of Φ) that are activated simultaneously but that cancel each other are summed up in $\|A(\theta, x)^T \Phi(\theta)\|_1$ *in absolute value*. This already does not seem desirable. For instance with an iid initialization, all these affine coefficients would cancel each other but $\|A(\theta, x)^T \Phi(\theta)\|_1$ would be very large as it would sum them in

absolute value. In the meantime, the generalization gap at initialization is equal to zero since the parameters are yet independent from the training samples. This shows that $\|A(\theta, x)^T \Phi(\theta)\|_1$ is not suited to explain generalization at least at initialization. This is even worse with the ℓ^1-path-norm, where this time *all the paths are assumed to be active simultaneously*. While this can be the case for specific[5] parameters θ and inputs x, this is not the case in general. Therefore, we can say that in a sense, the ℓ^1-path-norm that arises from the worst pattern of path-activations correspond to a worst-case on both the parameters and the inputs. Finally, let me also note that the bound $\|A(\theta, x)^T \Phi(\theta)\|_1 \leqslant \|\Phi(\theta)\|_1$ completely decorrelates the parameters θ from the inputs x, which is once again not desirable as the generalization gap precisely depends on the relation between the parameters and the input distribution.

Going average-case. An important challenge for the future would be to replace the ℓ^1-norm $\|\Phi(\theta)\|_1$ by a more refined quantity that would:

- depend on the expected path-activations over the input distribution, and not only on their worst-case value;
- be invariant to symmetries of the network (which is very likely if it is based on the path-lifting and the path-activations);
- be related to Lipschitz properties of the network, and hence be a good complexity measure for generalization;
- be computable in practice.

Such a complexity measure should be calibrated in simple situations where we roughly know the generalization gap. Possible candidates for the first three points may be derived from quantities of the type:

$$\mathbb{E}_x \|A(\theta, x)^T \Phi(\theta)\|_1.$$

For the fourth point, even if it does not seem evident to compute this exactly, it is at least possible to envision approximation with Monte-Carlo methods. This complexity measure would already be a huge step forward (keeping in mind that it still has non-desirable aspects as discussed above). Indeed, it would replace the worst-case analysis of the path-activations by an average-case analysis, which is likely to be much more informative, and it would restablish the relation between the parameters and the input distribution. While I haven't seen this type of complexity measure proposed before in the literature, I can already note that it is related to expected Lipschitz bounds in x for general DAG networks, as can be seen by differentiating the function $x \mapsto \mathbb{E}_{x'} \|R_\theta(x) - R_\theta(x')\|_1$ (fixed θ), using the formula proved in Theorem 2.4.2. Moreover, the *expected ℓ^1-path-norm* $\mathbb{E}_x \|A(\theta, x)^T \Phi(\theta)\|_1$ makes also way more sense *geometrically* than the ℓ^1-path-norm $\|\Phi(\theta)\|_1$. The coordinates of $\Phi(\theta)$ are the affine coefficients, that can be filtered by the path-activations and summed up to get the slopes of R_θ on the different region where R_θ is affine (Theorem 2.4.2). Therefore, $\|\Phi(\theta)\|_1$ is the sum of all the affine coefficients in absolue value, resulting in a worst-case uniform bound for all the regions' slopes. However, $\mathbb{E}_x \|A(\theta, x)^T \Phi(\theta)\|_1$ *filters the coefficients on each region*, unfortunately still sum

[5] For instance, consider positive parameters, positive input, and only ReLU neurons.

them in absolute value, but then *weights these sums by the probability of falling into these regions*. In particular, regions with high slopes but low probability of being activated would not contribute much to the complexity measure, something desirable to explain generalization in over-parameterized regimes where typical estimators have large Lipschitz constants to interpolate the training data, but only in small regions of the input space.

- **Consider other norms than the ℓ^1-path-norm.** Possible bounds involving the ℓ^q-path-norm for $q > 1$ deserve a particular attention, since numerical evaluations show that they are several orders of magnitude below the ℓ^1-norm. The current challenge is to make ℓ^q-path-norm arise in a non-combinatorial way. For now, the unique presence of the ℓ^1-path-norm is the result of the worst-case analysis on the path-activations discussed above (see also Lemma 3.3.1 and below). If we do the same worst-case analysis but with the ℓ^q-norm for $q > 1$, a constant depending on the number of the paths shows up in front of the ℓ^q-path-norm. Therefore, to find a non-combinatorial way to use the ℓ^q-path-norms also probably relies on going beyond the worst-case analysis of the path-activations, and to consider an average-case analysis. In the same vein as above, we could consider the quantity $\mathbb{E}_x \|A(\theta, x)^T \Phi(\theta)\|_q$ for $q > 1$.

- **Sparsity regularization.** Without changing the bound of Theorem 4.4.1, sparsity seems promising to reduce the ℓ^1-path-norm by several orders of magnitude without changing the performance of the network.

- **Take more finely into account weight-sharing into the framework.** Weight-sharing reduces the degrees of freedom of the problem. For the Rademacher bound of Theorem 4.3.1 based on covering numbers, I proved that it was possible to account for weight-sharing by reducing a term that was dependent on the number of parameters to the actual degrees of freedom when there is weight-sharing. While the more refined Rademacher bound I proved in Theorem 4.4.1 got rid of this undesirable term, the question remains open whether a more refined analysis complexity measure than the path-norm could lead to sharpened analysis in the presence of weight-sharing [Pitas et al., 2019, Galanti et al., 2023].

- **Remove in the path-norm excess paths related to max-pooling neurons.** A k-max-pooling neuron with kernel size K only activates $1/K$ of the paths, but the bound sums the coordinates of the path-lifting Φ related to these K paths. This may lead to a bound K times too large in general (or even more in the presence of multiple max-pooling layers).

Besides all these promising leads to find a better complexity measure based on the path-lifting than its ℓ^1-norm, there are also challenges concerning the general framework of statistical learning theory, as I now discuss.

7.3 On adapting the dominant statistical learning theory to modern practices

Modern practices in deep learning have raised important and exciting challenges in many fields of theory. In this section, I would like to discuss two such challenges that call for an adaptation of the dominant statistical learning theory.

In Chapter 4, I presented the dominant approach to the statistical learning theory: bound the estimation error and generalization gap with the Rademacher complexity [Shalev-Shwartz and Ben-David, 2014, Goodfellow et al., 2016, Bach, 2024]. This has proved to be a powerful tool to better understand the generalization properties of many models of interests [Bach, 2017, 2024] and to relate various complexity measures to the generalization gap [Neyshabur et al., 2015, Kawaguchi et al., 2017, Barron and Klusowski, 2019, Pérez and Louis, 2020, Jiang et al., 2020, Dziugaite et al., 2020].

As explained in Chapter 4, the basic block of this approach is Theorem 4.1.1. It essentially says that for any nice enough loss function ℓ, any set $\mathcal{F}$ of functions and any estimator $\hat{f}$ that takes a training set S and returns $\hat{f}(S) \in \mathcal{F}$, the Rademacher complexity of $\mathcal{F}$ bounds the generalization gap and estimation error.

The first challenge caused by practice is that it is not clear, when observing a network $\hat{f}(S)$ in practice, to which function set $\mathcal{F}$ this result should be applied. I will discuss this in Section 7.3.1.

The second and last challenge I would like to mention in this section is on being able to theoretically take into account some very subtle aspects of the data distribution that practically impact the generalization properties. I will discuss this in Section 7.3.2.

7.3.1 On applying the Rademacher bound in practice

A main obstacle to strictly apply the Rademacher bound in practice (Theorem 4.1.1) is the problem of identifying the set $\mathcal{F}$ in which is our estimator. In practice, we fix an algorithm (such as gradient descent with fixed so-called hyperparameters), and given n training samples $S = (x_i, y_i)_{i=1}^n$, we run the algorithm and get some $\hat{f}(S)$. This defines our estimator $\hat{f}$. However, we can only observe the output of our estimator (i.e., algorithm) for a finite number of possible training sets S. For the Rademacher bound, we obviously would like to know the smallest $\mathcal{F}$ that contains all the possible outputs $\hat{f}(S)$ of our estimator $\hat{f}$: the smaller the set $\mathcal{F}$, the smaller its Rademacher complexity. Actually, even before trying to identify a small $\mathcal{F}$ that contains $\hat{f}(S)$ *for all* S, it is already not clear how to identify *one such* $\mathcal{F}$. Indeed, we only observe $\hat{f}(S)$ for a single S (or a finite number when training several times to do so-called *cross-validation*), so without an explicit regularization at the end of the algorithm, it is not clear to know what kind of property will have $\hat{f}(S)$ for general S's.

For instance, when I evaluated the path-norm-based bound (Theorem 4.4.1) on ResNet18 training set S of ImageNet, I observed the ℓ^1-path-norm of this specific $\hat{f}(S)$, and defined $\mathcal{F}$ to be all the network functions with path-norm smaller than this specific ℓ^1-path-norm. However, *there is no guarantee that for over training sets S', the same training algorithm would have led to $\hat{f}(S')$ in this choice of $\mathcal{F}$, as it would require $\hat{f}(S')$ to have a smaller path-norm than the one of $\hat{f}(S)$*. It is therefore an open challenge to adapt the statistical learning theory to this practical setting where we only observe the output $\hat{f}(S)$ of the

estimator $\hat{f}$ for a finite number of training sets S, and for which we have to find a set $\mathcal{F}$ that contains all the possible outputs of $\hat{f}$.

7.3.2 On subtle practical impacts of the data distribution

The classical approach in statistical learning theory is to bound the *expected* generalization gap or estimation error over all possible training sets S, and then deduce high-confidence intervals for many S's using concentration inequalities [Boucheron et al., 2013, Shalev-Shwartz and Ben-David, 2014, Bach, 2024]. This essentially hides the belief that for training samples $S = (x_i, y_i)_{i=1}^n$ with n large, most of the *typical* samples S are likely to lead to similar generalization properties. However, this is not always the case in practice, and the generalization gap can be very sensitive to the choice of the training set S, even for extremely large n. This is exemplified with large language models (LLMs) where enormous amounts of data are used, but subtle modifications in the training set can have a huge impact on the generalization properties of the model [Scao et al., 2022, Section 3, Table 1][Penedo et al., 2023]. For instance, small repetitions (or near duplications) of samples in the training set can cause huge drop of the empirical risk [Hernandez et al., 2022]. This kind of sensitivity to the training set is not yet captured by the classical statistical learning theory, and it is an open challenge to find the good framework to do so.

7.4 On extending to other settings the success of the new kernel for Kronecker-sparse matrix multiplication

The advances made by our new kernel for Kronecker-sparse matrix multiplication raise several interesting and challenging questions for the future. In Section 7.4.1, I discuss how our heuristic paves the way to new research directions to design efficient Kronecker-sparse neural networks. In Section 7.4.2, I discuss what are the challenges to obtain the same gains in *half-precision*. I also mention two additional challenges raised by the practical use of the new kernel in Section 7.4.3.

7.4.1 On designing efficient Kronecker-sparse neural networks

In the literature, Dao et al. [2022a] are among the best reported results in terms of accuracy when replacing dense matrices $\mathbf{W}$ by a product of Kronecker-sparse matrices $\mathbf{W} = \mathbf{K}_1 \dots \mathbf{K}_L$ in neural networks. They use $L = 2$ Kronecker-sparse matrices, and choose the sparsity patterns $\pi_1 = (a_1, b_1, c_1, d_1)$ and $\pi_2 = (a_2, b_2, c_2, d_2)$ of $\mathbf{K}_1$ and $\mathbf{K}_2$ such that *the product of their supports is dense*, motivated by expressivity concerns. We followed the same protocol when making experiments in Section 5.5 to show that the new kernel accelerates the inference of neural networks using Kronecker-sparse matrices.

However, besides the criteria of expressivity to hope for a good accuracy, the choice of the sparsity patterns π_1 and π_2 is also crucial *to get the best performance in time and energy*. To the best of my knowledge, we provided in Chapter 5 the first heuristic to choose Kronecker sparsity patterns (a, b, c, d) that are particularly efficient for Kronecker-sparse

matrix multiplication, with the hope that it will avoid extensive empirical search. Indeed, we demonstrated in Chapter 5 that the new kernel is particularly efficient for Kronecker-sparse matrices of sparsity pattern (a, b, c, d) with large ratios $(b + c)/bc$ and $d(b + c)/bc$. This paves the way to new research directions to design efficient Kronecker-sparse neural networks.

Open research directions provided by this thesis. Consider the problem of choosing a number of factors L, and Kronecker sparsity patterns $\pi_1, \ldots, \pi_L$ in order to maximize both the accuracy and the gains in time and energy, when replacing a $\mathbf{W}$ by a product of Kronecker-sparse matrices $\mathbf{W} = \mathbf{K}_1 \ldots \mathbf{K}_L$ in neural networks. This problem is combinatorial given the number of patterns, and it is important to design grounded theoretical rules to restrict the search. The expressivity criterion of Dao et al. [2022a] is a first step to restrict the search to sparsity patterns $(\pi_\ell)_{\ell=1}^L$ such that the product of associated supports is dense. The new heuristic we provided in Chapter 5 is a second important step: among the all the patterns $\pi_\ell = (a_\ell, b_\ell, c_\ell, d_\ell)$ satisfying the expressivity criterion, we should now aim to maximize each ratio $(b_\ell + c_\ell)/b_\ell c_\ell$ and $d_\ell(b_\ell + c_\ell)/b_\ell c_\ell$. This raises several questions for the future. How to maximize each of these ratios? Should we maximize them independently, or is there a trade-off between them? Is increasing the number L of factors beneficial? Increasing L may increase the total time because of the sequential overhead of performing each multiplication one at a time, but it may also leave more room for choosing efficient sparsity patterns that satisfy the expressivity constraint.

I now turn to the challenge of obtaining the same gains in *half-precision*.

7.4.2 On obtaining the same gains in *half-precision*

The new kernel we proposed in Chapter 5 is particularly efficient in *float-precision* and uses the data type *float4* that stores 4 32-bit floating-point numbers (floats), and for which the read-and-write operations are *atomic*, i.e., the four elements are read and written at once, allowing for efficient memory access. This is only possible because the hardware (the NVIDIA GPU) supports atomic operations on this kind of data type. In CUDA, this data type can be conceptualized as follows (but with hardware support for atomic operations):

```
1    typedef struct {
2        float x, y, z, w;
3    } float4;
4
5    // Example usage:
6    float4 a = {1.0f, 2.0f, 3.0f, 4.0f};
```

However, the gains in time and energy are not as good in *half-precision* (Appendix D.2.4). This is mainly due to the absence of native support for a vectorized data type *half4*, similar to *float4* but for 16-bit floating-point numbers (halfs), in NVIDIA GPUs. The NVIDIA GPUs only support *half2* operations in *half-precision*, corresponding to vectors of 2 halfs and for which the read-and-write operations are atomics. This means we can only read and write at once 2 consecutive elements of the matrix, which is less efficient.

Open research directions provided by this thesis. As long as NVIDIA does not support *half4* operations in CUDA, the challenge is to design a new kernel that is as efficient in *half-precision* as the new kernel is in *float-precision*. For now, the kernel does not rely on any opaque NVIDIA routines, and is based on simple operations such as memory reads and writes, and explicit multiplications between the elements. This has the advantage of making the kernel easy to understand and to modify. In the future, it could be interesting to investigate the use of opaque routines in *half-precision*, such as *TensorCores* that are able to perform matrix multiplications of small 16×16 matrices in *half-precision* with a very high throughput. However, the use of *TensorCores* is not straightforward as it requires to first extract 16×16 dense sub-matrices from the Kronecker-sparse matrices, and it is not clear how to achieve that efficiently. This is an open challenge for the future.

7.4.3 Additional challenges raised by the new kernel

I would like to conclude this section by mentioning two additional challenges raised by the practical use of the new kernel.

First, the new kernel has been shown to be particularly performant in *batch-size-last*, which corresponds to a specific assumption on the memory layout of the input and output matrices. However, this *batch-size-last* version of the kernel cannot be used as it is in PyTorch, as PyTorch does the inverse assumption on the memory layout. The current PyTorch standard is *batch-size-first*, and this seems to be essentially because of a historical reason rather than a technical one. It would be interesting to investigate the possibility to implement other PyTorch operations in *batch-size-last*, to use the new kernel also in *batch-size-last*, but also to see if this could lead to further gains in time and energy for other operations. For instance, we saw that the *already existing generic algorithm for sparse matrix multiplication* in Pytorch *is accelerated by a factor of 7* when performed in *batch-size-last*, see Section 5.4.

Second, since our kernel has been shown to improve the existing implementations on NVIDIA GPUs, it would be interesting to see if it can also be adapted to other hardware. For instance, the translation of our kernel into OpenCL could enable it to run on AMD hardware and other platforms.

7.5 On challenges in approximation guarantees for quantized networks

In this thesis, I provided new results on the approximation power of quantized networks, i.e., networks with weights constrained to be in a finite set (e.g., floats). For instance, I characterized the error of simple uniform quantization scheme Q_η that acts coordinate-wise as $Q_\eta(x) = \lfloor x/\eta \rfloor \, \eta$ in Theorem 6.1.1, where I improved the known sufficient number of bits per coordinate for Q_η to provide an ε-quantization-error in $L^\infty([-D, D]^d)$.

However, and as discussed in Section 6.1, I did this work before being familiar with the path-lifting theory developed in Chapters 2 to 4. The path-lifting theory provides a new perspective to revisit these results by *taking into account the symmetries of the network*.

Open research directions provided by this thesis. As explained in Section 6.1, the quantization error is controlled by the Lipschitz constant of $\theta \mapsto R_\theta(x)$. In Section 6.1, I used Lipschitz bounds established in terms of the raw parameters θ, and valid only for *LFCNs*. But since then, I proved in Chapter 3 new Lipschitz bounds, that are not only more widely applicable to *DAG networks* but also *finer* (Section 3.4.2). These new Lipschitz bounds are established in terms of the path-lifting $\Phi(\theta)$, and are finer because they are *invariant to rescaling symmetries of the network*. This raises several questions for the future. Is it possible to use the path-lifting to provide tighter approximation guarantees for quantized networks, and to extend them to general DAG networks? For given parameters θ, remember that there are uncountably many equivalent parameters θ' that can be obtained by rescaling the coordinates of θ and that are functionally equivalent ($R_\theta = R_{\theta'}$). Is it possible to take these equivalent parameters into account when designing a quantization scheme? A first result that goes in this direction in the specific case of butterfly matrices is in Gribonval et al. [2023], where they aim at first reducing the parameters to a canonical form in the rescaling equivalence class, and then quantize these canonical parameters. I showed several times in Chapter 3 that *normalized parameters* are a particular choice of equivalent parameters that are well-suited to minimize some norms defined directly in terms of the raw parameters θ. This kind of analysis may be useful to find a canonical form that would minimize the quantization error.

Last words. My aim in this thesis has been to make some of the promising tools from the literature, like path-lifting, Kronecker sparsity, and approximation guarantees, more concrete and accessible. Throughout my work, I've tried to stay grounded in practical considerations, particularly when establishing theoretical results. My goal was either to better understand practical applications or to have a direct practical impact, like designing better algorithms.

I really enjoyed working on all these different topics, especially because they were so varied. From exploring Kronecker sparsity and learning how GPUs work and can be programmed with CUDA, to studying the path-lifting, which I now see as a great tool that defines a new geometry in the path-space, with many theoretical and practical connections to the geometry defined by neural network functions.

Looking ahead, I would be happy to continue embracing this duality between theory and practice. I hope you enjoyed reading this thesis as much as I enjoyed writing it. Thank you!

Bibliography

K. Alizadeh-Vahid, A. Prabhu, A. Farhadi, and M. Rastegari. Butterfly transform: An efficient FFT based neural architecture design. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*, pages 12021–12030. Computer Vision Foundation / IEEE, 2020. doi: 10.1109/CVPR42600.2020.01204. URL https://openaccess.thecvf.com/content_CVPR_2020/html/vahid_Butterfly_Transform_An_Efficient_FFT_Based_Neural_Architecture_Design_CVPR_2020_paper.html.

C. Anil, J. Lucas, and R. B. Grosse. Sorting out Lipschitz function approximation. In K. Chaudhuri and R. Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 291–301. PMLR, 2019. URL http://proceedings.mlr.press/v97/anil19a.html.

A. Araujo, B. Négrevergne, Y. Chevaleyre, and J. Atif. On lipschitz regularization of convolutional layers using toeplitz matrix theory. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pages 6661–6669. AAAI Press, 2021. doi: 10.1609/AAAI.V35I8.16824. URL https://doi.org/10.1609/aaai.v35i8.16824.

R. Arora, A. Basu, P. Mianjy, and A. Mukherjee. Understanding deep neural networks with rectified linear units. *Electron. Colloquium Comput. Complex.*, 24:98, 2017. URL https://eccc.weizmann.ac.il/report/2017/098.

L. Aschenbrenner. Situational awareness, the decade ahead. White paper, 2024. URL https://situational-awareness.ai/wp-content/uploads/2024/06/situationalawareness.pdf. Accessed: June 2024.

F. Bach. Learning from first principles, 2024. URL https://www.di.ens.fr/~fbach/ltfp_book.pdf.

F. R. Bach. Breaking the curse of dimensionality with convex neural networks. *J. Mach. Learn. Res.*, 18:19:1–19:53, 2017. URL http://jmlr.org/papers/v18/14-546.html.

J. Barr. Amazon EC2 Update – Inf1 Instances with AWS Inferentia Chips for High Performance Cost-Effective Inferencing, 2019. URL https://aws.amazon.com/blogs/aws/amazon-ec2-update-inf1-instances-with-aws-inferentia-chips-for-high-performance-cost- Accessed: [April 2024].

A. R. Barron and J. M. Klusowski. Complexity, statistical risk, and metric entropy of deep nets using total path variation. *CoRR*, abs/1902.00800, 2019. URL http://arxiv.org/abs/1902.00800.

P. L. Bartlett. For valid generalization the size of the weights is more important than the size of the network. In M. Mozer, M. I. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems 9, NIPS, Denver, CO, USA, December 2-5, 1996*, pages 134–140. MIT Press, 1996. URL https://dl.acm.org/doi/abs/10.5555/2998981.2999000.

P. L. Bartlett and S. Mendelson. Rademacher and Gaussian complexities: Risk bounds and structural results. *J. Mach. Learn. Res.*, 3:463–482, 2002. URL http://jmlr.org/papers/v3/bartlett02a.html.

P. L. Bartlett, D. J. Foster, and M. Telgarsky. Spectrally-normalized margin bounds for neural networks. In I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 6240–6249, 2017. URL https://proceedings.neurips.cc/paper/2017/hash/b22b257ad0519d4500539da3c8bcf4dd-Abstract.html.

E. M. Bender, T. Gebru, A. McMillan-Major, and S. Shmitchell. On the dangers of stochastic parrots: Can language models be too big? In M. C. Elish, W. Isaac, and R. S. Zemel, editors, *FAccT '21: 2021 ACM Conference on Fairness, Accountability, and Transparency, Virtual Event / Toronto, Canada, March 3-10, 2021*, pages 610–623. ACM, 2021. doi: 10.1145/3442188.3445922. URL https://doi.org/10.1145/3442188.3445922.

J. Berner, P. Grohs, and A. Jentzen. Analysis of the generalization error: Empirical risk minimization over deep artificial neural networks overcomes the curse of dimensionality in the numerical approximation of black-scholes partial differential equations. *SIAM J. Math. Data Sci.*, 2(3):631–657, 2020. doi: 10.1137/19M125649X. URL https://doi.org/10.1137/19M125649X.

S. Boehm. How to optimize a CUDA matmul kernel for cuBLAS-like performance: A worklog, 2022. https://siboehm.com/articles/22/CUDA-MMM [Accessed: April 2024].

J. Bona-Pellissier, F. Malgouyres, and F. Bachoc. Local identifiability of deep relu neural networks: the theory. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022. URL http://papers.nips.cc/paper_files/paper/2022/hash/b0ae046e198a5e43141519868a959c74-Abstract-Conference.html.

S. Boucheron, G. Lugosi, and P. Massart. *Concentration inequalities*. Oxford University Press, Oxford, 2013. ISBN 978-0-19-953525-5. doi: 10.1093/acprof:oso/9780199535255.001.0001. URL https://doi-org.acces.bibliotheque-diderot.fr/10.1093/acprof:oso/9780199535255.001.0001. A nonasymptotic theory of independence, With a foreword by Michel Ledoux.

B. Chen, T. Dao, K. Liang, J. Yang, Z. Song, A. Rudra, and C. Ré. Pixelated butterfly: Simple and efficient sparse training for neural network models. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022. URL https://openreview.net/forum?id=Nfl-iXa-y7R.

H. Cheng, M. Zhang, and J. Q. Shi. A survey on deep neural network pruning-taxonomy, comparison, analysis, and recommendations. *CoRR*, abs/2308.06767, 2023. doi: 10.48550/ARXIV.2308.06767. URL https://doi.org/10.48550/arXiv.2308.06767.

Chuan Li. Demystifying GPT-3. https://lambdalabs.com/blog/demystifying-gpt-3, 2021. Accessed: [April 2024].

P. L. Combettes and J. Pesquet. Lipschitz certificates for layered network structures driven by averaged activation operators. *SIAM J. Math. Data Sci.*, 2(2):529–557, 2020. doi: 10.1137/19M1272780. URL https://doi.org/10.1137/19M1272780.

T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, 3rd Edition*. MIT Press, 2009. ISBN 978-0-262-03384-8. URL http://mitpress.mit.edu/books/introduction-algorithms.

T. Dao, A. Gu, M. Eichhorn, A. Rudra, and C. Ré. Learning fast algorithms for linear transforms using butterfly factorizations. In K. Chaudhuri and R. Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 1517–1527. PMLR, 2019. URL http://proceedings.mlr.press/v97/dao19a.html.

T. Dao, N. S. Sohoni, A. Gu, M. Eichhorn, A. Blonder, M. Leszczynski, A. Rudra, and C. Ré. Kaleidoscope: An efficient, learnable representation for all structured linear maps. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL https://openreview.net/forum?id=BkgrBgSYDS.

T. Dao, B. Chen, N. S. Sohoni, A. D. Desai, M. Poli, J. Grogan, A. Liu, A. Rao, A. Rudra, and C. Ré. Monarch: Expressive structured matrices for efficient and accurate training. In K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvári, G. Niu, and S. Sabato, editors, *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, volume 162 of *Proceedings of Machine Learning Research*, pages 4690–4721. PMLR, 2022a. URL https://proceedings.mlr.press/v162/dao22a.html.

T. Dao, D. Fu, S. Ermon, A. Rudra, and C. Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 35:16344–16359, 2022b.

B. Delattre, Q. Barthélemy, and A. Allauzen. Spectral norm of convolutional layers with circular and zero paddings. *CoRR*, abs/2402.00240, 2024. doi: 10.48550/ARXIV.2402.00240. URL https://doi.org/10.48550/arXiv.2402.00240.

R. A. DeVore and G. G. Lorentz. *Constructive Approximation*, volume 303 of *Grundlehren der mathematischen Wissenschaften*. Springer, 1993. ISBN 978-3-540-50627-0.

R. A. DeVore, B. Hanin, and G. Petrova. Neural network approximation. *Acta Numer.*, 30: 327–444, 2021. doi: 10.1017/S0962492921000052. URL https://doi.org/10.1017/S0962492921000052.

Y. Ding, J. Liu, J. Xiong, and Y. Shi. On the universal approximability and complexity bounds of quantized relu neural networks. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL https://openreview.net/forum?id=SJe9rh0cFX.

A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL https://openreview.net/forum?id=YicbFdNTTy.

G. K. Dziugaite. *Revisiting Generalization for Deep Learning: PAC-Bayes, Flat Minima, and Generative Models*. PhD thesis, Department of Engineering University of Cambridge, 2018.

G. K. Dziugaite, A. Drouin, B. Neal, N. Rajkumar, E. Caballero, L. Wang, I. Mitliagkas, and D. M. Roy. In search of robust measures of generalization. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/86d7c8a08b4aaa1bc7c599473f5dddda-Abstract.html.

W. E, C. Ma, and L. Wu. The Barron space and the flow-induced function spaces for neural network models. *Constr. Approx.*, 55(1):369–406, 2022. ISSN 0176-4276. doi: 10.1007/s00365-021-09549-y. URL https://doi-org.acces.bibliotheque-diderot.fr/10.1007/s00365-021-09549-y.

D. Elbrächter, D. Perekrestenko, P. Grohs, and H. Bölcskei. Deep neural network approximation theory. *IEEE Trans. Inf. Theory*, 67(5):2581–2623, 2021. doi: 10.1109/TIT.2021.3062161. URL https://doi.org/10.1109/TIT.2021.3062161.

K. Eykholt, I. Evtimov, E. Fernandes, B. Li, A. Rahmati, C. Xiao, A. Prakash, T. Kohno, and D. Song. Robust physical-world attacks on deep learning visual classification. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 1625–1634. Computer Vision Foundation / IEEE Computer Society, 2018. doi: 10.1109/CVPR.2018.00175. URL http://openaccess.thecvf.com/content_cvpr_2018/html/Eykholt_Robust_Physical-World_Attacks_CVPR_2018_paper.html.

J. Frankle and M. Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL https://openreview.net/forum?id=rJl-b3RcF7.

J. Frankle, G. K. Dziugaite, D. M. Roy, and M. Carbin. Pruning neural networks at initialization: Why are we missing the mark? In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL https://openreview.net/forum?id=Ig-VyQc-MLK.

D. Y. Fu, S. Arora, J. Grogan, I. Johnson, E. S. Eyuboglu, A. W. Thomas, B. Spector, M. Poli, A. Rudra, and C. Ré. Monarch mixer: A simple sub-quadratic gemm-based architecture. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/f498c1ce6bff52eb04febf87438dd84b-Abstract-Conference.html.

Y. Furusho. *Analysis of Regularization and Optimization for Deep Learning*. PhD thesis, Nara Institute of Science and Technology, 2020.

T. Galanti, M. Xu, L. Galanti, and T. A. Poggio. Norm-based generalization bounds for compositionally sparse neural networks. *CoRR*, abs/2301.12033, 2023. doi: 10.48550/arXiv.2301.12033. URL https://doi.org/10.48550/arXiv.2301.12033.

N. Golowich, A. Rakhlin, and O. Shamir. Size-independent sample complexity of neural networks. In S. Bubeck, V. Perchet, and P. Rigollet, editors, *Conference On Learning Theory, COLT 2018, Stockholm, Sweden, 6-9 July 2018*, volume 75 of *Proceedings of Machine Learning Research*, pages 297–299. PMLR, 2018. URL http://proceedings.mlr.press/v75/golowich18a.html.

A. Gonon, N. Brisebarre, R. Gribonval, and E. Riccietti. Approximation speed of quantized versus unquantized relu neural networks and beyond. *IEEE Trans. Inf. Theory*, 69(6):3960–3977, 2023a. doi: 10.1109/TIT.2023.3240360. URL https://doi.org/10.1109/TIT.2023.3240360.

A. Gonon, L. Zheng, C. Lalanne, Q.-T. Le, G. Lauga, and et al. Can sparsity improve the privacy of neural networks? In *GRETSI 2023 - XXIXème Colloque Francophone de Traitement du Signal et des Images*, Grenoble, France, August 2023b. URL https://hal.science/hal-04062317/document.

A. Gonon, N. Brisebarre, E. Riccietti, and R. Gribonval. A path-norm toolkit for modern networks: consequences, promises and challenges. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2024a. doi: 10.48550/ARXIV.2310.01225. URL https://doi.org/10.48550/arXiv.2310.01225. Spotlight.

A. Gonon, N. Brisebarre, E. Riccietti, and R. Gribonval. Path-metrics, pruning, and generalization. *CoRR*, 2024b. URL https://hal.science/hal-04584311.

A. Gonon, L. Zheng, P. Carrivain, and Q.-T. Le. Fast inference with kronecker-sparse matrices. *CoRR*, 2024c. URL https://hal.science/hal-04584450.

I. J. Goodfellow, Y. Bengio, and A. C. Courville. *Deep Learning*. Adaptive computation and machine learning. MIT Press, 2016. ISBN 978-0-262-03561-3. URL http://www.deeplearningbook.org/.

R. Gribonval, G. Kutyniok, M. Nielsen, P. Peter Petersen, and C. Schwab. Approximation spaces of deep neural networks. *Constructive Approximation*, 55:259–367, 2022. doi: 10.1007/s00365-021-09543-4.

R. Gribonval, T. Mary, and E. Riccietti. Optimal quantization of rank-one matrices in floating-point arithmetic—with applications to butterfly factorizations. preprint, 2023. URL https://inria.hal.science/hal-04125381.

P. Grohs. Optimally sparse data representations. In *Harmonic and applied analysis*, Appl. Numer. Harmon. Anal., pages 199–248. Birkhäuser/Springer, Cham, 2015.

I. Gühring, G. Kutyniok, and P. Petersen. Error bounds for approximations with deep relu neural networks in $w^{s, p}$ norms. *CoRR*, abs/1902.07896, 2019. URL http://arxiv.org/abs/1902.07896.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 770–778. IEEE Computer Society, 2016. doi: 10.1109/CVPR.2016.90. URL https://doi.org/10.1109/CVPR.2016.90.

D. Hernandez, T. B. Brown, T. Conerly, N. DasSarma, D. Drain, S. E. Showk, N. Elhage, Z. Hatfield-Dodds, T. Henighan, T. Hume, S. Johnston, B. Mann, C. Olah, C. Olsson, D. Amodei, N. Joseph, J. Kaplan, and S. McCandlish. Scaling laws and interpretability of learning from repeated data. *CoRR*, abs/2205.10487, 2022. doi: 10.48550/ARXIV.2205.10487. URL https://doi.org/10.48550/arXiv.2205.10487.

T. Hoefler, D. Alistarh, T. Ben-Nun, N. Dryden, and A. Peste. Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks. *J. Mach. Learn. Res.*, 22:241:1–241:124, 2021. URL http://jmlr.org/papers/v22/21-0366.html.

J. Hoffmann, S. Borgeaud, A. Mensch, E. Buchatskaya, T. Cai, E. Rutherford, D. de Las Casas, L. A. Hendricks, J. Welbl, A. Clark, T. Hennigan, E. Noland, K. Millican, G. van den Driessche, B. Damoc, A. Guy, S. Osindero, K. Simonyan, E. Elsen, J. W. Rae, O. Vinyals, and L. Sifre. Training compute-optimal large language models. *CoRR*, abs/2203.15556, 2022. doi: 10.48550/ARXIV.2203.15556. URL https://doi.org/10.48550/arXiv.2203.15556.

A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017. URL http://arxiv.org/abs/1704.04861.

HPCwire. AWS Upgrades its GPU-Backed AI Inference Platform. https://www.hpcwire.com/2019/03/19/aws-upgrades-its-gpu-backed-ai-inference-platform/, March 2019. Accessed: [April 2024].

S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In F. R. Bach and D. M. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 448–456. JMLR.org, 2015. URL http://proceedings.mlr.press/v37/ioffe15.html.

Y. Jiang, B. Neyshabur, H. Mobahi, D. Krishnan, and S. Bengio. Fantastic generalization measures and where to find them. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL https://openreview.net/forum?id=SJgIPJBFvH.

S. M. Kakade, K. Sridharan, and A. Tewari. On the complexity of linear prediction: Risk bounds, margin bounds, and regularization. In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, editors, *Advances in Neural Information Processing Systems 21, Proceedings of the Twenty-Second Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 8-11, 2008*, pages 793–800. Curran Associates, Inc., 2008. URL https://proceedings.neurips.cc/paper/2008/hash/5b69b9cb83065d403869739ae7f0995e-Abstract.html.

J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei. Scaling laws for neural language models. *CoRR*, abs/2001.08361, 2020. URL https://arxiv.org/abs/2001.08361.

A. Karpathy. Deep neural nets: 33 years ago and 33 years from now (invited post). In *ICLR Blog Track*, 2022. URL https://iclr-blog-track.github.io/2022/03/26/lecun1989/. https://iclr-blog-track.github.io/2022/03/26/lecun1989/.

K. Kawaguchi, L. P. Kaelbling, and Y. Bengio. Generalization in deep learning. *CoRR*, abs/1710.05468, 2017. URL http://arxiv.org/abs/1710.05468.

G. Kerkyacharian and D. Picard. Thresholding algorithms, maxisets and well-concentrated bases. *Test*, 9(2):283–344, 2000.

G. Kerkyacharian and D. Picard. Entropy, universal coding, approximation, and bases properties. *Constr. Approx.*, 20(1):1–37, 2004. ISSN 0176-4276. doi: 10.1007/s00365-003-0556-z. URL https://doi-org.acces.bibliotheque-diderot.fr/10.1007/s00365-003-0556-z.

A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In P. L. Bartlett, F. C. N. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States*, pages 1106–1114, 2012. URL https://proceedings.neurips.cc/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html.

D. Kunin, J. Sagastuy-Breña, S. Ganguli, D. L. K. Yamins, and H. Tanaka. Neural mechanics: Symmetry and broken conservation laws in deep learning dynamics. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL https://openreview.net/forum?id=q8qLAbQBupm.

A. Lacoste, A. Luccioni, V. Schmidt, and T. Dandres. Quantifying the carbon emissions of machine learning. *CoRR*, abs/1910.09700, 2019. URL http://arxiv.org/abs/1910.09700.

Q. Le. *Algorithmic and theoretical aspects of sparse deep neural networks. (Aspects algorithmiques et th 'eoriques des r 'eseaux de neurones profonds parcimonieux)*. PhD thesis, École normale supérieure de Lyon, France, 2023. URL https://tel.archives-ouvertes.fr/tel-04329531.

Q. Le, L. Zheng, E. Riccietti, and R. Gribonval. Fast learning of fast transforms, with guarantees. In *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2022, Virtual and Singapore, 23-27 May 2022*, pages 3348–3352. IEEE, 2022. doi: 10.1109/ICASSP43922.2022.9747791. URL https://doi.org/10.1109/ICASSP43922.2022.9747791.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proc. IEEE*, 86(11):2278–2324, 1998. doi: 10.1109/5.726791. URL https://doi.org/10.1109/5.726791.

Y. LeCun, Y. Bengio, and G. E. Hinton. Deep learning. *Nat.*, 521(7553):436–444, 2015. doi: 10.1038/NATURE14539. URL https://doi.org/10.1038/nature14539.

M. Ledoux and M. Talagrand. *Probability in Banach spaces*, volume 23 of *Ergebnisse der Mathematik und ihrer Grenzgebiete (3) [Results in Mathematics and Related Areas (3)]*. Springer-Verlag, Berlin, 1991. ISBN 3-540-52013-9. doi: 10.1007/978-3-642-20212-4. URL https://doi.org/10.1007/978-3-642-20212-4. Isoperimetry and processes.

M. Leshno, V. Y. Lin, A. Pinkus, and S. Schocken. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural Networks*, 6(6):861–867, 1993. ISSN 0893-6080. doi: https://doi.org/10.1016/S0893-6080(05)80131-5. URL https://www.sciencedirect.com/science/article/pii/S0893608005801315.

X. Li, Y. Liang, S. Yan, L. Jia, and Y. Li. A coordinated tiling and batching framework for efficient GEMM on GPUs. In *Proceedings of the 24th Symposium on Principles and Practice of Parallel Programming*, 2019.

R. Lin, J. Ran, K. H. Chiu, G. Chesi, and N. Wong. Deformable butterfly: A highly structured and sparse linear transform. In M. Ranzato, A. Beygelzimer, Y. N. Dauphin, P. Liang, and J. W. Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 16145–16157, 2021. URL https://proceedings.neurips.cc/paper/2021/hash/86b122d4358357d834a87ce618a55de0-Abstract.html.

S. Marcotte, R. Gribonval, and G. Peyré. Abide by the law and follow the flow: Conservation laws for gradient flows. *CoRR*, abs/2307.00144, 2023. doi: 10.48550/arXiv.2307.00144. URL https://doi.org/10.48550/arXiv.2307.00144.

A. Maurer. A vector-contraction inequality for rademacher complexities. In R. Ortner, H. U. Simon, and S. Zilles, editors, *Algorithmic Learning Theory - 27th International Conference, ALT 2016, Bari, Italy, October 19-21, 2016, Proceedings*, volume 9925 of *Lecture Notes in Computer Science*, pages 3–17, 2016. doi: 10.1007/978-3-319-46379-7_1. URL https://doi.org/10.1007/978-3-319-46379-7_1.

Meta. Llama 3 on Hugging Face. https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct, 2024. Accessed: [May 2024].

I. Mirzadeh, K. Alizadeh, S. Mehta, C. C. D. Mundo, O. Tuzel, G. Samei, M. Rastegari, and M. Farajtabar. Relu strikes back: Exploiting activation sparsity in large language models. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2024. doi: 10.48550/ARXIV.2310.04564. URL https://doi.org/10.48550/arXiv.2310.04564. Oral.

D. Narayanan, M. Shoeybi, J. Casper, P. LeGresley, M. Patwary, V. Korthikanti, D. Vainbrand, P. Kashinkunti, J. Bernauer, B. Catanzaro, A. Phanishayee, and M. Zaharia. Efficient large-scale language model training on GPU clusters using megatron-lm. In B. R. de Supinski, M. W. Hall, and T. Gamblin, editors, *International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2021, St. Louis, Missouri, USA, November 14-19, 2021*, page 58. ACM, 2021. doi: 10.1145/3458817.3476209. URL https://doi.org/10.1145/3458817.3476209.

B. Neyshabur. Implicit regularization in deep learning. *CoRR*, abs/1709.01953, 2017. URL http://arxiv.org/abs/1709.01953.

B. Neyshabur, R. Tomioka, and N. Srebro. Norm-based capacity control in neural networks. In P. Grünwald, E. Hazan, and S. Kale, editors, *Proceedings of The 28th Conference on Learning Theory, COLT 2015, Paris, France, July 3-6, 2015*, volume 40 of *JMLR Workshop and Conference Proceedings*, pages 1376–1401. JMLR.org, 2015. URL http://proceedings.mlr.press/v40/Neyshabur15.html.

B. Neyshabur, S. Bhojanapalli, D. McAllester, and N. Srebro. Exploring generalization in deep learning. In I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5947–5956, 2017. URL https://proceedings.neurips.cc/paper/2017/hash/10ce03a1ed01077e3e289f3e53c72813-Abstract.html.

B. Neyshabur, S. Bhojanapalli, and N. Srebro. A PAC-Bayesian approach to spectrally-normalized margin bounds for neural networks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL https://openreview.net/forum?id=Skz_WfbCZ.

E. Noether. Invariante variationsprobleme. *Nachrichten von der Gesellschaft der Wissenschaften zu Göttingen, Mathematisch-Physikalische Klasse*, 1918:235–257, 1918. URL http://eudml.org/doc/59024.

NVIDIA. Efficient GEMM in CUDA: documentation, 2023a. https://github.com/NVIDIA/cutlass/blob/main/media/docs/efficient_gemm.md [Accessed: April 2024].

NVIDIA. Matrix multiplication background user's guide, 2023b. https://docs.nvidia.com/deeplearning/performance/dl-performance-matrix-multiplication/index.html [Accessed: April 2024].

NVIDIA. CUDA C++ programming guide, 2024. https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html [Accessed: April 2024].

I. Ohn and Y. Kim. Smooth function approximation by deep neural networks with general activation functions. *Entropy*, 21(7):627, 2019. doi: 10.3390/E21070627. URL https://doi.org/10.3390/e21070627.

OpenAI. AI and Compute. https://openai.com/research/ai-and-compute, May 2018. Accessed: [April 2024].

G. Penedo, Q. Malartic, D. Hesslow, R. Cojocaru, H. Alobeidli, A. Cappelli, B. Pannier, E. Almazrouei, and J. Launay. The refinedweb dataset for falcon LLM: outperforming curated corpora with web data only. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/fa3ed726cc5073b9c31e3e49a807789c-Abstract-Datasets_and_Benchmarks.html.

G. V. Pérez and A. A. Louis. Generalization bounds for deep learning. *CoRR*, abs/2012.04115, 2020. URL https://arxiv.org/abs/2012.04115.

P. Petersen and F. Voigtländer. Optimal approximation of piecewise smooth functions using deep relu neural networks. *Neural Networks*, 108:296–330, 2018. doi: 10.1016/J.NEUNET.2018.08.019. URL https://doi.org/10.1016/j.neunet.2018.08.019.

K. Pitas, A. Loukas, M. Davies, and P. Vandergheynst. Some limitations of norm based generalization bounds in deep neural networks. *CoRR*, abs/1905.09677, 2019. URL http://arxiv.org/abs/1905.09677.

A. Rogozhnikov. Einops: Clear and reliable tensor manipulations with einstein-like notation. In *ICLR*, 2021.

M. E. Sander, J. Puigcerver, J. Djolonga, G. Peyré, and M. Blondel. Fast, differentiable and sparse top-k: a convex analysis perspective. In A. Krause, E. Brunskill, K. Cho, B. Engelhardt, S. Sabato, and J. Scarlett, editors, *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 29919–29936. PMLR, 2023. URL https://proceedings.mlr.press/v202/sander23a.html.

G. Sastry, L. Heim, H. Belfield, M. Anderljung, M. Brundage, J. Hazell, C. O'Keefe, G. K. Hadfield, R. Ngo, K. Pilz, G. Gor, E. Bluemke, S. Shoker, J. Egan, R. F. Trager, S. Avin, A. Weller, Y. Bengio, and D. Coyle. Computing power and the governance of artificial intelligence. *CoRR*, abs/2402.08797, 2024. doi: 10.48550/ARXIV.2402.08797. URL https://doi.org/10.48550/arXiv.2402.08797.

T. L. Scao, T. Wang, D. Hesslow, S. Bekman, M. S. Bari, S. Biderman, H. Elsahar, N. Muennighoff, J. Phang, O. Press, C. Raffel, V. Sanh, S. Shen, L. Sutawika, J. Tae, Z. X. Yong, J. Launay, and I. Beltagy. What language model to train if you have one million GPU hours? In Y. Goldberg, Z. Kozareva, and Y. Zhang, editors, *Findings of the Association for Computational Linguistics: EMNLP 2022, Abu Dhabi, United Arab Emirates, December 7-11, 2022*, pages 765–782. Association for Computational Linguistics, 2022. doi: 10.18653/V1/2022.FINDINGS-EMNLP.54. URL https://doi.org/10.18653/v1/2022.findings-emnlp.54.

H. Sedghi, V. Gupta, and P. M. Long. The singular values of convolutional layers. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL https://openreview.net/forum?id=rJevYoA9Fm.

J. Sevilla, L. Heim, A. Ho, T. Besiroglu, M. Hobbhahn, and P. Villalobos. Compute trends across three eras of machine learning. In *International Joint Conference on Neural Networks, IJCNN 2022, Padua, Italy, July 18-23, 2022*, pages 1–8. IEEE, 2022. doi: 10.1109/IJCNN55064.2022.9891914. URL https://doi.org/10.1109/IJCNN55064.2022.9891914.

S. Shalev-Shwartz and S. Ben-David. *Understanding Machine Learning - From Theory to Algorithms*. Cambridge University Press, 2014. ISBN 978-1-10-705713-5. URL http://www.cambridge.org/de/academic/subjects/computer-science/pattern-recognition-and-machine-learning/understanding-machine-learning-theory-algorithms.

D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. P. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. Mastering the game of go with deep neural networks and tree search. *Nat.*, 529(7587):484–489, 2016. doi: 10.1038/NATURE16961. URL https://doi.org/10.1038/nature16961.

D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. P. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis. Mastering the game of go without human knowledge. *Nat.*, 550(7676):354–359, 2017. doi: 10.1038/NATURE24270. URL https://doi.org/10.1038/nature24270.

D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis. A general reinforcement learning algorithm that masters chess, shogi, and go through

self-play. *Science*, 362(6419):1140–1144, 2018. doi: 10.1126/science.aar6404. URL https://www.science.org/doi/abs/10.1126/science.aar6404.

K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In Y. Bengio and Y. LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1409.1556.

P. Stock and R. Gribonval. An embedding of ReLU networks and an analysis of their identifiability. *Constr. Approx.*, 57(2):853–899, 2023. ISSN 0176-4276,1432-0940. doi: 10.1007/s00365-022-09578-1. URL https://doi.org/10.1007/s00365-022-09578-1.

E. Strubell, A. Ganesh, and A. McCallum. Energy and policy considerations for deep learning in NLP. In A. Korhonen, D. R. Traum, and L. Màrquez, editors, *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pages 3645–3650. Association for Computational Linguistics, 2019. doi: 10.18653/V1/P19-1355. URL https://doi.org/10.18653/v1/p19-1355.

T. Suzuki. Adaptivity of deep relu network for learning in besov and mixed smooth besov spaces: optimal rate and curse of dimensionality. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL https://openreview.net/forum?id=H1ebTsActm.

C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. J. Goodfellow, and R. Fergus. Intriguing properties of neural networks. In Y. Bengio and Y. LeCun, editors, *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014. URL http://arxiv.org/abs/1312.6199.

C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, pages 1–9. IEEE Computer Society, 2015. doi: 10.1109/CVPR.2015.7298594. URL https://doi.org/10.1109/CVPR.2015.7298594.

R. Van Handel. Probability in high dimension. *Lecture Notes (Princeton University)*, 2014. URL https://web.math.princeton.edu/~rvan/APC550.pdf. [Accessed: April 2024].

C. F. Van Loan. The ubiquitous kronecker product. *Journal of computational and applied mathematics*, 123(1-2):85–100, 2000.

A. Virmaux and K. Scaman. Lipschitz regularity of deep neural networks: analysis and efficient estimation. In S. Bengio, H. M. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 3839–3848, 2018. URL https://proceedings.neurips.cc/paper/2018/hash/d54e99a6c03704e95e6965532dec148b-Abstract.html.

U. von Luxburg and O. Bousquet. Distance-based classification with Lipschitz functions. *J. Mach. Learn. Res.*, 5:669–695, 2004. URL http://jmlr.org/papers/volume5/luxburg04b/luxburg04b.pdf.

M. J. Wainwright. *High-dimensional statistics*, volume 48 of *Cambridge Series in Statistical and Probabilistic Mathematics*. Cambridge University Press, Cambridge, 2019. ISBN 978-1-108-49802-9. doi: 10.1017/9781108627771. URL https://doi-org.acces.bibliotheque-diderot.fr/10.1017/9781108627771. A non-asymptotic viewpoint.

P. Wang. Scaled dot-product attention implementation, 2024a. https://docs.nvidia.com/deeplearning/performance/dl-performance-matrix-multiplication/index.html [Accessed: April 2024].

P. Wang. Simple ViT implementation, 2024b. https://github.com/lucidrains/vit-pytorch/blob/main/vit_pytorch/simple_vit.py [Accessed: April 2024].

C. Wu, R. Raghavendra, U. Gupta, B. Acun, N. Ardalani, K. Maeng, G. Chang, F. A. Behram, J. Huang, C. Bai, M. Gschwind, A. Gupta, M. Ott, A. Melnikov, S. Candido, D. Brooks, G. Chauhan, B. Lee, H. S. Lee, B. Akyildiz, M. Balandat, J. Spisak, R. Jain, M. Rabbat, and K. M. Hazelwood. Sustainable AI: environmental implications, challenges and opportunities. In D. Marculescu, Y. Chi, and C. Wu, editors, *Proceedings of Machine Learning and Systems 2022, MLSys 2022, Santa Clara, CA, USA, August 29 - September 1, 2022*. mlsys.org, 2022. URL https://proceedings.mlsys.org/paper/2022/hash/ed3d2c21991e3bef5e069713af9fa6ca-Abstract.html.

X. Zhai, A. Kolesnikov, N. Houlsby, and L. Beyer. Scaling vision transformers. In *CVPR*, 2022.

C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals. Understanding deep learning (still) requires rethinking generalization. *Commun. ACM*, 64(3):107–115, 2021. doi: 10.1145/3446776. URL https://doi.org/10.1145/3446776.

L. Zheng, E. Riccietti, and R. Gribonval. Efficient identification of butterfly sparse matrix factorizations. *SIAM J. Math. Data Sci.*, 5(1):22–49, 2023. doi: 10.1137/22M1488727. URL https://doi.org/10.1137/22m1488727.

S. Zheng, Q. Meng, H. Zhang, W. Chen, N. Yu, and T. Liu. Capacity control of ReLU neural networks by basis-path norm. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*, pages 5925–5932. AAAI Press, 2019. doi: 10.1609/aaai.v33i01.33015925. URL https://doi.org/10.1609/aaai.v33i01.33015925.

Appendix A

Supplemental material for Chapter 2

I invite the reader to skip this since it only contains technical details, included for the sake of completeness.

Proof of Equation (2.8). My goal is to prove Equation (2.8) that I recall here for convenience:

$$v(\theta, x) = a_v(\theta, x) b_v + \sum_{u \in \mathrm{ant}(v)} u(\theta, x) a_{u \to v}(\theta, x) \theta^{u \to v}.$$

Case of an identity neuron. If v is an identity neuron, then by Definition 2.2.2

$$v(\theta, x) = b_v + \sum_{u \in \mathrm{ant}(v)} u(\theta, x) \theta^{u \to v}.$$

Moreover, by Definition 2.3.3 we have $a_v(\theta, x) = 1$ and $a_{u \to v}(\theta, x) = 1$, since v is an identity neuron, so that the previous equality can also be written as

$$v(\theta, x) = a_v(\theta, x) b_v + \sum_{u \in \mathrm{ant}(v)} u(\theta, x) a_{u \to v}(\theta, x) \theta^{u \to v}.$$

This shows Equation (2.8) in this case.

Case of a ReLU neuron. If v is a ReLU neuron then similarly

$$\begin{aligned} v(\theta, x) = \mathrm{ReLU}\left(b_v + \sum_{u \in \mathrm{ant}(v)} u(\theta, x) \theta^{u \to v} \right) &= \mathbb{1}_{v(\theta,x)>0} \left(b_v + \sum_{u \in \mathrm{ant}(v)} u(\theta, x) \theta^{u \to v} \right) \\ &= \underbrace{\mathbb{1}_{v(\theta,x)>0}}_{= a_v(\theta,x)} b_v + \sum_{u \in \mathrm{ant}(v)} u(\theta, x) \underbrace{\mathbb{1}_{v(\theta,x)>0}}_{= a_{u \to v}(\theta,x)} \theta^{u \to v}. \end{aligned}$$

This shows again Equation (2.8).

Case of a k-max-pooling neuron. When v is a k-max-pooling neuron, it holds:

$$\begin{aligned} v(\theta, x) &= k\text{-}\mathtt{pool}\left((b_v + u(\theta, x) \theta^{u \to v})_{u \in \mathrm{ant}(v)} \right) \\ &= b_v + \sum_{u \in \mathrm{ant}(v)} \underbrace{\mathbb{1}_{u \text{ is the first to realize this } k\text{-}\mathtt{pool}}}_{= a_{u \to v}(\theta, x) \text{ (Definition 2.3.3)}} u(\theta, x) \theta^{u \to v} \qquad \text{(Definition 2.2.1)} \\ &= \underbrace{a_v(\theta, x)}_{= 1 \text{ (Definition 2.3.3)}} b_v + \sum_{u \in \mathrm{ant}(v)} u(\theta, x) a_{u \to v}(\theta, x) \theta^{u \to v}. \end{aligned}$$

This finishes proving Equation (2.8) in every case. □

Appendix B

Supplemental material for Chapter 3

I invite readers to skip Appendix B that mostly contains technical details, included for the sake of completeness.

- Appendix B.1 proves that Algorithm 3.2.1 produces normalized parameters that are weakly rescaling-equivalent to the input parameters.
- Appendix B.2 shows that the mixed path-norm is equal to the minimum of the product of layers' norms over all weakly rescaling-equivalent parameters.
- Appendix B.3 compares the ℓ^1-path-metric to the ℓ^∞-metric on the parameters for LFCNs.

B.1 Proof of Lemma 3.2.1

I now prove Lemma 3.2.1: Algorithm 3.2.1 produces normalized parameters that are weakly rescaling-equivalent to the input parameters.

Proof of Lemma 3.2.1. **Step 1.** I first prove that for every $v \in N \setminus (N_{\text{in}} \cup N_{\text{out}})$:

$$\left\| \begin{pmatrix} \theta^{\to v} \\ b_v \end{pmatrix} \right\|_q \in \{0, 1\}. \tag{B.1}$$

Consider $v \notin N_{\text{in}} \cup N_{\text{out}}$. Right after v has been processed by the for loop of Algorithm 3.2.1 (line 2), it is clear that Equation (B.1) holds true for v. It remains to see that b_v and $\theta^{\to v}$ are not modified until the end of Algorithm 3.2.1. A bias can only be modified when this is the turn of the corresponding neuron, so b_v is untouched after the iteration corresponding to v. And $\theta^{\to v}$ can only be modified when treating antecedents of v, but since antecedents must be before v in any topological sorting, they cannot be processed after v. This proves that $\left\| \begin{pmatrix} \theta^{\to v} \\ b_v \end{pmatrix} \right\|_q \in \{0, 1\}$.

Step 2. I now prove that if $\left\| \begin{pmatrix} \theta^{\to v} \\ b_v \end{pmatrix} \right\|_q = 0$ then it also holds $\theta^{v\to} = 0$ (*outgoing* edges of v). Indeed, $\left\| \begin{pmatrix} \theta^{\to v} \\ b_v \end{pmatrix} \right\|_q = 0$ implies $\lambda_v = 0$ in Algorithm 3.2.1, which implies

that $\theta^{v\to} = 0$ right after rescaling (line 6) at the end of the iteration corresponding to v. Because the next iterations can only involve multiplication of the coordinates of $\theta^{v\to}$ by scalars, $\theta^{v\to} = 0$ is also satisfied at the end of the algorithm. This proves the claim.

Step 3. In order to prove that θ is q-normalized, it only remains to establish that $\|\Phi^{\to v}(\theta)\|_q = \|\begin{pmatrix} \theta^{\to v} \\ b_v \end{pmatrix}\|_q$. I prove this by induction on a topological sorting of the neurons.

A useful fact for the induction is that for every $v \notin N_{\text{in}}$:

$$\Phi^{\to v}(\theta) = \begin{pmatrix} (\Phi^{\to u}(\theta)\theta^{u\to v})_{u\in\text{ant}(v)} \\ b_v \end{pmatrix} \tag{B.2}$$

where I recall that $\Phi^{\to u}(\cdot) = 1$ for input neurons u. Equation (B.2) holds because $\Phi^{\to v}$ is the path-lifting of $G^{\to v}$ (see Definition 2.3.2), and the only paths in $G^{\to v}$ are $p = v$, and the paths going through antecedents of v (v has antecedents since it is not an input neuron).

Let's now start the induction. By definition, the first neuron $v \notin N_{\text{in}}$ in a topological sorting has only input neurons as antecedents. Therefore, $\Phi^{\to u}(\theta) = 1$ for every $u \in \text{ant}(v)$. Using Equation (B.2), we get

$$\|\Phi^{\to v}(\theta)\|_q^q = |b_v|^q + \sum_{u\in\text{ant}(v)} |\theta^{u\to v}|^q = \left\|\begin{pmatrix} \theta^{\to v} \\ b_v \end{pmatrix}\right\|_q^q.$$

This shows the claim for v.

Assume the result to be true for $v \notin N_{\text{in}}$ and all the neurons before v in the considered topological order (in particular, for every $u \in \text{ant}(u)$). By Equation (B.2), we have

$$\|\Phi^{\to v}(\theta)\|_q^q = |b_v|^q + \sum_{u\in\text{ant}(v)} \|\Phi^{\to u}(\theta)\|_q^q |\theta^{u\to v}|^q.$$

The induction hypothesis guarantees that $c_u := \|\Phi^{\to u}(\theta)\|_q = \|\begin{pmatrix} \theta^{\to u} \\ b_u \end{pmatrix}\|_q$ for every $u \in \text{ant}(v)$. By Equation (B.1), we also have $c_u \in \{0, 1\}$ for every $u \in \text{ant}(v)$. When $c_u = 1$, it clearly holds $\|\Phi^{\to u}(\theta)\|_q^q |\theta^{u\to v}|^q = |\theta^{u\to v}|^q$. Otherwise, $c_u = 0$, hence $\theta^{u\to} = 0$ as proved above, and we also obtain $\|\Phi^{\to u}(\theta)\|_q^q |\theta^{u\to v}|^q = |\theta^{u\to v}|^q$. We deduce that

$$\|\Phi^{\to v}(\theta)\|_q^q = |b_v|^q + \sum_{u\in\text{ant}(v)} |\theta^{u\to v}|^q = \left\|\begin{pmatrix} \theta^{\to v} \\ b_v \end{pmatrix}\right\|_q^q.$$

This concludes the induction.

Step 4. It only remains to see that θ is weakly rescaling-equivalent to $\tilde{\theta}$. This is clear since each iteration of the for loop of Algorithm 3.2.1 precisely applies an operation that preserves weak rescaling-equivalence of θ. □

B.2 Proof of Theorem 3.3.2

I now prove that mixed path-norms $\|\Phi(\theta)\|_{q,r}$ coincide with the extended product of layers' norms $\Pi_{q,r}(\theta)$ when the parameters are q-normalized. I also prove that $\|\Phi(\theta)\|_{q,r} \leqslant \Pi_{q,r}(\theta)$ in general, so we will deduce using Lemma 3.2.1 that:

$$\|\Phi(\theta)\|_{q,r} = \min_{\theta' \sim_{WR} \theta} \Pi_{q,r}(\theta').$$

Proof of Theorem 3.3.2. First, when θ is q-normalized, by Definition 3.2.1 and Equation (3.1):

$$\|\Phi(\theta)\|_{q,r} = \left\| \left(\left\| \begin{pmatrix} \theta^{\to v} \\ b_v \end{pmatrix} \right\|_q \right)_{v \in N_{\text{out}}} \right\|_r .$$

Let me prove by induction on a topological sorting of the neurons that for every $v \in N$:

$$\|\Phi^{\to v}(\theta)\|_q \leqslant \max_{p \in \mathcal{P}^{\to v}} \Pi(\theta, p, q) \tag{B.3}$$

with equality if θ is q-normalized. As a direct consequence of Equations (3.1) and (3.4), this will prove that $\|\Phi(\theta)\|_{q,r} \leqslant \Pi_{q,r}(\theta)$, with equality when θ is q-normalized, yielding all the claimed results.

I start with $v \in N_{\text{in}}$ since input neurons are the first to appear in a topological sorting. In this case, the only path in $\mathcal{P}^{\to v}$ is $p = v$, for which it holds $\Pi(\theta, p, q) = |\gamma_{p_0}| = |\gamma_v| = 1$ by Definition 3.3.1. Moreover, we also have $\Phi^{\to v}(\theta) = 1$ (Definition 2.3.3). This proves Equation (B.3) and the case of equality for input neurons even if θ is not normalized.

Now, consider $v \notin N_{\text{in}}$ and assume Equation (B.3), and the case of equality for q-normalized parameters, to be true for every antecedent of v. In this case, we have

$$\Phi^{\to v}(\theta) = \begin{pmatrix} (\Phi^{\to u}(\theta)\theta^{u \to v})_{u \in \text{ant}(v)} \\ b_v \end{pmatrix},$$

so it holds

$$\|\Phi^{\to v}(\theta)\|_q^q = |b_v|^q + \sum_{u \in \text{ant}(v)} \|\Phi^{\to u}(\theta)\|_q^q |\theta^{u \to v}|^q \leqslant |b_v|^q + \|\theta^{\to v}\|_q^q \max_{u \in \text{ant}(v)} \|\Phi^{\to u}(\theta)\|_q^q.$$

Using the induction hypothesis on every $u \in \text{ant}(v)$ yields:

$$\|\Phi^{\to v}(\theta)\|_q^q \leqslant |b_v|^q + \|\theta^{\to v}\|_q^q \max_{u \in \text{ant}(v)} \max_{p \in \mathcal{P}^{\to u}} (\Pi(\theta, p, q))^q$$

Consider $u \in \text{ant}(v)$ and $p \in \mathcal{P}^{\to u}$, and denote by $\tilde{p} = p \to v$ the path p concatenated with the edge $u \to v$. By Definition 3.3.1, we have (highlighting in orange important changes)

$$\begin{aligned}
&|b_v|^q + \|\theta^{\to v}\|_q^q (\Pi(\theta, p, q))^q \\
&= \underbrace{|b_v|^q}_{= |\gamma_v|^q \text{ since } v \notin N_{\text{in}}} + \|\theta^{\to v}\|_q^q \sum_{\ell=0}^{\texttt{length}(p)} \left|\underline{b}_{p_\ell}\right|^q \prod_{k=\ell+1}^{\texttt{length}(p)} \|\theta^{\to p_k}\|_q^q \\
&= |\gamma_{\tilde{p}_{\text{end}}}|^q + \|\theta^{\to \tilde{p}_{\text{end}}}\|_q^q \sum_{\ell=0}^{\texttt{length}(\tilde{p})-1} \left|\underline{b}_{\tilde{p}_\ell}\right|^q \prod_{k=\ell+1}^{\texttt{length}(\tilde{p})-1} \|\theta^{\to \tilde{p}_k}\|_q^q \qquad = (\Pi(\theta, \tilde{p}, q))^q
\end{aligned}$$

I deduce that

$$\|\Phi^{\to v}(\theta)\|_q^q \leqslant \max_{u \in \text{ant}(v)} \max_{p \in \mathcal{P}^{\to u}} (\Pi(\theta, p \to u, q))^q = \max_{\tilde{p} \in \mathcal{P}^{\to v}} (\Pi(\theta, \tilde{p}, q))^q.$$

This proves Equation (B.3) for v. To conclude the induction, it remains to treat the equality case for v assuming that θ is q-normalized. Since $v \notin N_{\text{in}}$, $\text{ant}(v) \neq \emptyset$, and since

each neuron $u \in \mathrm{ant}(v)$ cannot be an output neuron, the fact that θ is q-normalized implies by Definition 3.2.1 that $\|\Phi^{\to u}(\theta)\|_q \in \{0,1\}$, with $\theta^{u\to} = 0$ as soon as $\|\Phi^{\to u}(\theta)\|_q = 0$. This implies

$$\|\Phi^{\to u}(\theta)\|_q^q |\theta^{u\to v}|^q = |\theta^{u\to v}|^q.$$

As a result

$$\sum_{u\in\mathrm{ant}(v)} \|\Phi^{\to u}(\theta)\|_q^q |\theta^{u\to v}|^q = \sum_{u\in\mathrm{ant}(v)} |\theta^{u\to v}|^q = \|\theta^{\to v}\|_q^q = \|\theta^{\to v}\|_q^q \max_{u\in\mathrm{ant}(v)} \|\Phi^{\to u}(\theta)\|_q^q.$$

By the induction hypothesis, we also have

$$\max_{u\in\mathrm{ant}(v)} \|\Phi^{\to u}(\theta)\|_q^q = \max_{u\in\mathrm{ant}(v)} \max_{p\in\mathcal{P}^{\to u}} (\Pi(\theta,p,q))^q.$$

Putting everything together yields

$$\begin{aligned}
\|\Phi^{\to v}(\theta)\|_q^q &= |b_v|^q + \sum_{u\in\mathrm{ant}(v)} \|\Phi^{\to u}(\theta)\|_q^q |\theta^{u\to v}|^q \\
&= |b_v|^q + \|\theta^{\to v}\|_q^q \max_{u\in\mathrm{ant}(v)} \max_{p\in\mathcal{P}^{\to u}} (\Pi(\theta,p,q))^q \\
&= \max_{p\in\mathcal{P}^{\to v}} \Pi(\theta,p,q)^q.
\end{aligned}$$

This shows the case of equality and concludes the induction. □

B.3 Comparison of the ℓ^1-path-metric and ℓ^∞-metric on the parameters

Our first objective is to prove that for LFCNs, the ℓ^1-path-metric is smaller than another metric based on the raw parameters and that is used in the literature for Lipschitz bounds in θ.

Lemma B.3.1. *Consider a LFCN with L affine layers and no biases corresponding to functions of the form*

$$R_\theta(x) = M_L \,\mathrm{ReLU}(M_{L-1} \ldots \mathrm{ReLU}(M_1 x))$$

with $\theta = (M_1, \ldots, M_L)$ and $M_\ell \in \mathbb{R}^{d_{\ell+1}\times d_\ell}$. For a matrix M, denote by $\|M\|_{1,\infty} := \max_{row\ of\ M} \|row\|_1$. Define the width $W := \max_\ell d_\ell$. For θ, θ', define $R := \max_\ell(\|M_\ell\|_{1,\infty}, \|M'_\ell\|_{1,\infty})$. We have:

$$\|\Phi(\theta) - \Phi(\theta')\|_1 \leqslant L W^2 R^{L-1} \|\theta - \theta'\|_\infty. \tag{B.4}$$

To prove Lemma B.3.1, I start with the following Lipschitz property of $\theta \mapsto \Phi(\theta)$.

Lemma B.3.2. *Consider $q \in [1,\infty)$, parameters θ and θ', and a neuron v. Then, it holds:*

$$\begin{aligned}
&\|\Phi^{\to v}(\theta) - \Phi^{\to v}(\theta')\|_q^q \\
&\leqslant \max_{p\in\mathcal{P}^{\to v}} \sum_{\ell=1}^{\mathtt{length}(p)} \left(\prod_{k=\ell+1}^{\mathtt{length}(p)} \|\theta^{\to p_k}\|_q^q \right) \left(|b_{p_\ell} - b'_{p_\ell}|^q + \|\theta^{\to p_\ell} - (\theta')^{\to p_\ell}\|_q^q \max_{u\in\mathrm{ant}(p_\ell)} \|\Phi^{\to u}(\theta')\|_q^q \right)
\end{aligned} \tag{B.5}$$

with the convention that an empty sum and product are respectively equal to zero and one.

Note that when all the paths in $\mathcal{P}^{\to v}$ have the same length L, Equation (B.5) is homogeneous: multiplying both θ and θ' coordinate-wise by a scalar α scales both sides of the equations by α^L.

Proof. The proof of Equation (B.5) goes by induction on a topological sorting of the graph. The first neurons of the sorting are the neurons without antecedents: input neurons.

Consider an input neuron v. There is only a single path ending at v: the path $p = v$. By Definition 2.3.3, $\Phi^{\to v}(\cdot) = \Phi_v(\cdot) = 1$ (empty product) so the left hand-side is zero. On the right-hand side, there is only a single choice for a path ending at v: this is the path $p = v$ that starts and ends at v. Thus $D = 0$, and the maximum is zero (empty sum). This proves Equation (B.5) for input neurons.

Consider a neuron $v \notin N_{in}$ and assume that this is true for every neuron before v in the considered topological sorting. Recall that, by definition, $\Phi^{\to v}$ is the path-lifting of $G^{\to v}$ (see Definition 2.3.3). The paths in $G^{\to v}$ are $p = v$, and the paths going through antecedents of v (v has antecedents since it is not an input neuron). So $\Phi^{\to v}(\theta) = \begin{pmatrix} (\Phi^{\to u}(\theta)\theta^{u\to v})_{u\in \mathrm{ant}(v)} \\ b_v \end{pmatrix}$ and we have:

$$
\begin{aligned}
&\|\Phi^{\to v}(\theta) - \Phi^{\to v}(\theta')\|_q^q \\
&= |b_v - b'_v|^q + \sum_{u\in\mathrm{ant}(v)} \|\Phi^{\to u}(\theta)\theta^{u\to v} - \Phi^{\to u}(\theta')(\theta')^{u\to v}\|_q^q \\
&\leqslant |b_v - b'_v|^q + \sum_{u\in\mathrm{ant}(v)} \left(\|\Phi^{\to u}(\theta) - \Phi^{\to u}(\theta')\|_q^q |\theta^{u\to v}|^q + \|\Phi^{\to u}(\theta')\|_q^q |\theta^{u\to v} - (\theta')^{u\to v}|^q\right) \\
&\leqslant |b_v - b'_v|^q + \|\theta^{\to v}\|_q^q \max_{u\in\mathrm{ant}(v)} \|\Phi^{\to u}(\theta) - \Phi^{\to u}(\theta')\|_q^q + \|\theta^{\to v} - (\theta')^{\to v}\|_q^q \max_{u\in\mathrm{ant}(v)} \|\Phi^{\to u}(\theta')\|_q^q.
\end{aligned}
$$

Using the induction hypothesis (Equation (B.5)) on the antecedents of v and observing that $p \in \mathcal{P}^{\to v}$ if, and only if there are $u \in \mathrm{ant}(v)$ and $r \in \mathcal{P}^{\to u}$ such that $p = r \to v$, we get (highlighting in blue the important changes):

$$
\begin{aligned}
&\|\Phi^{\to v}(\theta) - \Phi^{\to v}(\theta)\|_q^q \leqslant |b_v - b'_v|^q + \|\theta^{\to v} - (\theta')^{\to v}\|_q^q \max_{u\in\mathrm{ant}(v)} \|\Phi^{\to u}(\theta')\|_q^q \\
&+\|\theta^{\to v}\|_q^q \max_{u\in\mathrm{ant}(v)} \max_{r\in\mathcal{P}^{\to u}} \sum_{\ell=1}^{\mathtt{length}(r)} \left(\prod_{k=\ell+1}^{\mathtt{length}(r)} \|\theta^{\to r_k}\|_q^q\right) \left(|b_{r_\ell} - b'_{r_\ell}|^q + \|\theta^{\to r_\ell} - (\theta')^{\to r_\ell}\|_q^q \max_{w\in\mathrm{ant}(r_\ell)} \|\Phi^{\to w}(\theta')\|_q^q\right) \\
&= |b_v - b'_v|^q + \|\theta^{\to v} - (\theta')^{\to v}\|_q^q \max_{u\in\mathrm{ant}(v)} \|\Phi^{\to u}(\theta')\|_q^q \\
&+ \max_{p\in\mathcal{P}^{\to v}} \sum_{\ell=1}^{\mathtt{length}(p)-1} \left(\prod_{k=\ell+1}^{\mathtt{length}(p)} \|\theta^{\to p_k}\|_q^q\right) \left(|b_{p_\ell} - b'_{p_\ell}|^q + \|\theta^{\to p_\ell} - (\theta')^{\to p_\ell}\|_q^q \max_{w\in\mathrm{ant}(p_\ell)} \|\Phi^{\to w}(\theta')\|_q^q\right) \\
&= \max_{p\in\mathcal{P}^{\to v}} \sum_{\ell=1}^{\mathtt{length}(p)} \left(\prod_{k=\ell+1}^{\mathtt{length}(p)} \|\theta^{\to p_k}\|_q^q\right) \left(|b_{p_\ell} - b'_{p_\ell}|^q + \|\theta^{\to p_\ell} - (\theta')^{\to p_\ell}\|_q^q \max_{w\in\mathrm{ant}(p_\ell)} \|\Phi^{\to w}(\theta')\|_q^q\right).
\end{aligned}
$$

This proves Equation (B.5) for v and concludes the induction. □

Proof of Lemma B.3.1. For every neuron v, define $f(v) := \ell$ such that neuron v belongs to the output neurons of matrix M_ℓ. By Lemma B.3.2 with $q = 1$, we have for every

neuron v

$$\|\Phi^{\to v}(\theta) - \Phi^{\to v}(\theta')\|_1$$

$$\leqslant \max_{p\in\mathcal{P}^{\to v}} \sum_{\ell=1}^{\mathtt{length}(p)} \left(\prod_{k=\ell+1}^{\mathtt{length}(p)} \underbrace{\|\theta^{\to p_k}\|_1}_{\substack{\leqslant \|M_{f(p_k)}\|_{1,\infty} \\ \leqslant R}} \right) \left(\underbrace{|b_{p_\ell} - b'_{p_\ell}|}_{=0 \text{ (no biases)}} + \underbrace{\|\theta^{\to p_\ell} - (\theta')^{\to p_\ell}\|_1}_{\leqslant |\operatorname{ant}(p_\ell)| \|\theta-\theta'\|_\infty \leqslant W\|\theta-\theta'\|_\infty} \max_{u\in\operatorname{ant}(p_\ell)} \|\Phi^{\to u}(\theta')\|_1 \right) \tag{B.6}$$

$$\leqslant W\|\theta - \theta'\|_\infty \max_{p\in\mathcal{P}^{\to v}} \sum_{\ell=1}^{\mathtt{length}(p)} R^{\mathtt{length}(p)-\ell} \max_{u\in\operatorname{ant}(p_\ell)} \|\Phi^{\to u}(\theta')\|_1. \tag{B.7}$$

with the convention that an empty sum and product are respectively equal to zero and one. Consider $\theta' = 0$. It holds $\|\Phi^{\to u}(\theta')\|_1 = 0$ for every $u \notin N_{\text{in}}$, and $\|\Phi^{\to u}(\theta')\|_1 = 1$ for input neurons u (Definition 2.3.3). Therefore, we have:

$$\max_{u\in\operatorname{ant}(p_\ell)} \|\Phi^{\to u}(\theta')\|_1 = \mathbb{1}_{\operatorname{ant}(p_\ell)\cap N_{\text{in}}\neq\emptyset} = \mathbb{1}_{\ell=1 \text{ and } p_0\in N_{\text{in}}}. \tag{B.8}$$

Specializing Equation (B.6) to $\theta' = 0$ and using Equation (B.8) yields

$$\begin{aligned}\|\Phi^{\to v}(\theta)\|_1 &\leqslant \max_{p\in\mathcal{P}^{\to v}} \sum_{\ell=1}^{\mathtt{length}(p)} \left(\prod_{k=\ell+1}^{\mathtt{length}(p)} R \right) \underbrace{\|\theta^{\to p_\ell}\|_1}_{\substack{\leqslant \|M_{f(p_\ell)}\|_{1,\infty} \\ \leqslant R}} \underbrace{\max_{u\in\operatorname{ant}(p_\ell)} \|\Phi^{\to u}(\theta')\|_1}_{=\mathbb{1}_{\ell=1 \text{ and } p_0\in N_{\text{in}}}} \\ &= \max_{p\in\mathcal{P}^{\to v}: p_0\in N_{\text{in}}} R^{\mathtt{length}(p)}\end{aligned} \tag{B.9}$$

Since the network is layered, every neuron $u \in \operatorname{ant}(p_\ell)$ is at distance $\ell - 1$ from the input neurons so every $p' \in \mathcal{P}^{\to u}$ is of length $\ell - 1$. We deduce using Equation (B.7), Equation (B.9) for θ' and u:

$$\begin{aligned}\|\Phi^{\to v}(\theta) - \Phi^{\to v}(\theta')\|_1 &\leqslant W\|\theta - \theta'\|_\infty \max_{p\in\mathcal{P}^{\to v}} \sum_{\ell=1}^{\mathtt{length}(p)} R^{\mathtt{length}(p)-\ell} \underbrace{\max_{u\in\operatorname{ant}(p_\ell)} \max_{p'\in\mathcal{P}^{\to u}: p'_0\in N_{\text{in}}} R^{\mathtt{length}(p')}}_{=R^{\ell-1}} \\ &= W\|\theta - \theta'\|_\infty \max_{p\in\mathcal{P}^{\to v}} \underbrace{\sum_{\ell=1}^{\mathtt{length}(p)} R^{\mathtt{length}(p)-1}}_{\leqslant LR^{L-1}} \\ &\leqslant LWR^{L-1}\|\theta - \theta'\|_\infty.\end{aligned}$$

We get the result:

$$\begin{aligned}\|\Phi(\theta) - \Phi(\theta')\|_1 &= \sum_{v\in N_{\text{out}}\setminus N_{\text{in}}} \|\Phi^{\to v}(\theta) - \Phi^{\to v}(\theta')\|_1 \\ &\leqslant |N_{\text{out}} \setminus N_{\text{in}}| \cdot LWR^{L-1}\|\theta - \theta'\|_\infty \\ &\leqslant LW^2R^{L-1}\|\theta - \theta'\|_\infty.\end{aligned}$$

□

Corollary B.3.1. *[Gonon et al., 2023a, Theorem III.1] Consider a LFCN with $L \geqslant 1$ affine layers, corresponding to functions*

$$R_\theta(x) = M_L \,\mathrm{ReLU}(M_{L-1} \dots \mathrm{ReLU}(M_1 x))$$

with each M_ℓ denoting a matrix, where $\theta = (M_1, \dots, M_L)$ and matrices $M_\ell \in \mathbb{R}^{d_{\ell+1} \times d_\ell}$. Recall that $\|M\|_{1,\infty}$ is the maximum L^1 norm of a row of M. Define the width $W := \max_\ell d_\ell$. For θ, θ', define $R := \max_\ell(\|M_\ell\|_{1,\infty}, \|M'_\ell\|_{1,\infty})$.

Theorem 3.4.1 implies for every input x:

$$\|R_\theta(x) - R_{\theta'}(x)\|_1 \leqslant \max(\|x\|_\infty, 1) 2LW^2 R^{L-1} \|\theta - \theta'\|_\infty.$$

Proof. Lemma B.3.1 shows that for every θ, θ', we have

$$\|\Phi(\theta) - \Phi(\theta')\|_1 \leqslant LW^2 R^{L-1} \|\theta - \theta'\|_\infty.$$

Using the Lipschitzness in θ proved in Corollary 3.4.1 for $q = 1$, we deduce that as soon as θ, θ' satisfy $\theta_i \theta'_i \geqslant 0$ for every parameter coordinate i, then for every input x:

$$\|R_\theta(x) - R_{\theta'}(x)\|_1 \leqslant \max(\|x\|_\infty, 1) LW^2 R^{L-1} \|\theta - \theta'\|_\infty. \tag{B.10}$$

Now, consider general parameters θ and θ'. Define θ^{inter} to be such that for every parameter coordinate i:

$$\theta_i^{\mathrm{inter}} = \begin{cases} \theta'_i & \text{if } \theta_i \theta'_i \geqslant 0, \\ 0 & \text{otherwise.} \end{cases}$$

By definition, it holds for every parameter coordinate i: $\theta_i^{\mathrm{inter}} \theta_i \geqslant 0$ and $\theta_i^{\mathrm{inter}} \theta'_i \geqslant 0$ so we can apply Equation (B.10) to the pairs $(\theta, \theta^{\mathrm{inter}})$ and $(\theta^{\mathrm{inter}}, \theta')$ to get:

$$\begin{aligned} \|R_\theta(x) - R_{\theta'}(x)\|_1 &\leqslant \|R_\theta(x) - R_{\theta^{\mathrm{inter}}}(x)\|_1 + \|R_{\theta^{\mathrm{inter}}}(x) - R_{\theta'}(x)\|_1 \\ &\leqslant \max(\|x\|_\infty, 1) LW^2 R^{L-1} \left(\|\theta - \theta^{\mathrm{inter}}\|_\infty + \|\theta^{\mathrm{inter}} - \theta'\|_\infty \right). \end{aligned}$$

It remains to see that $\|\theta - \theta^{\mathrm{inter}}\|_\infty + \|\theta^{\mathrm{inter}} - \theta'\|_\infty 2\|\theta - \theta'\|_\infty$. Consider a parameter coordinate i.

If $\theta_i \theta'_i \geqslant 0$ then $\theta_i^{\mathrm{inter}} = \theta'_i$ and:

$$|\theta_i - \theta'_i| = |\theta_i - \theta_i^{\mathrm{inter}}| + |\theta_i^{\mathrm{inter}} - \theta'_i|.$$

Otherwise, $\theta_i^{\mathrm{inter}} = 0$ and:

$$\begin{aligned} |\theta_i - \theta'_i| &= |\theta_i| + |\theta'_i| \\ &= |\theta_i - \theta_i^{\mathrm{inter}}| + |\theta_i^{\mathrm{inter}} - \theta'_i|. \end{aligned}$$

This implies $\|\theta - \theta^{\mathrm{inter}}\|_\infty = \max_i |\theta_i - \theta_i^{\mathrm{inter}}| \leqslant \max_i |\theta_i - \theta_i^{\mathrm{inter}}| + |\theta_i^{\mathrm{inter}} - \theta'_i| = \|\theta - \theta'\|_\infty$ and similarly $\|\theta^{\mathrm{inter}} - \theta'\|_\infty \leqslant \|\theta - \theta'\|_\infty$. This yields the desired result:

$$\|R_\theta(x) - R_{\theta'}(x)\|_1 \leqslant \max(\|x\|_\infty, 1) 2LW^2 R^{L-1} \|\theta - \theta'\|_\infty. \qquad \square$$

Appendix C

Supplemental material for Chapter 4

C.1 Proof of Lemma 4.3.1

In this section, I prove the version of Dudley's inequality that I stated in Lemma 4.3.1. This is classical, and is included only for completeness. The proof is based on the sub-Gaussianity of the Rademacher process and the application of the Dudley's inequality to this process [Van Handel, 2014, Corollary 5.25].

Proof of Lemma 4.3.1. **1st step: sub-Gaussianity.**

Consider the characterization of sub-Gaussianity given in Definition 5.20 of Van Handel [2014]: a real random process $(N_t)_{t\in T}$ is sub-Gaussian on the pseudo-metric space (T,d) (recall that a pseudo-metric does not necessarily separate points, that is $d(s,t)=0$ does not imply $s=t$) if it is *centered* and if:

$$\mathbb{E}\left(\exp\left(\lambda(N_t-N_s)\right)\right)\leqslant\exp\left(\frac{\lambda^2 d(s,t)^2}{2}\right),\forall\lambda>0,\forall s,t\in T.$$

Consider iid Rademacher variables $\varepsilon=(\varepsilon_{i,j})_{\substack{i=1,\dots,n\\ j=1,\dots,d_{\text{out}}}}$ independent of the samples $S=(x_i)_{i=1}^n$, meaning $\mathbb{P}(\varepsilon_{i,j}=1)=\mathbb{P}(\varepsilon_{i,j}=-1)=1/2$. Recall the notation $f(S)=(f(x_i))_{i=1}^n$. Consider the real random process $N=(N_f)_{f\in\mathcal{F}}$ defined by

$$N_f=\langle\varepsilon,f(S)\rangle\,.$$

By definition of the Rademacher complexity (Equation (4.9)), we have:

$$\mathcal{R}(\mathcal{F},\mu_x)=\mathbb{E}_{S,\varepsilon}\left(\sup_{f\in\mathcal{F}}N_f\right).$$

We now establish that conditionally on the samples S, the random process N is sub-Gaussian on the pseudo-metric space $(\mathcal{F},d_S)$ (where d_S is defined in Equation (4.16)). The process is centered since $\mathbb{E}\left(N_f|S\right)=\langle\mathbb{E}(\varepsilon),f(S)\rangle=0$ as S and ε are independent. Now, consider $f,g\in\mathcal{F}$, $(i,j)\in[\![n]\!]\times[\![d_{\text{out}}]\!]$ and denote by $d_{i,j}=f(x_i)_j-g(x_i)_j$. For any $t>0$, it holds $\frac{1}{2}(e^t+e^{-t})\leqslant e^{t^2/2}$ so for every $\lambda>0$:

$$\mathbb{E}\left(\exp\left(\lambda\varepsilon_{i,j}d_{i,j}\right)|S\right)=\frac{1}{2}\left(\exp(\lambda d_{i,j})+\exp(-\lambda d_{i,j})\right)\leqslant\exp\left(\frac{\lambda^2 d_{i,j}^2}{2}\right).$$

Thus, we have

$$\mathbb{E}\left(\exp\left(\lambda(N_f - N_g)\right)|S\right) = \prod_{(i,j)\in[\![n]\!]\times[\![d_{\text{out}}]\!]} \mathbb{E}\left(\exp\left(\lambda\varepsilon_{i,j}d_{i,j}\right)|S\right) \leqslant \exp\left(\frac{\lambda^2 d_S(f,g)^2}{2}\right).$$

This shows the claim about the sub-Gaussianity of N.

2nd step: Dudley's inequality.

Using Dudley's integral inequality [Van Handel, 2014, Corollary 5.25] conditionally on S yields almost surely:

$$\mathbb{E}_\varepsilon \sup_{f\in\mathcal{F}} N_f \leqslant 12\int_0^\infty \sqrt{\ln \mathcal{N}(\mathcal{F}, d_S, t)}dt$$

where $\mathbb{E}_\varepsilon$ denotes the expectation conditioned on everything (here S) except ε. Putting the steps together, we get the desired result:

$$\mathcal{R}(\mathcal{F}, \mu_x) = \mathbb{E}_{S,\varepsilon}\left(\sup_{f\in\mathcal{F}} N_f\right) \leqslant 12\mathbb{E}_S\left(\int_0^\infty \sqrt{\ln \mathcal{N}(\mathcal{F}, d_S, t)}dt\right).$$

□

C.2 Proof of Theorem 4.3.1, with possible weight-sharing

In this section, I prove Theorem 4.3.1, allowing for possible weight-sharing. I start by motivating the consideration of weight-sharing, then formally introduce weight-sharing and finally the weight-sharing version of Theorem 4.3.1.

C.2.1 Motivation for weight-sharing

When covering a set in dimension d, the covering numbers typically grow exponentially with d. In our case, we want to cover $(\Phi(\Theta), \|\cdot\|_1)$. This is a set whose ambient algebraic dimension is the number of paths, but the actual degrees of freedom must be at most the number of parameters (dimension of Θ), which is much less in general as already discussed at the beginning of Section 3.1. Moreover, in many practical cases of interest, Θ has often weights that are shared, i.e., constrained to be equal. This again reduces the possible degrees of freedom. Is it possible to bound these covering numbers exponentially in $d :=$ the number of free parameters, taking into account possible weight-sharing? I start with an example that shows that this is indeed possible at least in simple situations.

Example C.2.1. *Consider the model $R_\theta : x \in \mathbb{R}^d \mapsto W\,\mathrm{ReLU}(W^T x) \in \mathbb{R}^d$ with $\theta = (W, W^T)$, $W = (w_1 \dots w_d) \in \mathbb{R}^{d\times d}$, and each column w_i being in $\mathbb{R}^d$. In this case, there are $2d^2$ coordinates in θ, but only d^2 of them are free since there are two copies of W.*

The path-lifting is $\Phi(\theta) = (w_i \otimes w_i)_{i=1,\dots,d} \in \mathbb{R}^{d^3}$ (flattened) where $u \otimes v = uv^T$ is the tensor product of vectors u and v. Consider $r > 0$ and $\Theta = \Theta(r) := \{\theta \in \mathbb{R}^G, \|\Phi(\theta)\|_1 \leqslant$

$r\}$. *For parameters* $\theta = (W, W^T)$, *its normalized version* $\mathtt{N}(\theta)$, *defined as the output of Algorithm 3.2.1 for* $q = 1$, *satisfies*

$$\begin{aligned}\mathtt{N}(\theta) &= (\mathtt{N}(W), \mathtt{N}(W^T)),\\ \mathtt{N}(W) &= (\frac{w_1}{\|w_1\|_1} \dots \frac{w_d}{\|w_d\|_1}),\\ \mathtt{N}(W^T) &= (\|w_1\|_1^2 \frac{w_1}{\|w_1\|_1} \dots \|w_d\|_1^2 \frac{w_d}{\|w_d\|_1})^T.\end{aligned}$$

Fix the parameters θ *and* $t \in (0, \min(12, r)]$. *Consider the problem of finding* θ' *such that* $\|\Phi(\theta) - \Phi(\theta')\|_1 \leqslant t$. *Consider a* t*-covering of the unit sphere in dimension* d *for the* ℓ^1*-norm of cardinal at most equal to* $(12/t)^{d-1}$ *(see the end of Appendix C.2.3 for the existence of such a covering). For every* $i = 1, \dots, d$, *choose* u_i *in this covering in such a way that* $\|\mathtt{N}(w_i) - u_i\|_1 \leqslant t$ *where* I *denote* $\mathtt{N}(w) := \frac{w}{\|w\|_1}$ *for any vector* w. *Consider also* $r_i = \sqrt{\lfloor \|w_i\|_1^2 / rt \rfloor \, t}$. *Since* $\|\Phi(\theta)\|_1 = \sum_{i=1}^d \|w_i\|_1^2 \leqslant r$, *we have* $\|w_i\|_1^2 / r \leqslant 1$ *and there are at most* $\left\lfloor \frac{1}{t} \right\rfloor + 1 \leqslant \frac{12}{t}$ *possible values for* r_i *if we further restrict* $t \in (0, \min(11, r))$. *Define* $w_i' := r_i u_i$. *This results in at most* $(\frac{12}{t})^d$ *possible values for* w_i'. *Since* θ' *is built from* d *vectors* w_i' *that can be chosen independently of each other, there are at most* $\prod_{i=1}^d (\frac{12}{t})^d = (\frac{12}{t})^{d^2}$ *choices for* θ'. *Since* $\mathtt{N}(w_i') = u_i$, *it holds that* $\|\mathtt{N}(w_i) - \mathtt{N}(w_i')\|_1 \leqslant t$. *Moreover, we have* $|r_i^2 - \|w_i\|_1^2| \leqslant rt$. *We deduce that:*

$$\begin{aligned}\|\Phi(\theta) - \Phi(\theta')\|_1 &= \sum_{i=1}^d \|w_i \otimes w_i - w_i' \otimes w_i'\|_1 = \sum_{i=1}^d \left\| \|w_i\|_1^2 \mathtt{N}(w_i) \otimes \mathtt{N}(w_i) - \|w_i'\|_1^2 \mathtt{N}(w_i') \otimes \mathtt{N}(w_i') \right\|_1\\
&\leqslant \sum_{i=1}^d \left(\|w_i\|_1^2 \|\mathtt{N}(w_i) \otimes (\mathtt{N}(w_i) - \mathtt{N}(w_i'))\|_1 + \left\| (\|w_i\|_1^2 \mathtt{N}(w_i) - \|w_i'\|_1^2 \mathtt{N}(w_i')) \otimes \mathtt{N}(w_i') \right\|_1 \right)\\
&= \sum_{i=1}^d \left(\|w_i\|_1^2 \underbrace{\|\mathtt{N}(w_i)\|_1}_{=1} \|\mathtt{N}(w_i) - \mathtt{N}(w_i')\|_1 + \left\| \|w_i\|_1^2 \mathtt{N}(w_i) - \|w_i'\|_1^2 \mathtt{N}(w_i') \right\|_1 \underbrace{\|\mathtt{N}(w_i')\|_1}_{=1} \right)\\
&= \sum_{i=1}^d \left(\|w_i\|_1^2 \|\mathtt{N}(w_i) - \mathtt{N}(w_i')\|_1 + \left\| \|w_i\|_1^2 \mathtt{N}(w_i) - \|w_i'\|_1^2 \mathtt{N}(w_i') \right\|_1 \right)\\
&= \sum_{i=1}^d \left(\|w_i\|_1^2 \|\mathtt{N}(w_i) - \mathtt{N}(w_i')\|_1 + \left\| \|w_i\|_1^2 \mathtt{N}(w_i) - \|w_i\|_1^2 \mathtt{N}(w_i') + \|w_i\|_1^2 \mathtt{N}(w_i') - \|w_i'\|_1^2 \mathtt{N}(w_i') \right\|_1 \right)\\
&\leqslant \sum_{i=1}^d \left(\|w_i\|_1^2 \|\mathtt{N}(w_i) - \mathtt{N}(w_i')\|_1 + \|w_i\|_1^2 \|\mathtt{N}(w_i) - \mathtt{N}(w_i')\|_1 + \left| \|w_i\|_1^2 - \|w_i'\|_1^2 \right| \underbrace{\|\mathtt{N}(w_i')\|_1}_{=1} \right)\\
&= \sum_{i=1}^d \left(2\|w_i\|_1^2 \underbrace{\|\mathtt{N}(w_i) - \mathtt{N}(w_i')\|_1}_{\leqslant t} + \underbrace{\left| \|w_i\|_1^2 - \|w_i'\|_1^2 \right|}_{\leqslant rt} \right) \leqslant \underbrace{\left(\sum_{i=1}^d \|w_i\|_1^2 \right)}_{\leqslant r} 2t + drt \leqslant (d+2)rt.\end{aligned}$$

This shows that if we replace t *by* $\frac{t}{(d+2)r}$, *we get a* t*-covering of* $(\Phi(\Theta), \|\cdot\|_1)$ *of size at most equal to* $\left(\frac{12(d+2)r}{t}\right)^{d^2}$. **In this situation, when** $t > 0$ **goes to zero, the covering number essentially grows exponentially with** d^2**, the number of free coordinates, rather than with** $2d^2$**, the number of total coordinates.**

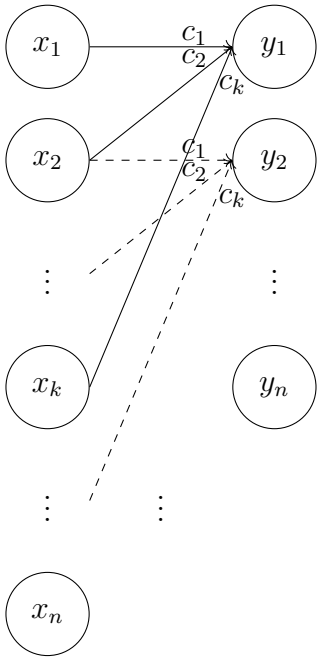


Figure C.1: Illustration of a convolutional circular layer with kernel size k as described in Example C.2.2. The connections corresponding to the first row of the matrix C are drawn as plain arrows, the ones corresponding to the second row are drawn as dashed arrows.

In the previous example, some weights are shared across *successive* layers of the networks. In contrast, most practical application share weights in the same (convolutional) layer. In particular, a path, and therefore a coordinate of $\Phi(\theta)$, cannot contain several copies of a same weight, in contrast to the previous example. I now prove that such a specific type of weight-sharing can be finely taken into account when bounding covering numbers, which already covers widely used convolutional layers.

C.2.2 Formal definition of weight-sharing

I now introduce the notion of a *directed regular partition* of a graph, which will allow me to formally define weight-sharing in a general setting. In short, a directed regular partition defines sets of neurons that are not connected by edges in the graph, and that can share biases and incoming weights as they have the same number of antecedents (incoming weights).

Definition C.2.1. *Consider a DAG $G = (N, E)$. A partition $N = \cup_{\ell=0}^{L} N_\ell$ of the neurons is said to be* directed *if for every $k \leqslant \ell$, we have $E \cap (N_\ell \times N_k) = \emptyset$ (no edge going from N_ℓ to N_k). It is said* regular *if for every $k < \ell$, every $u, v \in N_\ell$, it holds $|\operatorname{ant}(u) \cap N_k| = |\operatorname{ant}(v) \cap N_k|$ (same number of antecedents in N_k).*

Example C.2.2.

- *Every DAG admits at least one directed and regular partition. Indeed, consider any topological sorting $v_1, \dots, v_L$ of the neurons. The partition defined by $N_\ell := \{v_\ell\}$ for every $\ell = 1, \dots, L$ is both directed and regular.*

- *Consider a graph with a single (circular) convolutional layer with kernel size k as in Figure C.1, corresponding to a circular matrix*

$$C = \begin{pmatrix} c_1 & c_2 & \cdots & c_k & 0 & \dots & 0 \\ 0 & c_1 & \ddots & & c_k & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & & \ddots & 0 \\ \vdots & & \ddots & \ddots & \ddots & & c_k \\ \vdots & & \dots & \ddots & \ddots & \ddots & \vdots \\ c_3 & \ddots & \dots & & \ddots & c_1 & c_2 \\ c_2 & c_3 & \cdots & 0 & 0 & 0 & c_1 \end{pmatrix}$$

  With $N_0 := \{x_1, \dots, x_n\}$ and $N_1 := \{y_1, \dots, y_n\}$ the sets of input and output neurons of this layer, the partition $N = N_0 \cup N_1$ is directed and regular: by definition of the kernel size, every $u \in N_1$ satisfies $|\operatorname{ant}(u) \cap N_0| = |\operatorname{ant}(u)| = k$.

- *With the previous example, it is easy to see that for a neural network organized in $L+1$ layers of neurons, a directed and regular partition of the neurons is given by $N_0, \dots, N_L$ where N_ℓ is the set of neurons in layer ℓ.*

I now turn to our formal definition of weight-sharing: this allows for sharing biases and incoming weights for the neurons within the same set N_ℓ of a directed regular partition.

Definition C.2.2. *Consider a directed regular partition $N_0, \dots, N_L$ of a graph G. A set of parameters $\Theta \subset \mathbb{R}^G$ associated with G is said to be* weight-sharing compatible *with $N_0, \dots, N_L$ if for every $0 \leqslant k < \ell \leqslant L$, every pair of neurons $u, v \in N_\ell$ shares weights and biases in the following sense:*

- *$b_u = b_v$,*
- *there exists a bijection $\sigma_{uv} : \operatorname{ant}(u) \cap N_k \to \operatorname{ant}(v) \cap N_k$ such that for every $\theta \in \Theta$, every $w \in \operatorname{ant}(u) \cap N_k$, $\theta^{w \to u} = \theta^{\sigma_{uv}(w) \to v}$.*

Example C.2.3. *The set of parameters corresponding to all circular matrices C as in Example C.2.2 is weight-sharing compatible with the partition given in Example C.2.2 in this case.*

It is also possible to not have weight-sharing by considering the trivial directed regular partition given by $N_0 = N_{in}$ and $N_i = \{v_i\}$ for $i = 1, \dots, d$. This defines a fully-connected layers with input neurons N_{in} and output neurons $v_1, \dots, v_d$.

The extreme opposite is to make all the neurons $v_1, \dots, v_d$ sharing their weights by regrouping all of them in a single set of a directed regular partition: $N_0 = N_{in}$ and $N_1 = \{v_1, \dots, v_d\}$. This is the case of the circular convolutional layer above.

Now, consider the normalizing Algorithm 3.2.1 for $q = 1$. The next lemma shows that the normalizing scalar $\lambda_v(\theta)$ for Algorithm 3.2.1 ran on θ is the same for all the neurons in the same set N_ℓ of a directed regular partition with weight-sharing.

Lemma C.2.1. *Consider a set of parameters $\Theta \subset \mathbb{R}^G$ weight-sharing compatible with a directed regular partition $N_0, \dots, N_L$ of a DAG G. It holds:*

$$\lambda_u(\theta) = \lambda_v(\theta), \forall \theta \in \Theta, \forall u, v \in N_\ell \setminus (N_{in} \cup N_{out}), \forall \ell \in [\![0, L]\!].$$

Proof. The proof is by induction on L.

Initialization. For $L = 0$, since the partition is directed, there is no edge going from N_0 to N_0 so all neurons are input ones and the property is trivially true.

Induction. Assume this is true for $L \geqslant 0$ and consider the case $L+1$. Since the partition is directed, the neurons in $N_0, \dots, N_L$ are normalized by Algorithm 3.2.1 in the same way, irrespectively of whether we consider the graph G or its maximal subgraph with neurons restricted to the sets $N_0, \dots, N_L$. This shows the desired property for every $\ell \leqslant L$. It remains to consider $\ell = L+1$. Take $\theta \in \Theta$. We just saw that when normalizing θ with Algorithm 3.2.1, for every $k \leqslant L$, all the neurons in N_k have the same normalization scalar: denote it by $\lambda_k(\theta)$. Denote also $\theta^{N_k \to u} := (\theta^{w \to u})_{w \in \mathrm{ant}(u) \cap N_k}$. Since the partition is directed, and since the neurons are normalized in the order of a topological sorting, Algorithm 3.2.1 normalizes the neurons $u \in N_{L+1}$ only after having normalized all the ones in $N_0, \dots, N_L$. Therefore, when normalizing $u \in N_{L+1}$, we have

$$\lambda_u(\theta) = |b_u| + \sum_{k \leqslant L} \lambda_k(\theta) \|\theta^{N_k \to u}\|_1.$$

Consider $u, v \in N_{L+1}$. Since Θ is weight-sharing compatible, we have $b_u = b_v$, and $\|\theta^{N_k \to u}\|_1 = \|\theta^{N_k \to v}\|_1$. This proves that $\lambda_u(\theta) = \lambda_v(\theta)$ and concludes the induction. □

Since the neurons in a same N_ℓ of a directed regular partition share the same weights before normalization, and have the same normalization scalar according to Lemma C.2.1, they must be normalized in the same way by Algorithm 3.2.1. Therefore, they have the same values after normalization.

Corollary C.2.1. *In the context of Lemma C.2.1, consider $\ell \in [\![0, L]\!]$ and assume that either $N_\ell \cap V = \emptyset$ or that $N_\ell \subset V$ for both $V = N_{in}$ and N_{out}. For $\theta \in \Theta$, denote $\mathtt{N}(\theta)$ its normalized version, obtained as the output of Algorithm 3.2.1 on input θ. It holds for every $u, v \in N_\ell \setminus N_{in}$:*

$$\mathtt{N}(b)_u = \mathtt{N}(b)_v,$$
$$\mathtt{N}(\theta)^{w \to u} = \mathtt{N}(\theta)^{\sigma_{uv}(w) \to v}, \forall w \in \mathrm{ant}(u).$$

Proof. The assumption guarantees that all neurons in N_ℓ are updated in the same way by the normalizing algorithm (Algorithm 3.2.1).

Case $N_\ell \subset N_{in}$. There is nothing to prove.

Case $N_\ell \subset N_{out}$. When u is an output neuron, b_u is not modified by Algorithm 3.2.1 so $\mathtt{N}(b)_u = b_u$. Moreover, for every $w \in \mathrm{ant}(u)$, the last time $\theta^{w \to u}$ is modified is when w is considered in Algorithm 3.2.1, so:

$$\mathtt{N}(\theta)^{w \to u} = \lambda_w(\theta) \theta^{w \to u}.$$

It is easy to conclude using weight-sharing (Definition C.2.2) and Lemma C.2.1.

Case $N_\ell \cap (N_{in} \cup N_{out}) = \emptyset$. All neurons $u \in N \setminus (N_{\mathrm{in}} \cup N_{\mathrm{out}})$ are such that the last time b_u and $\theta^{w \to u}$ ($w \in \mathrm{ant}(u)$) are modified by Algorithm 3.2.1 is when u is being considered in the `for` loop, so it holds:

$$\mathtt{N}(b)_u = \frac{1}{\lambda_u(\theta)} b_u,$$
$$\mathtt{N}(\theta)^{w \to u} = \frac{1}{\lambda_u(\theta)} \theta^{w \to u}.$$

We again conclude using weight-sharing (Definition C.2.2) and Lemma C.2.1. □

C.2.3 Theorem 4.3.3, allowing for possible weight-sharing

I now use that weight-sharing parameters are normalized in the same way to cover the set $(\Phi(\Theta), \|\cdot\|_1)$ for a weight-sharing set of parameters Θ. This is a more general version of Theorem 4.3.3, allowing for possible weight-sharing. The case *without* weight-sharing corresponds to the trivial case of a directed regular partition where every N_ℓ is a singleton.

Theorem C.2.1. *Consider a DAG network G (Definition 2.2.2) and a set of parameters $\Theta \subset \mathbb{R}^G$ weight-sharing compatible (Definition C.2.2) with a directed regular partition $N_0, \dots, N_L$ (Definition C.2.1) of G. Assume that for $V = N_{in}$ and $V = N_{out}$, each N_ℓ is either disjoint from V or is a subset of V. Assume also that $N_{in} \cap N_{out} = \emptyset$. Define L_0 to be the unique integer in $[\![0, L]\!]$ such that $N_{L_0} \subset N_{in}$ and $N_{L_0+1} \cap N_{in} = \emptyset$. For $\ell \in [\![L_0, L]\!]$, denote by k_ℓ the common number of antecedents of all neurons in N_ℓ and define* $\#\text{rescalings} := L - L_0$ *and* $\#\text{params} = \sum_{\ell=L_0}^{L}(k_\ell + 1)$. *Recall that* $D = \max_{p\in\mathcal{P}} \texttt{length}(p)$ *is the depth of the graph and $d_{out} = |N_{out}|$ is the output dimension. Denote by $r := \sup_{\theta\in\Theta} \|\Phi(\theta)\|_1$. It holds:*

$$\mathcal{N}(\Phi(\Theta), \|\cdot\|_1, t) \leqslant 2^{\#\text{rescalings}} \max\left(1, \frac{24\max(D, d_{out})r}{t}\right)^{\#\text{params}-\#\text{rescalings}}$$

where the definition of covering numbers is recalled in Definition 4.3.1.

In order to prove it, I first relate the geometry of $(\Phi(\Theta), \|\cdot\|_1)$ to the geometry of Θ with more classical norms, in the special case of *normalized* parameters, as we already saw that this type of relation is often the tightest for this type of parameters, see, e.g., Theorem 3.3.2.

Remember the definition of q-normalized parameters in Definition 3.2.1: these are parameters $\tilde\theta$ essentially such that $\|\Phi^{\to v}(\theta)\|_q = \left\|\begin{pmatrix}\tilde\theta^{\to v}\\ \tilde b_v\end{pmatrix}\right\|_q \in \{0, 1\}$ for every $v \in N \setminus (N_{\text{out}} \cup N_{\text{in}})$. In this case, the Lipschitz property of $\theta \mapsto \Phi(\theta)$ established in Lemma B.3.2 becomes much simpler.

Lemma C.2.2. *Consider $q \in [1, \infty)$. For every q-normalized parameters θ, θ', it holds:*

$$\|\Phi(\theta) - \Phi(\theta')\|_q^q \leqslant \sum_{v\in N_{out}\setminus N_{in}} |b_v - b'_v|^q + \|\theta^{\to v} - (\theta')^{\to v}\|_q^q$$
$$+ \min\left(\|\Phi(\theta)\|_q^q, \|\Phi(\theta')\|_q^q\right) \max_{p\in\mathcal{P}:p_{\text{end}}\notin N_{in}} \sum_{\ell=1}^{\texttt{length}(p)-1} \left(|b_{p_\ell} - b'_{p_\ell}|^q + \|\theta^{\to p_\ell} - (\theta')^{\to p_\ell}\|_q^q\right). \quad \text{(C.1)}$$

For networks used in practice, it holds $N_{\text{out}} \cap N_{\text{in}} = \emptyset$ so that $N_{\text{out}} \setminus N_{\text{in}}$ is just N_{out}, but the previous result also covers the somewhat pathological case of DAG architectures G where one or more input neurons are also output neurons.

Proof of Lemma C.2.2. Since $\Phi(\theta) = (\Phi^{\to v}(\theta))_{v\in N_{\text{out}}}$, it holds

$$\|\Phi(\theta) - \Phi(\theta')\|_q^q = \sum_{v\in N_{\text{out}}} \|\Phi^{\to v}(\theta) - \Phi^{\to v}(\theta')\|_q^q.$$

By Definition 2.3.3, it holds for every input neuron v: $\Phi^{\to v}(\cdot) = 1$ (empty product). Thus, the sum can be taken over $v \in N_{\text{out}} \setminus N_{\text{in}}$:

$$\|\Phi(\theta) - \Phi(\theta')\|_q^q = \sum_{v \in N_{\text{out}} \setminus N_{\text{in}}} \|\Phi^{\to v}(\theta) - \Phi^{\to v}(\theta')\|_q^q.$$

Besides, observe that many norms appearing in Equation (B.5) are at most one for q-normalized parameters. Indeed, for such parameters it holds for every $u \in N \setminus (N_{\text{in}} \cup N_{\text{out}})$: $\|\theta^{\to u}\|_q^q \leqslant 1$ (Definition 3.2.1). As a consequence, for $p \in \mathcal{P}$ and any $\ell \in [\![0, \mathtt{length}(p) - 1]\!]$ we have:

$$\prod_{k=\ell+1}^{\mathtt{length}(p)} \|\theta^{\to p_k}\|_q^q = \left(\prod_{k=\ell+1}^{\mathtt{length}(p)-1} \underbrace{\|\theta^{\to p_k}\|_q^q}_{\leqslant 1} \right) \|\theta^{\to p_{\text{end}}}\|_q^q \leqslant \|\theta^{\to p_{\text{end}}}\|_q^q.$$

Moreover, for q-normalized parameters θ and $u \notin N_{\text{out}}$, it also holds $\|\Phi^{\to u}(\theta)\|_q^q \leqslant 1$ (Definition 3.2.1). Thus, Lemma B.3.2 via Equation (B.5) implies for any $v \in N_{\text{out}}$, and any q-normalized parameters θ and θ':

$$\begin{aligned}
&\|\Phi^{\to v}(\theta) - \Phi^{\to v}(\theta')\|_q^q \\
&\leqslant |b_v - b'_v|^q + \|\theta^{\to v} - (\theta')^{\to v}\|_q^q + \|\theta^{\to v}\|_q^q \max_{p \in \mathcal{P}^{\to v}} \sum_{\ell=1}^{\mathtt{length}(p)-1} \left(|b_{p_\ell} - b'_{p_\ell}|^q + \|\theta^{\to p_\ell} - (\theta')^{\to p_\ell}\|_q^q \right).
\end{aligned}$$

Thus, we get:

$$\begin{aligned}
&\|\Phi(\theta) - \Phi(\theta')\|_q^q \\
&= \sum_{v \in N_{\text{out}} \setminus N_{\text{in}}} \|\Phi^{\to v}(\theta) - \Phi^{\to v}(\theta')\|_q^q \\
&\leqslant \sum_{v \in N_{\text{out}} \setminus N_{\text{in}}} \left(|b_v - b'_v|^q + \|\theta^{\to v} - (\theta')^{\to v}\|_q^q \right) \\
&+ \sum_{v \in N_{\text{out}} \setminus N_{\text{in}}} \|\theta^{\to v}\|_q^q \max_{p \in \mathcal{P}^{\to v}} \sum_{\ell=1}^{\mathtt{length}(p)-1} \left(|b_{p_\ell} - b'_{p_\ell}|^q + \|\theta^{\to p_\ell} - (\theta')^{\to p_\ell}\|_q^q \right) \\
&\leqslant \sum_{v \in N_{\text{out}} \setminus N_{\text{in}}} \left(|b_v - b'_v|^q + \|\theta^{\to v} - (\theta')^{\to v}\|_q^q \right) \\
&+ \left(\sum_{v \in N_{\text{out}} \setminus N_{\text{in}}} \|\theta^{\to v}\|_q^q \right) \max_{p \in \mathcal{P} : p_{\text{end}} \notin N_{\text{in}}} \sum_{\ell=1}^{\mathtt{length}(p)-1} \left(|b_{p_\ell} - b'_{p_\ell}|^q + \|\theta^{\to p_\ell} - (\theta')^{\to p_\ell}\|_q^q \right).
\end{aligned}$$

It remains to use that $\sum_{v \in N_{\text{out}} \setminus N_{\text{in}}} \|\theta^{\to u_v}\|_q^q \leqslant \|\Phi(\theta)\|_q^q$ for q-normalized parameters θ to conclude that:

$$\begin{aligned}
\|\Phi(\theta) - \Phi(\theta')\|_q^q \leqslant &\sum_{v \in N_{\text{out}} \setminus N_{\text{in}}} \left(|b_v - b'_v|^q + \|\theta^{\to v} - (\theta')^{\to v}\|_q^q \right) \\
&+ \|\Phi(\theta)\|_q^q \max_{p \in \mathcal{P} : p_{\text{end}} \notin N_{\text{in}}} \sum_{\ell=1}^{\mathtt{length}(p)-1} \left(|b_{p_\ell} - b'_{p_\ell}|^q + \|\theta^{\to p_\ell} - (\theta')^{\to p_\ell}\|_q^q \right).
\end{aligned}$$

The term in blue can be replaced by $\min\left(\|\Phi(\theta)\|_q^q, \|\Phi(\theta')\|_q^q \right)$ by repeating the proof with θ and θ' exchanged (everything else is invariant under this exchange). □

We can now prove the covering bound of Theorem C.2.1.

Proof of Theorem C.2.1. For $\theta \in \Theta$, denote $\mathtt{N}(\theta)$ its normalized version obtained as the output of Algorithm 3.2.1. By Lemma 3.2.1, $\Phi(\theta) = \Phi(\mathtt{N}(\theta))$ so for every $\theta, \theta' \in \Theta$:

$$\|\Phi(\theta) - \Phi(\theta')\|_1 = \|\Phi(\mathtt{N}(\theta)) - \Phi(\mathtt{N}(\theta'))\|_1.$$

For every neuron $u \in N \setminus N_{\text{in}}$ and all parameters θ, denote by $\theta(u) := (b_u, \theta^{\to u})$. By Lemma C.2.2 with $q = 1$, we have:

$$\begin{aligned}
&\|\Phi(\mathtt{N}(\theta)) - \Phi(\mathtt{N}(\theta'))\|_1 \\
&\leqslant \sum_{v \in N_{\text{out}} \setminus N_{\text{in}}} |\mathtt{N}(b)_v - \mathtt{N}(b)'_v| + \|\mathtt{N}(\theta)^{\to v} - (\mathtt{N}(\theta'))^{\to v}\|_1 \\
&+ \underbrace{\min\left(\|\Phi(\mathtt{N}(\theta))\|_1, \|\Phi(\mathtt{N}(\theta'))\|_1\right)}_{\leqslant r} \max_{p \in \mathcal{P} : p_{\mathtt{end}} \notin N_{\text{in}}} \sum_{\ell=1}^{\mathtt{length}(p)-1} \left(|\mathtt{N}(b)_{p_\ell} - \mathtt{N}(b)'_{p_\ell}| + \|\mathtt{N}(\theta)^{\to p_\ell} - (\mathtt{N}(\theta'))^{\to p_\ell}\|_1\right) \\
&\leqslant \underbrace{\sum_{v \in N_{\text{out}} \setminus N_{\text{in}}} \|\mathtt{N}(\theta)(v) - \mathtt{N}(\theta')(v)\|_1}_{=:(1)} \\
&+ r \underbrace{\max_{p \in \mathcal{P} : p_{\mathtt{end}} \notin N_{\text{in}}} \sum_{\ell=1}^{\mathtt{length}(p)-1} \|\mathtt{N}(\theta)(p_\ell) - \mathtt{N}(\theta')(p_\ell)\|_1}_{=:(2)}.
\end{aligned}$$

Consider $L_0 \in [\![0, L]\!]$ such that $N_{L_0} \subset N_{\text{in}}$ and $N_{L_0+1} \cap N_{\text{in}} = \emptyset$. The integer L_0 is well defined since every N_ℓ is either disjoint from N_{in} or is a subset of N_{in}, and at least one of them must be disjoint since $N_0, \dots, N_L$ is a partition of the neurons and $N_{\text{in}} \cap N_{\text{out}} = \emptyset$.

For every $\ell \in [\![L_0, L]\!]$, consider an arbitrary $v_\ell \in N_\ell$. By Corollary C.2.1, we have $\mathtt{N}(\theta)(v) = \mathtt{N}(\theta)(v_\ell)$ for every $v \in N_\ell$, every $\ell \in [\![L_0, L]\!]$ and every parameters $\theta \in \Theta$. We get

$$\begin{aligned}
(1) &= \sum_{\ell=0}^{L} \sum_{v \in (N_{\text{out}} \cap N_\ell) \setminus N_{\text{in}}} \|\mathtt{N}(\theta)(v) - \mathtt{N}(\theta')(v)\|_1 \\
&= \sum_{\ell=L_0}^{L} |N_{\text{out}} \cap N_\ell| \|\mathtt{N}(\theta)(v_\ell) - \mathtt{N}(\theta')(v_\ell)\|_1.
\end{aligned}$$

Consider $p \in \mathcal{P}$ and $f : [\![0, \mathtt{length}(p)]\!] \mapsto [\![0, L]\!]$ the function defined by $p_\ell \in N_{f(\ell)}$ for every $\ell \in [\![0, \mathtt{length}(p)]\!]$. Once again using Corollary C.2.1, since $p_\ell \notin N_{\text{in}}$ for $\ell > 0$, we have $\mathtt{N}(\theta)(p_\ell) = \mathtt{N}(\theta)(v_{f(\ell)})$ for every $\ell \in [\![1, \mathtt{length}(p)]\!]$ and every parameters $\theta \in \Theta$. This yields

$$(2) = \max_{p \in \mathcal{P} : p_{\mathtt{end}} \notin N_{\text{in}}} \sum_{\ell=1}^{\mathtt{length}(p)-1} \|\mathtt{N}(\theta)(v_{f(\ell)}) - \mathtt{N}(\theta')(v_{f(\ell)})\|_1.$$

Assume that for every $\ell \in [\![0, L]\!]$, it holds

$$\|\mathtt{N}(\theta)(v_{f(\ell)}) - \mathtt{N}(\theta')(v_{f(\ell)})\|_1 \leqslant \begin{cases} \frac{t}{2d_{\text{out}}} & \text{if } \ell = L, \\ \frac{t}{2Dr} & \text{otherwise.} \end{cases}$$

where I recall that $d_{\text{out}} = |N_{\text{out}}$ is the output dimension and $D = \max_{p\in\mathcal{P}} \texttt{length}(p)$ is the depth of the graph. This implies:

$$(1) = \sum_{\ell=L_0}^{L} |N_{\text{out}} \cap N_\ell| \|\mathtt{N}(\theta)(v_\ell) - \mathtt{N}(\theta')(v_\ell)\|_1 \leqslant \frac{t}{2d_{\text{out}}} \sum_{\ell=L_0}^{L} |N_{\text{out}} \cap N_\ell| \leqslant \frac{t}{2d_{\text{out}}}.$$

Consider $p \in \mathcal{P}$. Since the partition $N_0, \dots, N_L$ is directed and $p_\ell \to p_{\ell+1}$ is an edge, we have $f(k) < f(\ell)$ for every $k < \ell$. In particular, $f(\ell) < f(\texttt{length}(p)) \leqslant L$ for every $\ell \in [\![1, \texttt{length}(p) - 1]\!]$, so we have

$$(2) = \max_{p\in\mathcal{P}:p_{\text{end}}\notin N_{\text{in}}} \sum_{\ell=1}^{\texttt{length}(p)-1} \|\mathtt{N}(\theta)(v_{f(\ell)}) - \mathtt{N}(\theta')(v_{f(\ell)})\|_1 \leqslant \frac{t}{2Dr} \max_{p\in\mathcal{P}:p_{\text{end}}\notin N_{\text{in}}} (\texttt{length}(p)-1) \leqslant \frac{t}{2r}.$$

Therefore, we get:

$$\begin{aligned} \|\Phi(\mathtt{N}(\theta)) - \Phi(\mathtt{N}(\theta'))\|_1 &\leqslant (1) + r(2) \\ &\leqslant \frac{t}{2} + r\frac{t}{2r} = t. \end{aligned}$$

For a neuron $v \notin N_{\text{in}}$, denote $\mathtt{N}(\Theta)(v) := \{\mathtt{N}(\theta)(v), \theta \in \Theta\}$. In terms of covering numbers, we just proved that:

$$\mathcal{N}(\Phi(\theta), \|\cdot\|_1, t) \leqslant \prod_{\substack{\ell\in[\![L_0,L]\!] \\ N_\ell\subset N_{\text{out}}}} \mathcal{N}(\mathtt{N}(\Theta)(v_\ell), \|\cdot\|_1, t/2d_{\text{out}}) \prod_{\substack{\ell\in[\![L_0,L]\!] \\ N_\ell\cap N_{\text{out}}=\emptyset}} \mathcal{N}(\mathtt{N}(\Theta)(v_\ell), \|\cdot\|_1, t/2Dr).$$

We now bound the latter.

Consider $v \in N_{out} \setminus N_{in}$. By Algorithm 3.2.1, it holds $\mathtt{N}(\theta)(v) \leqslant r$ for every $\theta \in \Theta$. In this situation, $\mathtt{N}(\Theta)(v)$ is a subset of the closed ℓ^1-ball $B_{k_v+1}(0, r)$, with $k_v := |\operatorname{ant}(v)|$ (k for kernel size) so

$$\mathcal{N}(\mathtt{N}(\Theta)(v), \|\cdot\|_1, t/2d_{\text{out}}) \leqslant \mathcal{N}(B_{k_v+1}(0, r), \|\cdot\|_1, t/4d_{\text{out}}).$$

It is well known that the covering with respect to $\|\cdot\|_1$ of the closed ball $B_d \subset \mathbb{R}^d$ with center 0 and radius R satisfies [Wainwright, 2019, Lemma 5.7]:

$$\mathcal{N}(B_d, \|\cdot\|_1, t) \leqslant \max\left(1, \frac{3R}{t}\right)^d.$$

Since the partition $N_0, \dots, N_L$ is regular, all neurons in N_ℓ have the same number of antecedents: denote it by k_ℓ. We get:

$$\prod_{\substack{\ell\in[\![L_0,L]\!] \\ N_\ell\subset N_{\text{out}}}} \mathcal{N}(\mathtt{N}(\Theta)(v_\ell), \|\cdot\|_1, t/2d_{\text{out}}) \leqslant \prod_{\substack{\ell\in[\![L_0,L]\!] \\ N_\ell\subset N_{\text{out}}}} \max\left(1, \frac{12d_{\text{out}}r}{t}\right)^{k_\ell}.$$

Case $v \in N \setminus (N_{out} \cup N_{in})$. For every $\theta \in \Theta$, Algorithm 3.2.1 guarantees that $\|\mathtt{N}(()\theta)(v)\|_1 \in \{0, 1\}$ so $\mathtt{N}(\Theta)(v) \subset \{0\} \cup S^{k_v}$ with S^{k_v} the sphere of radius 1 in dimension $k_v + 1$ with respect to $\|\cdot\|_1$. We deduce that a t-covering of $\mathtt{N}(\Theta)(v)$ is given by the union of the null vector and a $t/2$-covering of the sphere S^{k_v}:

$$\mathcal{N}(\mathtt{N}(\Theta)(v), \|\cdot\|_1, t/2Dr) \leqslant 1 + \mathcal{N}(S^{k_v}, \|\cdot\|_1, t/4Dr).$$

The unit sphere S_d in dimension $d+1$ satisfies

$$S_d = f(B_d) \cup g(B_d)$$

where $f(x_1, \dots, x_d) = (x_1, \dots, x_d, 1 - \|x\|_1)$ and $g(x_1, \dots, x_d) = (x_1, \dots, x_d, \|x\|_1 - 1)$. For every $x, \tilde{x} \in \mathbb{R}^d$:

$$\|f(x) - f(\tilde{x})\|_1 = \sum_{i \leqslant d} |x_i - \tilde{x}_i| + |(1 - \|x\|_1) - (1 - \|\tilde{x}\|_1)| \leqslant 2\|x - \tilde{x}\|_1.$$

Thus, the union of the images of a $\frac{t}{2}$-covering of B_d under both f and g is a t-covering of S_d:

$$\mathcal{N}(S_d, \|\cdot\|_1, t) \leqslant 2\mathcal{N}(B_d, \|\cdot\|_1, t/2) \leqslant 2\max\left(1, \frac{6}{t}\right)^d.$$

We deduce that

$$\prod_{\substack{\ell \in [\![L_0, L]\!] \\ N_\ell \cap N_{\text{out}} = \emptyset}} \mathcal{N}(\mathtt{N}(\Theta)(v_\ell), \|\cdot\|_1, t/2Dr) \leqslant \prod_{\substack{\ell \in [\![L_0, L]\!] \\ N_\ell \cap N_{\text{out}} = \emptyset}} 2\max\left(1, \frac{24Dr}{t}\right)^{k_\ell}.$$

Returning to the covering of $\Phi(\Theta)$, we deduce that:

$$\mathcal{N}(\Phi(\Theta), \|\cdot\|_1, t) \leqslant \prod_{\substack{\ell \in [\![L_0, L]\!] \\ N_\ell \subset N_{\text{out}}}} \max\left(1, \frac{12 d_{\text{out}} r}{t}\right)^{k_\ell} \prod_{\substack{\ell \in [\![L_0, L]\!] \\ N_\ell \cap N_{\text{out}} = \emptyset}} 2\max\left(1, \frac{24Dr}{t}\right)^{k_\ell}.$$

Denote $\#\text{rescalings} := L - L_0$ and $\#\text{params} := \sum_{\ell=L_0}^{L}(k_\ell + 1)$. We get the desired result:

$$\mathcal{N}(\Phi(\Theta), \|\cdot\|_1, t) \leqslant 2^{\#\text{rescalings}} \max\left(1, \frac{24\max(D, d_{\text{out}})r}{t}\right)^{\#\text{params} - \#\text{rescalings}}. \qquad \square$$

C.2.4 Rademacher bound, allowing for possible weight-sharing

I now prove a more general version of Theorem 4.3.1, allowing for possible weight-sharing. The case *without* weight-sharing corresponds to the trivial case of a directed regular partition where every N_ℓ is a singleton.

Theorem C.2.2. *Consider a DAG network G (Definition 2.2.2) and a set of parameters $\Theta \subset \mathbb{R}^G$ weight-sharing compatible (Definition C.2.2) with a directed regular partition $N_0, \dots, N_L$ (Definition C.2.1) of G. Consider the associated set of functions $\mathcal{F}_\Theta := \{R_\theta, \theta \in \Theta\}$. Consider n iid input samples $S = (x_i)_{i=1,\dots,n}$ drawn from a distribution μ_x and denote $\sigma = \mathbb{E}_S\left(\sum_{i=1}^n \max(1, \|x_i\|_\infty^2)\right)^{1/2}$. Denote by $r := \sup_{\theta \in \Theta} \|\Phi(\theta)\|_1$. Assume that for $V = N_{in}$ and $V = N_{out}$, each N_ℓ is either disjoint from V or is a subset of V. Assume also that $N_{in} \cap N_{out} = \emptyset$. Define L_0 to be the unique integer in $[\![0, L]\!]$ such that $N_{L_0} \subset N_{in}$ and $N_{L_0+1} \cap N_{in} = \emptyset$. For $\ell \in [\![L_0, L]\!]$, denote by $k_\ell := |\operatorname{ant}(u)| = \sum_{j<\ell} |\operatorname{ant}(u) \cap N_j|$ for $u \in N_\ell$, the common number of antecedents of the neurons in N_ℓ. Define nparams $:= \sum_{\ell=L_0}^{L}(k_\ell + 1)$.*

We have:

$$\mathcal{R}(\mathcal{F}_\Theta, \mu_x) \leqslant 144\sigma \max(D, d_{out})\sqrt{\#\text{params}} \times r, \tag{C.2}$$

where I recall that $D = \max_{p \in \mathcal{P}} \mathtt{length}(p)$ is the depth of the graph and $d_{out} = |N_{out}|$ is the output dimension.

Proof. The proof is literally the same as the one given in the case without weight-sharing (Section 4.3.4) but by using the new covering bound of Theorem C.2.1 instead of Theorem 4.3.3, allowing for possible weight-sharing. The only difference is that the number of parameters #params is now defined as $\sum_{\ell=L_0}^{L}(k_\ell+1)$ and #rescalings $:= L - L_0$ to take into account the weight-sharing. □

C.3 Relevant (and apparently new) contraction lemmas for Theorem 4.4.1

The main result is Lemma C.3.1.

Lemma C.3.1. *Consider finite sets I, W, Z, and for each $z \in Z$, consider a set $T^z \subset (\mathbb{R}^W)^I$. We denote $t = (t_i)_{i\in I} \in T^z$ with $t_i = (t_{i,w})_{w\in W} \in \mathbb{R}^W$. Consider functions $f_{i,z} : \mathbb{R}^W \to \mathbb{R}$ and a finite family $\varepsilon = (\varepsilon_j)_{j\in J}$ of independent identically distributed Rademacher variables, with the index set J that will be clear from the context. Finally, consider a convex and non-decreasing function $g : \mathbb{R} \to \mathbb{R}$. Assume that at least one of the following setting holds.*

Setting 1: scalar input case. *$|W| = 1$ and for every $i \in I$ and $z \in Z$, $f_{i,z}$ is 1-Lipschitz with $f_{i,z}(0) = 0$.*

Setting 2: $*$-max-pooling case. *For every $i \in I$ and $z \in Z$, there is $k_{i,z} \in \mathbb{N}_{>0}$ such that for every $t \in T^z$, $f_{i,z}(t) = t_{(k_{i,z})}$ is the $k_{i,z}$-th largest coordinate of t.*

Then we have:

$$\mathbb{E}\max_{z\in Z}\sup_{t\in T^z} g\left(\sum_{i\in I}\varepsilon_{i,z}f_{i,z}(t_i)\right) \leqslant \mathbb{E}\max_{z\in Z}\sup_{t\in T^z} g\left(\sum_{i\in I, w\in W}\varepsilon_{i,w,z}t_{i,w}\right). \tag{C.3}$$

The scalar input case is a simple application of Theorem 4.12 in Ledoux and Talagrand [1991]. Indeed, for $z \in Z$ and $t \in T^z$, define $s(z,t) \in \mathbb{R}^{I\times Z}$ to be the matrix with coordinates $(i,v) \in I \times Z$ given by $[s(z,t)]_{i,v} = t_i$ if $v = z$, 0 otherwise. Define also $f_{i,v} = f_i$. Since $f_{i,v}(0) = 0$, it holds:

$$\sum_{i\in I}\varepsilon_{i,z}f_i(t_i) = \sum_{i\in I, v\in Z}\varepsilon_{i,v}f_{i,v}([s(z,t)]_{i,v}).$$

If $S := \{s(z,t), z \in Z, t \in T^z\}$ and $J := I \times Z$ then the result claimed in the scalar case reads

$$\mathbb{E}\sup_{s\in S} g\left(\sum_{j\in J}\varepsilon_j f_j(s_j)\right) \leqslant \mathbb{E}\sup_{s\in S} g\left(\sum_{j\in J}\varepsilon_j s_j\right).$$

The latter is true by Theorem 4.12 of Ledoux and Talagrand [1991]. However, I present an additional proof below, which I employ to establish the new scenario involving $*$-max-pooling. This alternative proof closely follows the structure of the proof outlined in Theorem 4.12 of Ledoux and Talagrand [1991]: the beginning of the proof is the same for the scalar case and the $*$-max-pooling case, and then the arguments become specific to each case.

Note that Theorem 4.12 of Ledoux and Talagrand [1991] does not apply for the $*$-max-pooling case because the t_i's are now vectors. The most related result I could find

is a vector-valued contraction inequality [Maurer, 2016] that is known in the specific case where $|Z| = 1$, g is the identity, and for arbitrary 1-Lipschitz functions $f_{i,z}$ such that $f_{i,z}(0) = 0$ (with a different proof, and with a factor $\sqrt{2}$ on the right-hand side). Here, the vector-valued case I am interested in is $f_{i,z} = k_{i,z}$-`pool` and $g = \exp$, which is covered by Lemma C.3.1. I could not find it stated elsewhere.

In the proof of Lemma C.3.1, I reduce to the simpler case where $|Z| = 1$ and $|I| = 1$ that corresponds to the next lemma. Again, the scalar input case is given by Ledoux and Talagrand [1991, Equation (4.20)] while the $*$-max-pooling case is apparently new. This corresponds to the contraction lemma given in Lemma 4.4.1.

Lemma C.3.2. *Consider a finite set W, a set T of elements $t = (t_1, t_2) \in \mathbb{R}^W \times \mathbb{R}$ and a function $f : \mathbb{R}^W \to \mathbb{R}$. Consider also a convex non-decreasing function $F : \mathbb{R} \to \mathbb{R}$ and a family of iid Rademacher variables $(\varepsilon_j)_{j\in J}$ where J will be clear from the context. Assume that we are in one of the two following situations.*

Scalar input case. *f is 1-Lipschitz, satisfies $f(0) = 0$ and has a scalar input ($|W| = 1$).*

$*$-max-pooling case. *There is $k \in \mathbb{N}_{>0}$ such that f computes the k-th largest coordinate of its input.*

Denoting $t_1 = (t_{1,w})_{w\in W}$, it holds:

$$\mathbb{E} \sup_{t\in T} F\left(\varepsilon_1 f(t_1) + t_2\right) \leqslant \mathbb{E} \sup_{t\in T} F\left(\sum_w \varepsilon_{1,w} t_{1,w} + t_2\right).$$

The proof of Lemma C.3.2 is postponed. I now prove Lemma C.3.1.

Proof of Lemma C.3.1. First, because of the Lipschitz assumptions on the f_i's and the convexity of g, everything is measurable and the expectations are well defined.

I prove the result by reducing to the simpler case of Lemma C.3.2. This is inspired by the reduction done in the proof of Ledoux and Talagrand [1991, Theorem 4.12] in the special case of scalar t_i's ($|W| = 1$).

Reduce to the case $|Z| = 1$ by conditioning and iteration. For $z \in Z$, define

$$A_z := \sup_{t\in T^z} g\left(\sum_{i\in I} \varepsilon_{i,z} f_{i,z}(t_i)\right),$$
$$B_z := \sup_{t\in T^z} g\left(\sum_{i\in I, w\in W} \varepsilon_{i,w,z} t_{i,w}\right).$$

Lemma C.3.3 applies since these random variables are independent. Thus, it is enough to prove that for every $c \in [-\infty, \infty)$:

$$\mathbb{E} \max(A_z, c) \leqslant \mathbb{E} \max(B_z, c).$$

Define $F(x) = \max(g(x), c)$. This can be rewritten as (inverting the supremum and the maximum)

$$\mathbb{E} \sup_{t\in T^z} F\left(\sum_{i\in I} \varepsilon_{i,z} f_{i,z}(t_i)\right) \leqslant \mathbb{E} \sup_{t\in T^z} F\left(\sum_{i\in I, w\in W} \varepsilon_{i,w,z} t_{i,w}\right). \tag{C.4}$$

I just reduced to the case where there is a single z to consider, up to the price of replacing g by F. Since g and $x \mapsto \max(x, c)$ are non-decreasing and convex, so is F by composition.

Alternatively, note that I could also have reduced to the case $|Z| = 1$ by defining $S := \{s(z,t), z \in Z, t \in T^z\}$ just as it is done right after the statement of Lemma C.3.1. In order to apply Lemma C.3.2, it remains to reduce to the case $|I| = 1$.

Reduce to the case $|I| = 1$ by conditioning and iteration. Lemma C.3.4 shows that in order to prove Equation (C.4), it is enough to prove that for every $i \in I$ and every subset $R \subset \mathbb{R}^W \times R$, denoting $r = (r_1, r_2) \in \mathbb{R}^W \times \mathbb{R}$, it holds

$$\mathbb{E} \sup_{r \in R} F\left(\varepsilon_{i,z} f_{i,z}(r_1) + r_2\right) \leqslant \mathbb{E} \sup_{r \in R} F\left(\sum_{w \in W} \varepsilon_{i,w,z} r_{1,w} + r_2\right).$$

I just reduced to the case $|I| = 1$ since one can now consider the indices i one by one. The latter inequality is now a direct consequence of Lemma C.3.2. This proves the result. □

Lemma C.3.3. *Consider a finite set Z and independent families of independent real random variables $(A_z)_{z \in Z}$ and $(B_z)_{z \in Z}$. If for every $z \in Z$ and every constant $c \in [-\infty, \infty)$, it holds $\mathbb{E} \max(A_z, c) \leqslant \mathbb{E} \max(B_z, c)$, then*

$$\mathbb{E} \max_{z \in Z} A_z \leqslant \mathbb{E} \max_{z \in Z} B_z.$$

Proof of Lemma C.3.3. The proof is by conditioning and iteration. To prove the result, it is enough to prove that if

$$\mathbb{E} \max_{z \in Z} A_z \leqslant \mathbb{E} \max\left(\max_{z \in Z_1} A_z, \max_{z \in Z_2} B_z\right)$$

for some partition Z_1, Z_2 of Z, with Z_2 possibly empty for the initialization of the induction, then for every $z_0 \in Z_1$:

$$\mathbb{E} \max_{z \in Z} A_z \leqslant \mathbb{E} \max\left(\max_{z \in Z_1 \setminus \{z_0\}} A_z, \max_{z \in Z_2 \cup \{z_0\}} B_z\right),$$

with the convention that the maximum over an empty set is $-\infty$. Indeed, the claim would then come directly by induction on the size of Z_2.

Now, consider an arbitrary partition Z_1, Z_2 of Z, with Z_2 possibly empty, and consider $z_0 \in Z_1$. It is then enough to prove that

$$\mathbb{E} \max\left(\max_{z \in Z_1} A_z, \max_{z \in Z_2} B_z\right) \leqslant \mathbb{E} \max\left(\max_{z \in Z_1 \setminus \{z_0\}} A_z, \max_{z \in Z_2 \cup \{z_0\}} B_z\right). \tag{C.5}$$

Define the random variable $C = \max\left(\max_{z \in Z_1 \setminus \{z_0\}} A_z, \max_{z \in Z_2} B_z\right)$ which may be equal to $-\infty$ when the maximum is over empty sets, and which is independent of A_{z_0} and B_{z_0}. It holds:

$$\max\left(\max_{z \in Z_1} A_z, \max_{z \in Z_2} B_z\right) = \max\left(A_{z_0}, C\right)$$

and

$$\max\left(\max_{z \in Z_1 \setminus \{z_0\}} A_z, \max_{z \in Z_2 \cup \{z_0\}} B_z\right) = \max\left(B_{z_0}, C\right).$$

Equation (C.5) is then equivalent to

$$\mathbb{E} \max(A_{z_0}, C) \leqslant \mathbb{E} \max(B_{z_0}, C)$$

with C independent of A_{z_0} and B_{z_0}. For a constant $c \in [-\infty, \infty)$, denote $A(c) = \mathbb{E}\max(A_{z_0}, c)$ and $B(c) = \mathbb{E}\max(B_{z_0}, c)$. We have:

$$\begin{aligned} \mathbb{E}\max(A_{z_0}, C) &= \mathbb{E}\left(\mathbb{E}\left(\max\left(A_{z_0}, C\right)|A_{z_0}\right)\right) && \text{law of total expectation} \\ &= \mathbb{E}A(C) && \text{independence of } C \text{ and } A_{z_0}. \end{aligned}$$

and similarly $\mathbb{E}\max(B_{z_0}, C) = \mathbb{E}B(C)$. It is then enough to prove that $A(C) \leqslant B(C)$ almost surely. Since $C \in [-\infty, \infty)$, this is true by assumption. This proves the claims. □

Lemma C.3.4. *Consider finite sets I, W and independent families of independent real random variables $(\varepsilon_i)_{i\in I}$ and $(\varepsilon_{i,w})_{i\in I, w\in W}$. Consider functions $f_i : \mathbb{R}^W \to \mathbb{R}$ and $F : \mathbb{R} \to \mathbb{R}$ that are continuous. Assume that for every $i \in I$ and every subset $R \subset \mathbb{R}^W \times \mathbb{R}$, denoting $r = (r_1, r_2) \in R$ with $r_1 = (r_{1,w})_w \in \mathbb{R}^W$ and $r_2 \in \mathbb{R}$ the components of r, it holds*

$$\mathbb{E}\sup_{r\in R} F(\varepsilon_i f_i(r_1) + r_2) \leqslant \mathbb{E}\sup_{r\in R} F(\sum_{w\in W} \varepsilon_{i,w} r_{1,w} + r_2).$$

Consider an arbitrary $T \subset (\mathbb{R}^W)^I$ and for $t = (t_i)_{i\in I} \in T$, denote $t_{i,w}$ the w-th coordinate of $t_i \in \mathbb{R}^W$. It holds:

$$\mathbb{E}\sup_{t\in T} F(\sum_{i\in I} \varepsilon_i f_i(t_i)) \leqslant \mathbb{E}\sup_{t\in T} F(\sum_{i\in I, w\in W} \varepsilon_{i,w} t_{i,w}).$$

Proof of Lemma C.3.4. The continuity assumption on F and the f_i's is only used to make all the considered suprema measurable. The proof goes by conditioning and iteration. For any $J \subset I$, denote ε_J the family that contains both $(\varepsilon_j)_{j\in J}$ and $(\varepsilon_{j,w})_{j\in J, w\in W}$. Define

$$\begin{aligned} h_J(t, \varepsilon_J) &:= \sum_{j\in J} \varepsilon_j f_j(t_j), \\ H_J(t, \varepsilon_J) &:= \sum_{j\in J, w\in W} \varepsilon_{j,w} t_{j,w}, \end{aligned}$$

with the convention that an empty sum is zero. To make notations lighter, if $J = \{j\}$ then I may write h_j and H_j instead of h_J and H_J. I also omit to write the dependence on ε_J as soon as possible. What I want to prove is thus equivalent to

$$\mathbb{E}\sup_{t\in T} F(h_I(t)) \leqslant \mathbb{E}\sup_{t\in T} F(H_I(t)).$$

It is enough to prove that for every partition I_1, I_2 of I, with I_2 possibly empty, if

$$\mathbb{E}\sup_{t\in T} F(h_I(t)) \leqslant \mathbb{E}\sup_{t\in T} F(h_{I_1}(t) + H_{I_2}(t)),$$

then for every $j \in I_1$,

$$\mathbb{E}\sup_{t\in T} F(h_I(t)) \leqslant \mathbb{E}\sup_{t\in T} F(h_{I_1\setminus\{j\}}(t) + H_{I_2\cup\{j\}}(t)).$$

Indeed, the result would then come by induction on the size of I_2. Fix an arbitrary partition I_1, I_2 of I with I_2 possibly empty, and $j \in I_1$. It is then enough to prove that

$$\mathbb{E}\sup_{t\in T} F(h_{I_1}(t) + H_{I_2}(t)) \leqslant \mathbb{E}\sup_{t\in T} F(h_{I_1\setminus\{j\}}(t) + H_{I_2\cup\{j\}}(t)). \tag{C.6}$$

Denote $\varepsilon_{-j} := \varepsilon_{I\setminus\{j\}}$ and $\varphi(t, \varepsilon_{-j}) := h_{I_1\setminus\{j\}}(t, \varepsilon_{I_1\setminus\{j\}}) + H_{I_2}(t, \varepsilon_{I_2}))$. It holds:

$$h_{I_1}(t) + H_{I_2}(t) = h_j(t, \varepsilon_j) + \varphi(t, \varepsilon_{-j})$$

and, writing $\varepsilon_{j,\cdot} = (\varepsilon_{j,w})_{w\in W}$:

$$h_{I_1\setminus\{j\}}(t) + H_{I_2\cup\{j\}}(t) = H_j(t, \varepsilon_{j,\cdot}) + \varphi(t, \varepsilon_{-j}).$$

Consider the measurable functions

$$g(\varepsilon_j, \varepsilon_{-j}) := \sup_{t\in T} F(h_j(t, \varepsilon_j) + \varphi(t, \varepsilon_{-j}))$$

and

$$G(\varepsilon_{j,\cdot}, \varepsilon_{-j}) := \sup_{t\in T} F(H_j(t, \varepsilon_{j,\cdot}) + \varphi(t, \varepsilon_{-j})).$$

Denote Δ the ambiant space of ε_{-j} and consider a constant $\delta \in \Delta$. Define $\hat{g}(\delta) = \mathbb{E}g(\varepsilon_j, \delta)$ and $\hat{G}(\delta) = \mathbb{E}G(\varepsilon_{j,\cdot}, \delta)$. It holds

$$\begin{aligned}
\mathbb{E}\sup_{t\in T} F(h_{I_1}(t) + H_{I_2}(t)) &= \mathbb{E}g(\varepsilon_j, \varepsilon_{-j}) && \text{by definition of } g \\
&= \mathbb{E}\left(\mathbb{E}\left(g(\varepsilon_j, \varepsilon_{-j})|\varepsilon_{-j}\right)\right) && \text{law of total expectation} \\
&= \mathbb{E}\hat{g}(\varepsilon_{-j}) && \text{independence of } \varepsilon_j \text{ and } \varepsilon_{-j}
\end{aligned}$$

and similarly $\mathbb{E}\sup_{t\in T} F(h_{I_1\setminus\{j\}}(t) + H_{I_2\cup\{j\}}(t)) = \mathbb{E}\hat{G}(\varepsilon_{-j})$. Thus, Equation (C.6) is equivalent to $\mathbb{E}\hat{g}(\varepsilon_{-j}) \leqslant \mathbb{E}\hat{G}(\varepsilon_{-j})$. For every $\delta \in \Delta$, we can define $R(\delta) = \{(t_j, \varphi(t, \delta)) \in \mathbb{R}^W \times \mathbb{R}, t \in T\}$ and it holds

$$\hat{g}(\delta) = \mathbb{E}\sup_{r\in R} F(\varepsilon_j f_j(r_1) + r_2)$$

and

$$\hat{G}(\delta) = \mathbb{E}\sup_{r\in R} F(\sum_{w\in} \varepsilon_{j,w} r_{1,w} + r_2).$$

Thus, $\hat{g}(\delta) \leqslant \hat{G}(\delta)$ for every $\delta \in \Delta$ by assumption. This shows the claim. □

Proof of Lemma C.3.2. Recall that we want to prove

$$\mathbb{E}\sup_{t\in T} F\left(\varepsilon_1 f(t_1) + t_2\right) \leqslant \mathbb{E}\sup_{t\in T} F\left(\sum_{w\in W} \varepsilon_{1,w} t_{1,w} + t_2\right). \tag{C.7}$$

Scalar input case. In this case, $|W| = 1$, that is the inputs t_1 are scalar and the result is well-known, see Ledoux and Talagrand [1991, Equation (4.20)].

k-max-pooling case. In this case, f computes the k-th largest coordinate of its input. Computing explicitly the expectation where the only random thing is $\varepsilon_1 \in \{-1, 1\}$, the left-hand side of Equation (C.7) is equal to

$$\frac{1}{2}\sup_{t\in T} F\left(f(t_1) + t_2\right) + \frac{1}{2}\sup_{s\in T} F\left(-f(s_1) + s_2\right).$$

Consider $s, t \in T$. Recall that $s_1, t_1 \in \mathbb{R}^W$. Denote $s_{1,(k)}$ the k-th largest component of vector s_1. The set $\{w \in W : s_{1,w} \leqslant s_{1,(k)}\}$ has at least $|W| - k + 1$ elements, and

$\{w \in W : t_{1,(k)} \leqslant t_{1,w}\}$ has at least k elements, so their intersection is not empty. Consider *any*[1] $w(s,t)$ in this intersection. I am now going to use that both $f(t_1) = t_{1,(k)} \leqslant t_{1,w(s,t)}$ and $-f(s_1) = -s_{1,(k)} \leqslant -s_{1,w(s,t)}$. Even if we are not going to use it, note that this implies $f(t) - f(s) \leqslant t_{1,w(s,t)} - s_{1,w(s,t)}$: we are exactly using an argument that establishes that f is 1-Lipschitz. Since $f(t_1) = t_{1,(k)} \leqslant t_{1,w(s,t)}$ and F is non-decreasing, it holds:

$$\begin{aligned} F\left(f(t_1) + t_2\right) \leqslant & F\left(t_{1,w(s,t)} + t_2\right) \\ \underset{\varepsilon \text{ centered}}{=} & F\left(t_{1,w(s,t)} + \mathbb{E}\left(\sum_{w \neq w(s,t)} \varepsilon_{1,w} t_{1,w}\right) + t_2\right) \\ \underset{\text{Jensen}}{\leqslant} & \mathbb{E}F\left(t_{1,w(s,t)} + \sum_{w \neq w(s,t)} \varepsilon_{1,w} t_{1,w} + t_2\right). \end{aligned}$$

Moreover, $-f(s_1) = -s_{1,(k)} \leqslant -s_{1,w(s,t)}$ so that in a similar way:

$$\begin{aligned} F\left(-f(s_1) + s_2\right) \leqslant & F\left(-s_{1,w(s,t)} + s_2\right) \\ \leqslant & F\left(-s_{1,w(s,t)} + \mathbb{E}\left(\sum_{w \neq w(s,t)} \varepsilon_{1,w} s_{1,w}\right) + s_2\right) \\ \leqslant & \mathbb{E}F\left(-s_{1,w(s,t)} + \sum_{w \neq w(s,t)} \varepsilon_{1,w} s_{1,w} + s_2\right). \end{aligned}$$

At the end, we get

$$\begin{aligned} & \frac{1}{2}F\left(f(t_1) + t_2\right) + \frac{1}{2}F\left(-f(s_1) + s_2\right) \\ & \leqslant \frac{1}{2}\mathbb{E}F\left(t_{1,w(s,t)} + \sum_{w \neq w(s,t)} \varepsilon_{1,w} t_{1,w} + t_2\right) \\ & + \frac{1}{2}\mathbb{E}F\left(-s_{1,w(s,t)} + \sum_{w \neq w(s,t)} \varepsilon_{1,w} s_{1,w} + s_2\right) \\ & \leqslant \frac{1}{2}\mathbb{E}\sup_{r \in T} F\left(r_{1,w(s,t)} + \sum_{w \neq w(s,t)} \varepsilon_{1,w} r_{1,w} + r_2\right) \\ & + \frac{1}{2}\mathbb{E}\sup_{r \in T} F\left(-r_{1,w(s,t)} + \sum_{w \neq w(s,t)} \varepsilon_{1,w} r_{1,w} + r_2\right) \\ & = \mathbb{E}\sup_{r \in T} F\left(\varepsilon_{1,w(s,t)} r_{1,w(s,t)} + \sum_{w \neq w(s,t)} \varepsilon_{1,w} r_{1,w} + r_2\right) \\ & = \mathbb{E}\sup_{r \in T} F\left(\sum_{w} \varepsilon_{1,w} r_{1,w} + r_2\right). \end{aligned}$$

The latter is independent of s, t. Taking the supremum over all $s, t \in T$ yields Equation (C.7) and thus the claim. □

[1] The choice of a specific w has no importance, unlike when defining the activations of k-max-pooling neurons.

C.4 Peeling argument for Theorem 4.4.1

This section proves a new peeling argument used in the proof of Theorem 4.4.1. First, I state a simple lemma that will be used several times.

Lemma C.4.1. *Consider a vector $\varepsilon \in \mathbb{R}^n$ with iid Rademacher coordinates, meaning that $\mathbb{P}(\varepsilon_i = 1) = \mathbb{P}(\varepsilon_i = -1) = 1/2$. Consider a function $g : \mathbb{R} \to \mathbb{R}_{\geqslant 0}$. Consider a set $X \subset \mathbb{R}^n$. It holds:*

$$\mathbb{E}_\varepsilon \sup_{x\in X} g\left(\left|\sum_{i=1}^n \varepsilon_i x_i\right|\right) \leqslant 2\mathbb{E}_\varepsilon \sup_{x\in X} g\left(\sum_{i=1}^n \varepsilon_i x_i\right).$$

Proof of Lemma C.4.1. Since $g \geqslant 0$, it holds $g(|x|) \leqslant g(x) + g(-x)$. Thus

$$\mathbb{E}_\varepsilon \sup_{x\in X} g\left(\left|\sum_{i=1}^n \varepsilon_i x_i\right|\right) \leqslant \mathbb{E}_\varepsilon \sup_{x\in X} g\left(\sum_{i=1}^n \varepsilon_i x_i\right) + \mathbb{E}_\varepsilon \sup_{x\in X} g\left(\sum_{i=1}^n (-\varepsilon_i) x_i\right).$$

Since ε is symmetric, that is $-\varepsilon$ has the same distribution as ε, we deduce that the latter is just $2\mathbb{E}_\varepsilon \sup_{x\in X} g\left(\sum_{i=1}^n \varepsilon_i x_i\right)$. This proves the claim. □

Notations I now fix for all the next results of this section n vectors $x_1, \dots, x_n \in \mathbb{R}^{d_{\text{in}}}$, for some $d_{\text{in}} \in \mathbb{N}_{>0}$. I denote by $x_{i,u}$ the coordinate u of x_i.

For any neural network architecture, recall that $v(\theta, x)$ is the output of neuron v for parameters θ and input x, and $\text{ant}^d(v)$ is the set of neurons u for which there exists a path from u to v of distance d. For a set of neurons V, denote $R_V(\theta, x) = (v(\theta, x))_{v\in V}$.

Introduction to peeling This section shows that some expected sum over output neurons v can be reduced to an expected maximum over $\text{ant}(v)$, and iteratively over an expected maximum over $\text{ant}^d(v)$ for increasing d's. Eventually, the maximum is only over input neurons as soon as d is large enough. I start with the next lemma which is the initialization of the induction over d: it peels off the output neurons v to reduce to their antecedents $\text{ant}(v)$.

Lemma C.4.2. *Consider a neural network architecture as in Definition 2.2.2 with $N_{in} \cap N_{out} = \emptyset$. Consider an associated set Θ of parameters θ such that $\sum_{v\in N_{out}} \|\theta^{\to v}\|_1 + |b_v| \leqslant r$. Consider a family of independent Rademacher variables $(\varepsilon_j)_{j\in J}$ with J that will be clear from the context. Consider a non-decreasing function $g : \mathbb{R} \to \mathbb{R}_{\geqslant 0}$. Consider a new neuron $v_{\texttt{bias}}$ and set by convention $x_{v_{\texttt{bias}}} = 1$ for every input x. It holds*

$$\mathbb{E}_\varepsilon g\left(\sup_{\theta\in\Theta} \sum_{\substack{i=1,\dots,n,\\ v\in N_{out}}} \varepsilon_{i,v} v(\theta, x_i)\right)$$
$$\leqslant \mathbb{E}_\varepsilon g\left(r \max_{v\in N_{out}} \max_{u\in(\text{ant}(v)\cap N_{in})\cup\{v_{\texttt{bias}}\}} \left|\sum_{i=1}^n \varepsilon_{i,v} x_{i,u}\right|\right)$$
$$+ \mathbb{E}_\varepsilon g\left(r \max_{v\in N_{out}} \max_{u\in\text{ant}(v)\setminus N_{in}} \sup_\theta \left|\sum_{i=1}^n \varepsilon_{i,v} u(\theta, x_i)\right|\right)$$

where in the last term if $ant(v) \setminus N_{in} = \emptyset$, by convention $\max_{u\in\text{ant}(v)\setminus N_{in}} = -\infty$, and $g(-\infty) := 0$.

Proof of Lemma C.4.2. Recall that for a set of neurons V, we denote $R_V(\theta, x) = (v(\theta, x))_{v \in V}$. Recall that by Definition 2.2.2, output neurons v have $\rho_v = \mathrm{id}$ so for every $v \in N_{\mathrm{out}}$, $\theta \in \Theta$ and every input x:

$$v(\theta, x) = \left\langle \begin{pmatrix} \theta^{\to v} \\ b_v \end{pmatrix}, \begin{pmatrix} R_{\mathrm{ant}(v)}(\theta, x) \\ 1 \end{pmatrix} \right\rangle.$$

Denote by $v_{\texttt{bias}}$ a new neuron that computes the constant function equal to one ($v_{\texttt{bias}}(\theta, x) = 1$), we get:

$$\begin{aligned}
&\mathbb{E}_\varepsilon g \left(\sup_\theta \sum_{\substack{i=1,\dots,n \\ v \in N_{\mathrm{out}}}} \varepsilon_{i,v} v(\theta, x_i) \right) \\
&= \mathbb{E}_\varepsilon g \left(\sup_\theta \sum_{v \in N_{\mathrm{out}}} \left\langle \begin{pmatrix} \theta^{\to v} \\ b_v \end{pmatrix}, \sum_{i=1}^n \varepsilon_{i,v} \begin{pmatrix} R_{\mathrm{ant}(v)}(\theta, x_i) \\ 1 \end{pmatrix} \right\rangle \right) \\
&\underset{\text{Hölder}}{\leqslant} \mathbb{E}_\varepsilon g \left(\sup_\theta \underbrace{\left(\sum_{v \in N_{\mathrm{out}}} \|\theta^{\to v}\|_1 + |b_v| \right)}_{\leqslant r \text{ by assumption}} \max_{v \in N_{\mathrm{out}}} \left(\left| \sum_{i=1}^n \varepsilon_{i,v} \right|, \max_{u \in \mathrm{ant}(v)} \left| \sum_{i=1}^n \varepsilon_{i,v} u(\theta, x_i) \right| \right) \right) \\
&\leqslant \mathbb{E}_\varepsilon g \left(r \max_{v \in N_{\mathrm{out}}} \max_{u \in \mathrm{ant}(v) \cup \{v_{\texttt{bias}}\}} \sup_\theta \left| \sum_{i=1}^n \varepsilon_{i,v} u(\theta, x_i) \right| \right).
\end{aligned}$$

Everything is non-negative so the maximum over $u \in \mathrm{ant}(v) \cup \{v_{\texttt{bias}}\}$ is smaller than the sum of the maxima over $u \in (\mathrm{ant}(v) \cap N_{\mathrm{in}}) \cup \{v_{\texttt{bias}}\}$ and $u \in \mathrm{ant}(v) \setminus N_{\mathrm{in}}$. Note that when u is an input neuron, it simply holds $u(\theta, x_i) = x_{i,u}$. This proves the result. □

I now show how to peel neurons to reduce the maximum over $\mathrm{ant}^d(v)$ to $\mathrm{ant}^{d+1}(v)$. Later, I will repeat that until the maximum is only on input neurons. Compared to the previous lemma, note the presence of an index $m = 1, \dots, M$ in the maxima. This is because after d steps of peeling (when the maximum over u has been reduced to $u \in \mathrm{ant}^d(v)$), we will have $M = K^{d-1}$ where K is the kernel size. Indeed, the number of copies indexed by m gets multiplied by K after each peeling step. The next lemma is the formal version of Lemma 4.4.2.

Lemma C.4.3. *Consider a neural network architecture with an associated set Θ of parameters θ such that every neuron $v \notin N_{out} \cup N_{in}$ satisfies $\|\theta^{\to v}\|_1 + |b_v| \leqslant 1$. Assume that $b_v = 0$ for every $v \in N_{*\texttt{-pool}}$. Consider a family of independent Rademacher variables $(\varepsilon_j)_{j \in J}$ with J that will be clear from the context. Consider arbitrary $M, d \in \mathbb{N}$ and a convex non-decreasing function $g : \mathbb{R} \to \mathbb{R}_{\geqslant 0}$. Take a symbol v_{bias} which does not correspond to a neuron ($v_{bias} \notin N$) and set by convention $x_{v_{bias}} = 1$ for every input x. Define $P := |\{k \in \mathbb{N}_{>0}, \exists u \in N_{k\texttt{-pool}}\}|$ as the number of different types of $*$-max-pooling neurons in G, and $K := \max_{u \in N_{*\texttt{-pool}}} |\mathrm{ant}(u)|$ the maximal kernel size of the network ($K := 1$ if*

$P = 0$). *It holds:*

$$\mathbb{E}_\varepsilon g\left(\max_{\substack{v\in N_{out},\\ m=1,\dots,M}} \max_{u\in \mathrm{ant}^d(v)\setminus N_{in}} \sup_\theta \left|\sum_{i=1}^n \varepsilon_{i,v,m} u(\theta, x_i)\right|\right)$$
$$\leqslant (3+2P)\mathbb{E}_\varepsilon g\left(\max_{\substack{v\in N_{out},\\ m=1,\dots,KM}} \max_{u\in(\mathrm{ant}^{d+1}(v)\cap N_{in})\cup\{v_{bias}\}} \left|\sum_{i=1}^n \varepsilon_{i,v,m} x_{i,u}\right|\right)$$
$$+ (3+2P)\mathbb{E}_\varepsilon g\left(\max_{\substack{v\in N_{out},\\ m=1,\dots,KM}} \max_{u\in \mathrm{ant}^{d+1}(v)\setminus N_{in}} \sup_\theta \left|\sum_{i=1}^n \varepsilon_{i,v,m} u(\theta, x_i)\right|\right)$$

with similar convention as in Lemma C.4.2 for empty maxima.

Proof. **Step 1: split the neurons depending on their activation function.** In the term that we want to bound from above, the neurons $u \in \mathrm{ant}^d(v) \setminus N_{\mathrm{in}}$ are not input neurons so they compute something of the form $\rho_u(\dots)$ where ρ_u is the activation associated with u, 1-Lipschitz, and satisfies $\rho_u(0) = 0$. The first step of the proof is to get rid of ρ_u using a contraction lemma similar to Theorem 4.12 in Ledoux and Talagrand [1991]. However, here, the function ρ_u depends on the neuron u, what we are taking a maximum over so that classical contraction lemmas do not apply directly. To resolve this first obstacle, I split the neurons according to their activation function. Below, I highlight in orange what is important and/or the changes from one line to another. Denote N_ρ the neurons that have ρ as their associated activation function, and the term with a maximum over all $u \in N_\rho$ is denoted:

$$e(\rho) := \mathbb{E}_\varepsilon g\left(\max_{\substack{v\in N_{\mathrm{out}},\\ m=1,\dots,M}} \max_{u\in(\mathrm{ant}^d(v)\cap N_\rho)\setminus N_{\mathrm{in}}} \sup_\theta \left|\sum_{i=1}^n \varepsilon_{i,v,m} u(\theta, x_i)\right|\right),$$

with the convention $e(\rho) = 0$ if N_ρ is empty. This yields a first bound

$$\mathbb{E}_\varepsilon g\left(\max_{\substack{v\in N_{\mathrm{out}},\\ m=1,\dots,M}} \max_{u\in \mathrm{ant}^d(v)\setminus N_{\mathrm{in}}} \sup_\theta \left|\sum_{i=1}^n \varepsilon_{i,v,m} u(\theta, x_i)\right|\right) \leqslant e(\mathrm{ReLU}) + e(\mathrm{id}) + \sum_k e(k\text{-}\mathtt{pool})$$

where the sum of the right-hand side is on all the $k \in \mathbb{N}_{>0}$ such that there is at least one neuron in $N_{k\text{-}\mathtt{pool}}$. Define $E(\rho)$ to be the same thing as $e(\rho)$ but without the absolute values:

$$E(\rho) := \mathbb{E}_\varepsilon g\left(\max_{\substack{v\in N_{\mathrm{out}},\\ m=1,\dots,M}} \max_{u\in(\mathrm{ant}^d(v)\cap N_\rho)\setminus N_{\mathrm{in}}} \sup_\theta \sum_{i=1}^n \varepsilon_{i,v,m} u(\theta, x_i)\right).$$

Lemma C.4.1 gets rid of the absolute values by paying a factor 2:

$$e(\rho) \leqslant 2E(\rho).$$

I now want to bound each $E(\rho)$.

Step 2: get rid of the $*$-max-pooling and ReLU activation functions. Since the maximal kernel size is K, any $*$-max-pooling neuron u must have at most K antecedents. When a $u \in N_{*\text{-}\mathtt{pool}}$ has less than K antecedents, I artificially add neurons

w to $\mathrm{ant}(u)$ to make it of cardinal K, and I set by convention $\theta^{w\to u}=0$. I also fix an arbitrary order on the antecedents of u and write $\mathrm{ant}(u)_w$ for the antecedent number w, with $R_{\mathrm{ant}(u)_w}$ the function associated with this neuron. For a ReLU or $*$-max-pooling neuron u, define the pre-activation of u to be

$$\mathrm{pre}_u(\theta,x):=\begin{cases}\left\langle\begin{pmatrix}\theta^{\to u}\\ b_u\end{pmatrix},\begin{pmatrix}R_{\mathrm{ant}(u)}(\theta,x)\\ 1\end{pmatrix}\right\rangle & \text{if } u\in N_{\mathrm{ReLU}},\\ \Big(b_u+\theta^{\mathrm{ant}(u)_w\to u}R_{\mathrm{ant}(u)_w}(\theta,x)\Big)_{w=1,\dots,k} & \text{otherwise when } u\in N_{*\texttt{-pool}}.\end{cases}$$

where I recall that $b_u=0$ for $u\in N_{*\texttt{-pool}}$ by assumption. I nevertheless keep b_u in the computation until the point where this assumption is apparently actually needed to continue. Note that the pre-activation has been defined to satisfy $u(\theta,x)=\rho_u(\mathrm{pre}_u(\theta,x))$. When ρ is the ReLU or k-pool, we can thus rewrite $E(\rho)$ in terms of the pre-activations:

$$E(\rho)=\mathbb{E}_\varepsilon g\left(\max_{\substack{v\in N_{\mathrm{out}},\\ m=1,\dots,M}}\max_{u\in(\mathrm{ant}^d(v)\cap N_\rho)\setminus N_{\mathrm{in}}}\sup_\theta\sum_{i=1}^n\varepsilon_{i,v,m}\rho(\mathrm{pre}_u(\theta,x_i))\right).$$

Consider the finite set $Z=\{(v,m),v\in N_{\mathrm{out}},m=1,\dots,M\}$ and for every $z=(v,m)\in Z$, define $T^z=\{(\mathrm{pre}_u(\theta,x_i))_{i=1,\dots,n}:u\in(\mathrm{ant}^d(v)\cap N_\rho)\setminus N_{\mathrm{in}},\theta\in\Theta\}$. An element of T^z will be denoted $t=(t_i)_{i=1}^n\in T^z\subset\mathbb{R}^n$ if $\rho=\mathrm{ReLU}$, and $t=(t_i)_{i=1}^n\in T^z\subset(\mathbb{R}^k)^n$ with $t_i=(t_{i,w})_{w=1}^k\in\mathbb{R}^k$ if $u\in N_{*\texttt{-pool}}$. We can again rewrite $E(\rho)$ as

$$E(\rho)=\mathbb{E}_\varepsilon g\left(\max_{z\in Z}\sup_{t\in T^z}\sum_{i=1}^n\varepsilon_{i,z}\rho(t_i)\right).$$

We now want to get rid of the activation function ρ with a contraction lemma. There is a second difficulty that prevents us from directly applying classical contraction lemmas such as Theorem 4.12 of Ledoux and Talagrand [1991]. It is the presence of a maximum over multiple copies indexed by $z\in Z$ of a supremum that depends on iid families $(\varepsilon_{i,z})_{i=1\dots n}$. Indeed, Theorem 4.12 of Ledoux and Talagrand [1991] only deals with a single copy ($|Z|=1$). This motivates the contraction lemma established for the occasion in Lemma C.3.1. Once the activation functions removed, we can conclude separately for $\rho=\mathrm{ReLU},\mathrm{id}$ and $\rho=k$-pool.

Step 3a: deal with $\rho=k$-pool via rescaling. In the case $\rho=k$-pool, Lemma C.3.1 shows that

$$\begin{aligned}&\mathbb{E}_\varepsilon g\left(\max_{z\in Z}\sup_{t\in T^z}\sum_{i=1}^n\varepsilon_{i,z}k\text{-pool}(t_i)\right)\\ &\leqslant\mathbb{E}_\varepsilon g\left(\max_{z\in Z}\sup_{t\in T^z}\sum_{\substack{i=1,\dots,n,\\ w=1,\dots,K}}\varepsilon_{i,z,w}t_{i,w}\right).\end{aligned}$$

The right-hand side is equal to

$$\mathbb{E}_\varepsilon g\left(\max_{\substack{v\in N_{\mathrm{out}},\\ m=1,\dots,M}}\sup_{\substack{u\in(\mathrm{ant}^d(v)\cap N_{*\texttt{-pool}})\setminus N_{\mathrm{in}},\\ \theta\in\Theta}}\sum_{\substack{i=1,\dots,n,\\ w=1,\dots,K}}\varepsilon_{i,v,m,w}(b_u+\theta^{\mathrm{ant}(u)_w\to u}R_{\mathrm{ant}(u)_w}(\theta,x_i))\right).\quad\text{(C.8)}$$

I now deal with this using the assumption on the norm of incoming weights. Recalling that $b_u = 0$ for every $u \in N_{*\texttt{-pool}}$:

$$\begin{aligned}
&\sum_{\substack{i=1,\dots,n,\\ w=1,\dots,K}} \varepsilon_{i,v,m,w}\Big(\underbrace{b_u}_{=0} + \theta^{\mathrm{ant}(u)_w \to u} R_{\mathrm{ant}(u)_w}(\theta, x_i)\Big) \\
&\qquad = \sum_{w=1,\dots,K} \theta^{\mathrm{ant}(u)_w \to u} \left(\sum_{i=1,\dots,n} \varepsilon_{i,v,m,w} R_{\mathrm{ant}(u)_w}(\theta, x_i) \right) \\
&\qquad \underset{\text{Hölder}}{\leqslant} \underbrace{\|\theta^{\to u}\|_1}_{\leqslant 1 \text{ by assumption}} \max_{w=1,\dots,K} \left| \sum_{i=1,\dots,n} \varepsilon_{i,v,m,w} R_{\mathrm{ant}(u)_w}(\theta, x_i) \right| \\
&\qquad\qquad \underset{\text{decoupling } w \text{ and } \mathrm{ant}(u)_w}{\leqslant} \max_{w \in \mathrm{ant}(u)} \max_{w'=1,\dots,K} \left| \sum_{i=1}^{n} \varepsilon_{i,v,m,w'} w(\theta, x_i) \right|.
\end{aligned}$$

Note for the curious reader that this inequality is the current obstacle when the biases are nonzero since we would end up with $K|b_u| + \|\theta^{\to u}\|_1$ and we could not anymore use that this is $\leqslant 1$.

We deduce that Equation (C.8) is bounded from above by

$$\mathbb{E}_\varepsilon g \left(\max_{\substack{v \in N_{\mathrm{out}},\\ m=1,\dots,M}} \sup_{u \in (\mathrm{ant}^d(v) \cap N_{*\texttt{-pool}}) \setminus N_{\mathrm{in}}, \theta \in \Theta} \max_{w \in \mathrm{ant}(u)} \max_{w'=1,\dots,K} \left| \sum_{i=1}^{n} \varepsilon_{i,v,m,w'} w(\theta, x_i) \right| \right).$$

Instead of having $\varepsilon_{i,v,m,w'}$ with $m = 1, \dots, M$ and $w' = 1, \dots, K$, I re-index it as $\varepsilon_{i,v,m}$ with $m = 1, \dots, KM$. Note also that $u \in (\mathrm{ant}^d(v) \cap N_{*\texttt{-pool}}) \setminus N_{\mathrm{in}}$ and $w \in \mathrm{ant}(u)$ implies $w \in \mathrm{ant}^{d+1}(v)$, so considering a maximum over $w \in \mathrm{ant}^{d+1}(v)$ can only yield something larger. Moreover, we can add a new neuron $v_{\texttt{bias}}$ that computes the constant function equal to one ($v_{\texttt{bias}}(\theta, x) = 1$) and add $v_{\texttt{bias}}$ to the maximum over w. Implementing all these changes, Equation (C.8) is bounded by

$$H := \mathbb{E}_\varepsilon g \left(\max_{\substack{v \in N_{\mathrm{out}},\\ m=1,\dots,KM}} \sup_{w \in \mathrm{ant}^{d+1}(v) \cup \{v_{\texttt{bias}}\}} \sup_{\theta \in \Theta} \left| \sum_{i=1}^{n} \varepsilon_{i,v,m} w(\theta, x_i) \right| \right).$$

I now derive similar inequalities when $\rho = \mathrm{id}$ and $\rho = \mathrm{ReLU}$.

Step 3b: deal with $\rho = \mathrm{id}, \mathrm{ReLU}$ via rescaling. In the case $\rho = \mathrm{ReLU}$, Lemma C.3.1 shows that

$$\begin{aligned}
&\mathbb{E}_\varepsilon g \left(\max_{z \in Z} \sup_{t \in T^z} \sum_{i=1}^{n} \varepsilon_{i,z} \mathrm{ReLU}(t_i) \right) \\
&\leqslant \mathbb{E}_\varepsilon g \left(\max_{z \in Z} \sup_{t \in T^z} \sum_{i=1,\dots,n} \varepsilon_{i,z} t_i \right).
\end{aligned}$$

The difference with the $*$-max-pooling case is that each t_i is scalar so this does not introduce an additional index w to the Rademacher variables. The right-hand side can be rewritten as

$$\mathbb{E}_\varepsilon g \left(\max_{\substack{v \in N_{\mathrm{out}},\\ m=1,\dots,M}} \sup_{u \in (\mathrm{ant}^d(v) \cap N_{\mathrm{ReLU}}) \setminus N_{\mathrm{in}}, \theta \in \Theta} \sum_{i=1,\dots,n} \varepsilon_{i,v,m} \left\langle \begin{pmatrix} \theta^{\to u} \\ b_u \end{pmatrix}, \begin{pmatrix} R_{\mathrm{ant}(u)}(\theta, x_i) \\ 1 \end{pmatrix} \right\rangle \right)$$

I can only increase the latter by considering a maximum over all $u \in \mathrm{ant}^d(v)$, not only the ones in N_{ReLU}. I also add absolutes values. This is then bounded by

$$F := \mathbb{E}_\varepsilon g\left(\max_{\substack{v\in N_{\mathrm{out}},\\ m=1,\dots,M}} \sup_{u\in\mathrm{ant}^d(v)\setminus N_{\mathrm{in}},\theta\in\Theta} \left|\sum_{i=1,\dots,n} \varepsilon_{i,v,m}\left\langle \begin{pmatrix}\theta^{\to u}\\ b_u\end{pmatrix}, \begin{pmatrix}R_{\mathrm{ant}(u)}(\theta,x_i)\\ 1\end{pmatrix}\right\rangle\right|\right). \tag{C.9}$$

This means that $E(\mathrm{ReLU}) \leqslant F$. Let me also observe that $e(\mathrm{id}) \leqslant F$. Indeed, recall that by definition

$$e(\mathrm{id}) = \mathbb{E}_\varepsilon g\left(\max_{\substack{v\in N_{\mathrm{out}},\\ m=1,\dots,M}} \max_{u\in(\mathrm{ant}^d(v)\cap N_{\mathrm{id}})\setminus N_{\mathrm{in}}} \sup_\theta \left|\sum_{i=1}^n \varepsilon_{i,v,m} u(\theta,x_i)\right|\right).$$

I can only increase the latter by considering a maximum over all $u \in \mathrm{ant}^d(v)$. Moreover, for an identity neuron u, it holds $u(\theta,x) = \left\langle \begin{pmatrix}\theta^{\to u}\\ b_u\end{pmatrix}, \begin{pmatrix}R_{\mathrm{ant}(u)}(\theta,x)\\ 1\end{pmatrix}\right\rangle$. This shows that $e(\mathrm{id}) \leqslant F$. It remains to bound F using that the assumption on the norm of the parameters. Introduce a new neuron $v_{\texttt{bias}}$ that computes the constant function equal to one: $v_{\texttt{bias}}(\theta,x) = 1$. Note that

$$\begin{aligned}
&\sum_{i=1,\dots,n} \varepsilon_{i,v,m}\left\langle \begin{pmatrix}\theta^{\to u}\\ b_u\end{pmatrix}, \begin{pmatrix}R_{\mathrm{ant}(u)}(\theta,x_i)\\ 1\end{pmatrix}\right\rangle\\
&= \left\langle \begin{pmatrix}\theta^{\to u}\\ b_u\end{pmatrix}, \sum_{i=1,\dots,n} \varepsilon_{i,v,m}\begin{pmatrix}R_{\mathrm{ant}(u)}(\theta,x_i)\\ 1\end{pmatrix}\right\rangle\\
&\underset{\text{Hölder}}{\leqslant} \underbrace{\left(\|\theta^{\to u}\|_1 + |b_u|\right)}_{\leqslant 1 \text{ by assumption}} \max_{w\in\mathrm{ant}(u)\cup\{v_{\texttt{bias}}\}} \left|\sum_{i=1}^n \varepsilon_{i,v,m} w(\theta,x_i)\right|.
\end{aligned}$$

This shows that

$$F \leqslant \mathbb{E}_\varepsilon g\left(\max_{\substack{v\in N_{\mathrm{out}},\\ m=1,\dots,M}} \sup_{u\in\mathrm{ant}^d(v)\setminus N_{\mathrm{in}},\theta\in\Theta} \max_{w\in\mathrm{ant}(u)\cup\{v_{\texttt{bias}}\}} \left|\sum_{i=1}^n \varepsilon_{i,v,m} w(\theta,x_i)\right|\right).$$

Obviously, introducing additional copies of ε to make the third index going from $m=1$ to KM can only make it larger. Moreover, $u \in \mathrm{ant}^d(v)\setminus N_{\mathrm{in}}$ and $w \in \mathrm{ant}(u)$ implies $w \in \mathrm{ant}^{d+1}(u)$, so we can instead consider a maximum over $w \in \mathrm{ant}^{d+1}(v)$. This gives the upper-bound

$$\begin{aligned}
F &\leqslant \mathbb{E}_\varepsilon g\left(\max_{\substack{v\in N_{\mathrm{out}},\\ m=1,\dots,KM}} \max_{w\in\mathrm{ant}^{d+1}(v)\cup\{v_{\texttt{bias}}\}} \sup_{\theta\in\Theta} \left|\sum_{i=1}^n \varepsilon_{i,v,m} w(\theta,x_i)\right|\right)\\
&= H.
\end{aligned}$$

Step 4: putting everything together. At the end, recalling that there are at most P different $k \in \mathbb{N}_{>0}$ associated with an existing k-max-pooling neuron, we get the final

bound

$$\begin{aligned}
&\mathbb{E}_\varepsilon g\left(\max_{\substack{v\in N_{\text{out}},\\ m=1,\dots,M}} \max_{u\in \text{ant}^d(v)\setminus N_{\text{in}}} \sup_\theta \left|\sum_{i=1}^n \varepsilon_{i,v,m} u(\theta, x_i)\right|\right)\\
&\leqslant e(\text{id}) + e(\text{ReLU}) + \sum_k e(k\text{-}\texttt{pool})\\
&\leqslant e(\text{id}) + 2E(\text{ReLU}) + 2\sum_k E(k\text{-}\texttt{pool})\\
&\leqslant F + 2F + 2\sum_k E(k\text{-}\texttt{pool})\\
&\leqslant H + 2H + 2\sum_k H\\
&\leqslant H + 2H + 2PH = (3+2P)H.
\end{aligned}$$

The term $(3+2P)H$ can again be bounded by splitting the maximum over $w \in \text{ant}^{d+1}(v) \cup \{v_{\texttt{bias}}\}$ between the w's that are input neurons, and those that are not, since everything is non-negative. This yields the claim. □

Remark C.4.1 (Improved dependencies on the kernel size)**.** *Note that in the proof of Lemma C.4.3, the multiplication of M by K can be avoided if there are no $*$-max-pooling neurons in $\text{ant}^d(v)$. Because of skip connections, even if there is a single $*$-max-pooling neuron in the architecture, it can be in $\text{ant}^d(v)$ for many d's. A more advanced version of the argument is to peel only the ReLU and identity neurons, by leaving the $*$-max-pooling neurons as they are, until we reach a set of $*$-max-pooling neurons large enough that we decide to peel simultaneously. This would prevent the multiplication by K every time d is increased.*

I can now state the main peeling theorem, which directly result from Lemma C.4.2 and Lemma C.4.3 by induction on d. Note that these lemmas contain assumptions on the size of the incoming weights of the different neurons. These assumptions are met using Algorithm 3.2.1, that normalizes the parameters without changing the associated function nor the path-norm (Lemma 3.2.1).

Theorem C.4.1. *Consider a neural network architecture as in Definition 2.2.2 with $N_{out} \cap N_{in} = \emptyset$. Assume that $b_v = 0$ for every $v \in N_{*\text{-}\texttt{pool}}$. Define $P := |\{k \in \mathbb{N}_{>0}, \exists u \in N_{k\text{-}\texttt{pool}}\}|$ the number of different types of $*$-max-pooling neurons in G, and $K := \max_{u\in N_{*\text{-}\texttt{pool}}}$ the maximum kernel size ($K := 1$ by convention if $P = 0$). Denote by $v_{\texttt{bias}}$ a new input neuron and define $x_{v_{\texttt{bias}}} = 1$ for any input x. For any set of parameters Θ associated with the network, such that $\|\Phi(\theta)\|_1 \leqslant r$ for every $\theta \in \Theta$, it holds for every convex non-decreasing function $g : \mathbb{R} \to \mathbb{R}_{\geqslant 0}$*

$$\begin{aligned}
&\mathbb{E}_\varepsilon g\left(\sup_{\theta\in\Theta} \sum_{\substack{i=1,\dots,n,\\ v\in N_{out}}} \varepsilon_{i,v} v(\theta, x_i)\right)\\
&\leqslant \frac{(3+2P)^D}{2+2P} \mathbb{E}_\varepsilon g\left(r \max_{\substack{v\in N_{out},\\ m=1,\dots,K^{D-1}}} \max_{u\in N_{in}\cup\{v_{\texttt{bias}}\}} \left|\sum_{i=1}^n \varepsilon_{i,v,m} x_{i,u}\right|\right).
\end{aligned}$$

Proof of Theorem C.4.1. Without loss of generality, we can replace Θ by its image under Algorithm 3.2.1 with $q = 1$, as Algorithm 3.2.1 does not change the associated function R_θ nor the path-norm $\|\Phi(\theta)\|_1$ (Lemma 3.2.1) so that we still have $\|\Phi(\theta)\|_1 \leqslant r$ and the supremum over $\theta \in \Theta$ on the left-hand side can be taken over rescaled parameters. By Lemma 3.2.1, the parameters are 1-normalized (Definition 3.2.1), so we will be able to use Lemma C.4.2 and Lemma C.4.3. Indeed, 1-normalized parameters θ satisfy $\sum_{v\in N_{\text{out}}} \|\theta^{\to v}\|_1 + |b_v| = \|\Phi(\theta)\|_1 \leqslant r$ so Lemma C.4.2 applies. We also have $\|\theta^{\to v}\|_1 + |b_v| \leqslant 1$ for every $v \notin N_{\text{in}} \cup N_{\text{out}}$, by definition of 1-normalization, so Lemma C.4.3 also applies.

By induction on $d \geqslant 1$, I prove that (highlighting in orange what is important)

$$\begin{aligned}
&\mathbb{E}_\varepsilon g\left(\sup_{\theta\in\Theta} \sum_{\substack{i=1,\dots,n,\\ v\in N_{\text{out}}}} \varepsilon_{i,v} v(\theta, x_i)\right) \\
&\leqslant \sum_{\ell=1}^{d} (3+2P)^{\ell-1} \mathbb{E}_\varepsilon g\left(r \max_{\substack{v\in N_{\text{out}},\\ m=1,\dots,K^{\ell-1}}} \max_{u\in(\text{ant}^\ell(v)\cap N_{\text{in}})\cup\{v_{\texttt{bias}}\}} \left|\sum_{i=1}^{n} \varepsilon_{i,v,m} x_{i,u}\right|\right) \\
&+ (3+2P)^{d-1} \mathbb{E}_\varepsilon g\left(r \max_{\substack{v\in N_{\text{out}},\\ m=1,\dots,K^{d-1}}} \max_{u\in\text{ant}^d(v)\setminus N_{\text{in}}} \sup_{\theta\in\Theta} \left|\sum_{i=1}^{n} \varepsilon_{i,v,m} u(\theta, x_i)\right|\right),
\end{aligned}$$

with the same convention as in Lemma C.4.2 for maxima over empty sets. This is true for $d = 1$ by Lemma C.4.2. The induction step is verified using Lemma C.4.3. This concludes the induction. Applying the result for $d = D$, and since $\text{ant}^D(v) \setminus N_{\text{in}} = \emptyset$, we get:

$$\begin{aligned}
&\mathbb{E}_\varepsilon g\left(\sup_{\theta\in\Theta} \sum_{\substack{i=1,\dots,n,\\ v\in N_{\text{out}}}} \varepsilon_{i,v} v(\theta, x_i)\right) \\
&\leqslant \sum_{d=1}^{D} (3+2P)^{d-1} \mathbb{E}_\varepsilon g\left(r \max_{\substack{v\in N_{\text{out}},\\ m=1,\dots,K^{d-1}}} \max_{u\in(\text{ant}^d(v)\cap N_{\text{in}})\cup\{v_{\texttt{bias}}\}} \left|\sum_{i=1}^{n} \varepsilon_{i,v,m} x_{i,u}\right|\right).
\end{aligned}$$

We can only increase the right-hand side by considering maximum over all $u \in N_{\text{in}} \cup \{v_{\texttt{bias}}\}$ and by adding independent copies indexed from $m = 1$ to $m = K^{D-1}$. Moreover, $\sum_{d=1}^{D} (3+2P)^{d-1} = ((3+2P)^D - 1)/(2+2P)$. This shows the final bound:

$$\begin{aligned}
&\mathbb{E}_\varepsilon g\left(\sup_{\theta\in\Theta} \sum_{\substack{i=1,\dots,n,\\ v\in N_{\text{out}}}} \varepsilon_{i,v} v(\theta, x_i)\right) \\
&\leqslant \frac{(3+2P)^D}{2+2P} \mathbb{E}_\varepsilon g\left(r \max_{\substack{v\in N_{\text{out}},\\ m=1,\dots,K^{D-1}}} \max_{u\in N_{\text{in}}\cup\{v_{\texttt{bias}}\}} \left|\sum_{i=1}^{n} \varepsilon_{i,v,m} x_{i,u}\right|\right).
\end{aligned}$$

□

C.5 Proof of Theorem 4.4.1

Proof of Theorem 4.4.1. The proof is given directly in the general case with nonzero biases on every $v \notin N_{*\texttt{-pool}}$ and uses an additional input neuron $v_{\texttt{bias}}$. I highlight along the proof where improved results can be obtained assuming zero biases, yielding Theorem 4.4.1 as a consequence. Define the random matrices $E = (\varepsilon_{i,v})_{i,v} \in \mathbb{R}^{n \times d_{\text{out}}}$ and $R(\theta, S) = (v(\theta, x_i))_{i,v} \in \mathbb{R}^{n \times d_{\text{out}}}$ so that $\langle E, R(\theta, S)\rangle = \sum_{i,v} \varepsilon_{i,v}(R_\theta(x_i))_v$. By definition of the Rademacher complexity (Equation (4.9)), we have

$$\mathcal{R}(\mathcal{F}_\Theta, \mu_x) = \mathbb{E}_{S,\varepsilon}\left(\sup_\theta \langle E, R(\theta, S)\rangle\right)$$

where $S = (x_i)_{i=1}^n \sim (\mu_x)^{\otimes n}$.

I condition on S and denote $\mathbb{E}_\varepsilon$ the conditional expectation. For any random variable $\lambda(S) > 0$ measurable in S, it holds

$$\begin{aligned}
\mathbb{E}_\varepsilon\left(\sup_\theta \langle E, R(\theta, S)\rangle\right) &= \frac{1}{\lambda(S)} \log\exp\left(\lambda(S)\mathbb{E}_\varepsilon\left(\sup_\theta \langle E, R(\theta, S)\rangle\right)\right) \\
&\underset{\lambda \text{ measurable in } S}{=} \frac{1}{\lambda(S)} \log\exp\left(\mathbb{E}_\varepsilon\left(\lambda(S)\sup_\theta \langle E, R(\theta, S)\rangle\right)\right) \\
&\underset{\text{Jensen}}{\leqslant} \frac{1}{\lambda(S)} \log \mathbb{E}_\varepsilon \exp\left(\lambda(S)\sup_\theta \langle E, R(\theta, S)\rangle\right).
\end{aligned}$$

For a deterministic $s = (x_i)_{i=1}^n \in (\mathbb{R}^{d_{\text{in}}})^n$, denote

$$e(s) = \mathbb{E}_\varepsilon \exp\left(\lambda(s)\sup_\theta \langle E, R(\theta, x)\rangle\right).$$

Since S is independent of ε, the expectation conditioned to S is simply equal to

$$\mathbb{E}_\varepsilon \exp\left(\lambda(S)\sup_\theta \langle E, R(\theta, S)\rangle\right) = e(S).$$

Denote $r = \sup_{\theta\in\Theta} \|\Phi(\theta)\|_1$. For s as above, simply denote $\lambda := \lambda(s)$. Since $N_{\text{in}} \cap N_{\text{out}} = \emptyset$ and the biases of $*$-max-pooling neurons are null, the peeling argument given by Theorem C.4.1 for $g : t \in \mathbb{R} \mapsto \exp(\lambda t)$ guarantees:

$$e(s) \leqslant \frac{(3+2P)^D}{2+2P} \mathbb{E}_\varepsilon \exp\left(\lambda r \max_{\substack{v\in N_{\text{out}},\\ m=1,\dots,K^{D-1}}} \max_{u\in N_{\text{in}}\cup\{v_{\texttt{bias}}\}} \left|\sum_{i=1}^n \varepsilon_{i,v,m} x_{i,u}\right|\right),$$

where $x_{i,u}$ is coordinate u of vector $x_i \in \mathbb{R}^{d_{\text{in}}}$, and where $v_{\texttt{bias}}$ is an added neuron for which I set by convention $x_{v_{\texttt{bias}}} = 1$ for any input x. It is easy to check that the same bound holds true with a maximum only over $u \in N_{\text{in}}$ (not considering $v_{\texttt{bias}}$ in the maximum) when all biases are constrained to be null. In such a setting, all the $\max_{u\in N_{\text{in}}\cup\{v_{\texttt{bias}}\}}$ below can be replaced by $\max_{u\in N_{\text{in}}}$. Denote

$$\sigma(s) := \max_{u\in N_{\text{in}}\cup\{v_{\texttt{bias}}\}} \left(\sum_{i=1}^n x_{i,u}^2\right)^{1/2} \geqslant \sqrt{n}.$$

Using Lemma C.5.1, it holds

$$\mathbb{E}_\varepsilon \exp\left(\lambda r \max_{\substack{v\in N_{\text{out}},\\ u\in N_{\text{in}}\cup\{v_{\texttt{bias}}\},\\ m=1,\dots,K^{D-1}}} \left|\sum_{i=1}^n \varepsilon_{i,v,m}(x_i)_u\right|\right) \leqslant 2K^{D-1}(d_{\text{in}}+1)d_{\text{out}} \exp\left(\frac{(r\lambda(s)\sigma(s))^2}{2}\right).$$

When biases are constrained to be null, $d_{\text{in}}+1$ is replaced by d_{in}. Putting everything together, we get:

$$\mathbb{E}_\varepsilon\left(\sup_\theta \langle E, R(\theta,S)\rangle\right) = e(S) \leqslant \left(\frac{1}{\lambda}\log(C_1) + \lambda(S)C_2(S)\right)$$

with

$$C_1 = 2K^{D-1}(d_{\text{in}}+1)d_{\text{out}} \times \frac{(3+2P)^D}{2+2P} = \frac{3+2P}{1+P}((3+2P)K)^{D-1}(d_{\text{in}}+1)d_{\text{out}}$$

(again with $d_{\text{in}}+1$ replaced by d_{in} when all biases are null) and

$$C_2(S) = \frac{1}{2}(r\sigma(S))^2.$$

Choosing $\lambda(S) = \sqrt{\frac{\log(C_1)}{C_2(S)}}$ yields:

$$\begin{aligned}\mathbb{E}_\varepsilon\left(\sup_\theta \langle E, R(\theta,S)\rangle\right) &\leqslant 2\sqrt{\log(C_1)C_2(S)}\\ &\leqslant \underbrace{\sqrt{2}\sigma(S)r}_{=2\sqrt{C_2(S)}} \underbrace{\left(\log\left(\frac{3+2P}{1+P}(d_{\text{in}}+1)d_{\text{out}}\right) + D\log\left((3+2P)K\right)\right)^{1/2}}_{\geqslant\sqrt{\log(C_1)}}\end{aligned}$$

with $d_{\text{in}}+1$ replaced by d_{in} when all biases are null. Taking the expectation on both sides over S yields Theorem 4.4.1. □

The next lemma is classical [Golowich et al., 2018, Section 7.1] and is here only for completeness.

Lemma C.5.1. *For any $d, k \in \mathbb{N}_{>0}$ and $\lambda > 0$, it holds*

$$\mathbb{E}_\varepsilon \exp\left(\lambda \max_{\substack{m=1,\dots,k,\\ u=1,\dots,d}} \left|\sum_{i=1}^n \varepsilon_{i,m}(x_i)_u\right|\right) \leqslant 2kd \max_{u=1,\dots,d} \exp\left(\frac{\lambda^2}{2}\sum_{i=1}^n (x_i)_u^2\right).$$

Proof. It holds

$$\mathbb{E}_\varepsilon \exp\left(\lambda \max_{\substack{m=1,\dots,k,\\ u=1,\dots,d}} \left|\sum_{i=1}^n \varepsilon_{i,m}(x_i)_u\right|\right) \leqslant \sum_{\substack{m=1,\dots,k,\\ u=1,\dots,d}} \mathbb{E}_\varepsilon \exp\left(\lambda\left|\sum_{i=1}^n \varepsilon_{i,m}(x_i)_u\right|\right).$$

For given u and m:

$$\mathbb{E}_\varepsilon \exp\left(\lambda \left|\sum_{i=1}^n \varepsilon_{i,m}(x_i)_u\right|\right) \underset{\text{Lemma C.4.1}}{\leqslant} 2\mathbb{E}_\varepsilon \exp\left(\lambda \sum_{i=1}^n \varepsilon_{i,m}(x_i)_u\right)$$
$$= 2\prod_{i=1}^n \frac{\exp\left(\lambda(x_i)_u\right) + \exp\left(-\lambda(x_i)_u\right)}{2} \leqslant 2\exp\left(\frac{\lambda^2}{2}\sum_{i=1}^n (x_i)_u^2\right)$$

using $\exp(x) + \exp(-x) \leqslant 2\exp(x^2/2)$ in the last inequality. □

C.6 The cross-entropy loss is Lipschitz continuous

Theorem 4.1.1 *applies to the cross-entropy loss with* $L = \sqrt{2}$. To see this, first recall that with C classes, the cross-entropy loss is defined as

$$\ell : (x, y) \in \mathbb{R}^C \times \{0,1\}^C \mapsto -\sum_{c=1}^{d_{\text{out}}} y_c \log\left(\frac{\exp(x_c)}{\sum_d \exp(x_d)}\right).$$

Consider $y \in \{0,1\}^C$ with exactly one nonzero coordinate and an exponent $p \in [1,\infty]$ with conjugate exponent p' $(1/p + 1/p' = 1)$. For every $x, x' \in \mathbb{R}^C$:

$$\ell(x,y) - \ell(x',y) \leqslant 2^{1/p'}\|x - x'\|_p.$$

Consider a class $c \in \{1,\dots,C\}$ and take $y \in \{0,1\}^C$ to be a one-hot encoding of c (meaning that $y_{c'} = \mathbb{1}_{c'=c}$). Consider an exponent $p \in [1,\infty]$ with conjugate exponent p' $(1/p + 1/p' = 1)$. The function $f : x \mapsto \ell(x,y) = -\sum_c y_c \log\left(\frac{\exp(x_c)}{\sum_{c'=1}^C \exp(x_{c'})}\right) = -\log\left(\frac{\exp(x_c)}{\sum_{c'=1}^C \exp(x_{c'})}\right)$ is continuously differentiable so that for every $x, x' \in \mathbb{R}^C$:

$$f(x) - f(x') = \int_0^1 \langle \nabla f(tx + (1-t)x'), x - x'\rangle\, dt \leqslant \sup_{t\in[0,1]} \|\nabla f(tx + (1-t)x')\|_p \|x - x'\|_{p'}.$$

In order to differentiate f, let's start to differentiate $g(x) = \frac{\exp(x_c)}{\sum_{c'=1}^C \exp(x_{c'})}$. Denote ∂_i the partial derivative with respect to coordinate i. For $i \neq c$:

$$\partial_c g(x) = \frac{\exp(x_c)\left(\sum_{c'} \exp(x_{c'})\right) - \exp(x_c)\left(\exp(x_c)\right)}{\left(\sum_{c'} \exp(x_{c'})\right)^2}$$
$$= g(x)\frac{\sum_{c'\neq c} \exp(x_{c'})}{\sum_{c'} \exp(x_{c'})}.$$
$$\partial_i g(x) = \frac{0\left(\sum_{c'} \exp(x_{c'})\right) - \exp(x_c)\left(\exp(x_i)\right)}{\left(\sum_{c'} \exp(x_{c'})\right)^2}$$
$$= g(x)\frac{-\exp(x_i)}{\sum_{c'} \exp(x_{c'})}.$$

Since $f(x) = (-\log \circ h)(x)$:

$$\partial_i f(x) = -\frac{\partial_i g(x)}{g(x)}$$
$$= \frac{1}{\sum_{c'=1}^C \exp(x_{c'})} \times \begin{cases} -\sum_{c'\neq c} e^{x_{c'}} & \text{if } i = c, \\ e^{x_i} & \text{otherwise.} \end{cases}$$

Thus

$$\begin{aligned}
\|\nabla f(x)\|_p^p &= \sum_{i=1}^{C} |\partial_i f(x)|^p \\
&= \frac{\left(\sum_{c' \neq c} \exp(x_{c'})\right)^p + \sum_{c' \neq c} \exp(x_{c'})^p}{\left(\sum_{c'=1}^{C} \exp(x_{c'})\right)^p} \\
&\leqslant 2 \frac{\left(\sum_{c' \neq c} \exp(x_{c'})\right)^p}{\left(\sum_{c'=1}^{C} \exp(x_{c'})\right)^p} \\
&\leqslant 2 \frac{\left(\sum_{c' \neq c} \exp(x_{c'})\right)^p}{\left(\sum_{c' \neq c} \exp(x_{c'})\right)^p} \\
&= 2.
\end{aligned}$$

where I used in the first inequality that $\|v\|_p^p \leqslant \|v\|_1^p$ for any vector v. This shows that for every $x, x' \in \mathbb{R}^C$:

$$\ell(x, y) - \ell(x', y) \leqslant 2^{1/p} \|x - x'\|_{p'}.$$

Appendix D

Supplemental material for Chapter 5

D.1 Related works

We now review the numerical results we found in the literature about time efficiency of existing algorithms for Kronecker-sparse matrix multiplication.

It is reported in Dao et al. [2022a] that replacing dense matrices by a product of two Kronecker-sparse matrices led to a twice faster training for image classification and language modeling.

In Fu et al. [2023] is reported an acceleration of $\mathbf{X} \mapsto \mathbf{W}^{-1}(\mathbf{U} \odot \mathbf{W}\mathbf{X})$ where $\mathbf{U}$ is some dense weight matrix, $\odot$ is the element-wise multiplication, and $\mathbf{W}$ is the DFT matrix (which admits a factorization in Kronecker-sparse matrices), as soon as the dimensions of $\mathbf{W}$ are at least equal to 4096.

Our study is complementary to these observations: we extensively benchmark the efficiency of the Kronecker-sparse matrix multiplication alone.

D.2 Experiments

This section provides additional details on the experiments presented in Chapter 5 and complementary results.

D.2.1 Details on the experiments

The pytorch package version is 2.2 and pytorch-cuda is 12.1.

Matrix sizes. In all our experiments with matrices, we set the batch size to $B = 128 \times 196 = 25088$, a very standard choice for ViTs, as this quantity corresponds to the standard number of tokens per sequence (192) multiplied by the standard number of sequences in a batch of inputs (128). When dealing with a batch of images in neural networks, we choose the standard choice of batch size $B = 128$.

Matrix entries. The coordinates of any Kronecker-sparse matrix $\mathbf{K} \in \mathbb{R}^{abd \times acd}$ with sparsity pattern (a, b, c, d) are drawn i.i.d. uniformly in $[-\frac{1}{\sqrt{c}}, \frac{1}{\sqrt{c}}]$, corresponding to the

initialization used for training in Dao et al. [2022a]. The coordinates of the inputs $\mathbf{X}$ are drawn i.i.d. according to a standard normal distribution $\mathcal{N}(0, 1)$.

Benchmarking time execution. All the experiments measuring time execution of a Kronecker-sparse matrix multiplication algorithm (Tables 5.3 to 5.5 and D.2, Figures 5.5 to 5.8, 5.10 and D.2 to D.7) are performed on a NVIDIA A100-PCIE-40GB GPU associated with an Intel(R) Xeon(R) Silver 4215R CPU @ 3.20GHz with 377G of memory. The full benchmark took approximately 3 days in an isolated environment, ensuring that no other processes were running concurrently.

Measurements are done using the PyTorch tool `torch.utils.benchmark.Timer`. The medians are computed on at least 10 measurements of 10 runs. In 94.2% of the cases, we have an interquartile range (IQR) that is at least 100 times smaller than the median (resp. 98% for 50 times smaller, and 99.7% for 10 times smaller).

Benchmarking energy consumption. Measurements of the energy consumption (Figure 5.9) is done on a NVIDIA Tesla V100-PCIE-16GB GPU associated with an Intel(R) Xeon(R) Silver 4215R CPU @ 3.20GHz with 754G of memory. The full benchmark took approximately 1.5 days in an isolated environment. Measurements are made using the pyJoules software toolkit. The medians are computed on at 10 measurements of at least 16 runs. In 96% of the cases, the IQR is at least 10 times smaller than the median, and 5 times smaller in all the cases.

Kronecker sparsity patterns benchmarked for time measurements (Section 5.4). The considered patterns are generated by the Python code written in Figure D.1. In all the cases, we only consider patterns (a, b, c, d) with $b = c$ or $b = 4c$ or $c = 4d$ to have an input size d_{in} and an output size d_{out} such that $d_{\text{in}} = d_{\text{out}}$ or $d_{\text{in}} = 4d_{\text{out}}$ or $d_{\text{out}} = 4d_{\text{in}}$. This choice is motivated by the fact that fully-connected layers in ViTs satisfy have input and output sizes satisfying these constraints.

The first "for" loop in Figure D.1 generates a wide range of patterns (a, b, c, d) with $a = 1$, as this represents the simplest scenario. Indeed, the case $a > 1$ simply corresponds to repeating a times the case $a = 1$ in parallel.

The second "for" loop in Figure D.1 generates patterns with $a > 1$ offering fewer choices for d to keep the benchmark concise in terms of execution time. This loop also imposes additional conditions on b and c (line 28 of the code) that we now explain. Many graphs are plotted based on the ratio $(b+c)/bc$, as introduced in Equation (5.2). Because of that, our goal was to include as many distinct ratios $(b + c)/bc$ as possible while keeping the benchmark brief. We excluded certain (b, c) values because they resulted in a ratio that was very close to one already in the benchmark and were more computationally intensive.

Patterns benchmarked for energy measurements (Section 5.4). For the energy measurements, the goal is to have diverse sparsity patterns (a, b, c, d) corresponding to many different ratios $d(b + c)/bc$ to observe the trend in Figure 5.9, while keeping the benchmark as short as possible. We chose to consider the cartesian product of

```
1 a_list = [1, 4, 16, 32, 64]
2 b_list = [48, 64, 96, 128, 192, 256, 384, 512, 768, 1024]
```

```
c_list = [48, 64, 96, 128, 192, 256, 384, 512, 768, 1024]
d_list = [1, 2, 3, 4, 6, 8, 12, 16, 24, 32, 48, 64]
```

by skipping as in Figure D.1 all the patterns with

```
(b,c) in [(1024 , 256) , (256 , 1024) , (128 , 512) , (512 , 128)
    , (64 , 256) , (256 , 64)]
```

and also all the patterns such that

```
 b != c and b != 4 * c and c != 4 * b
```

for the same reasons as explained above for time measurements.

Details on boxplots. In all boxplots (Figures 5.5 to 5.10 and D.2 to D.7), the orange line corresponds to the median, the boxes to the first and third quartile and the whiskers to the 5th and the 95th percentile. Outliers are not represented on the graph.

D.2.2 Estimating the time for memory rewritings in the `bmm` implementation (Section 5.2)

Protocol. Given a Kronecker sparsity pattern $\pi = (a, b, c, d)$, an associated π-Kronecker-sparse matrix $\mathbf{K}$ (Definition 5.1.1) and an input $\mathbf{X} \in \mathbb{R}^{B \times acd}$ for some batch size B, we first measure the time Δt to compute $\mathbf{Y} := \mathbf{X}\mathbf{K}^\top$ using the `bmm` implementation. Then, we measure the time $\Delta\tilde{t}$ to perform only the multiplication operations $\mathbf{Y}[:, \texttt{row}] = \mathbf{X}[:, \texttt{col}]\mathbf{K}^\top[\texttt{col}, \texttt{row}]$ in the `bmm` implementation (line 2 of Algorithm 5.1.2). Therefore, the estimated relative time to perform the memory rewritings of lines 1 and 3 of Algorithm 5.1.2 is simply $\frac{\Delta t - \Delta\tilde{t}}{\Delta t}$.

Results. Figure 5.5, which is replicated in the left part of Figure D.2, shows that the relative time spent doing memory rewritings in `bmm` increases with the ratio $(b + c)/(bc)$, in the *batch-size-first* memory layout. Figure D.2 shows that this is similar for both *batch-size-first* and *batch-size-last*.

D.2.3 Time spent in linear layers in vision transformers

This section gives a numerical lower bound estimate on the time spent in fully-connected layers in a Vision Transformer (ViT).

Results. Table D.1 shows that, for different ViTs, the fraction of computation time solely dedicated to linear layers in feed-forward network modules varies between 31% and 53% in *half-precision*, and 46% and 61% in *float-precision*. This proportion increases with the size of the architecture. This shows that a non-negligible amount of ViTs inference is dedicated to fully-connected layers. Note that the time for the fully-connected linear layers in the multi-head attention module is not included in our measurements, so our estimate is *only a lower bound* on the time effectively devoted to all fully-connected layers in transformer architectures.

Table D.1: Median execution times (ms) of the forward-pass in a ViT, and the forward-pass in an MLP containing only all the linear layers involved in the feed-forward network modules of the ViT. The latter is reported with its ratio over the first. FP16 is *half-precision*, FP32 is *float-precision*.

Architecture	fp16 (s)		fp32 (s)	
	Complete	Linear in FFNs	Complete	Linear in FFNs
ViT-S/16	0.014	0.0046 (31%)	0.090	0.04 (46%)
ViT-B/16	0.036	0.015 (42%)	0.30	0.16 (54%)
ViT-L/16	0.11	0.050 (46%)	1.0	0.58 (58%)
ViT-H/14	0.31	0.16 (53%)	2.6	1.6 (61%)

Details on the estimation. The transformer architecture is composed of a sequence of transformer blocks, where each block contains a multi-head attention module and a feed-forward network module. The feed-forward network module is an MLP with one hidden layer of neurons, involving two fully-connected linear layers. Table D.1 reports the time to perform sequentially all the fully-connected linear layers (without biases) appearing in feed-forward network modules of the considered ViT. This is compared to the total forward time of the transformer network. This is expected to yield a lower bound since we did not measure the time spent in fully-connected linear layers in the multi-head attention module.

Experimental settings. The architecture ViT-S/16 corresponds to the one in Zhai et al. [2022], while the architecture ViT-B/16, ViT-L/16 and ViT-H/14 correspond to those in Dosovitskiy et al. [2021]. Input images are of size 224×224. In *float-precision*, the PyTorch implementation of ViT architecture are taken from Wang [2024b]. In *half-precision*, the considered implementation of the transformer architecture uses FlashAttention [Dao et al., 2022b] to compute the scaled dot product attention, like in Wang [2024a]. The MLP containing only the linear layers of the feed-forward modules in the transformer architecture is implemented using `torch.nn.Sequential` and `torch.nn.Linear`. Experiments are done on a single A100-40GB GPU on AMD EPYC 7742 64-Core Processor. Measurements are done using the PyTorch tool `torch.utils.benchmark.Timer` for benchmarking. The image batch size is set at 128.

D.2.4 Additional results in half-precision

For the sake of completeness we perform the benchmark described in Section 5.4 in *half-precision*. The equivalent of Table 5.3, Figure 5.8, Figures 5.6, 5.7, 5.10 and D.2 in *half-precision* are Table D.2, Figure D.3, Figures D.4 to D.7, respectively. Note that just as Figure 5.8, the Figure D.3 only considers sparsity patterns for which min time(`kernel`, `bmm`, `bsr`, `einsum`) $<$ min time(`dense`, `sparse`). This corresponds to 87% of the tested patterns in *half-precision*, cf. Table D.2.

Table D.2: Percentage out of 600 patterns (a, b, c, d) where `algo1` is *faster* than the `algo2` in *half-precision* (denoted by time(`algo1`) < time(`algo2`)), and the median acceleration factor in such cases (that is, the median ratio $\frac{\text{time of } \texttt{algo2}}{\text{time of } \texttt{algo1}}$). For each implementation, we take the minimum time between the *batch-size-first* and the *batch-size-last* memory layout. Experiments are carried in *half-precision*.

min time $\begin{pmatrix}\texttt{kernel}\\ \texttt{bmm}\\ \texttt{einsum}\\ \texttt{bsr}\end{pmatrix}$ < min time $\begin{pmatrix}\texttt{dense}\\ \texttt{sparse}\end{pmatrix}$	time(`bmm`) < min time $\begin{pmatrix}\texttt{einsum}\\ \texttt{bsr}\\ \texttt{dense}\\ \texttt{sparse}\end{pmatrix}$	time(`kernel`) < min time $\begin{pmatrix}\texttt{bmm}\\ \texttt{einsum}\\ \texttt{bsr}\\ \texttt{dense}\\ \texttt{sparse}\end{pmatrix}$
86.95% (×8.45)	83.22% (×1.83)	36.69% (×1.46)

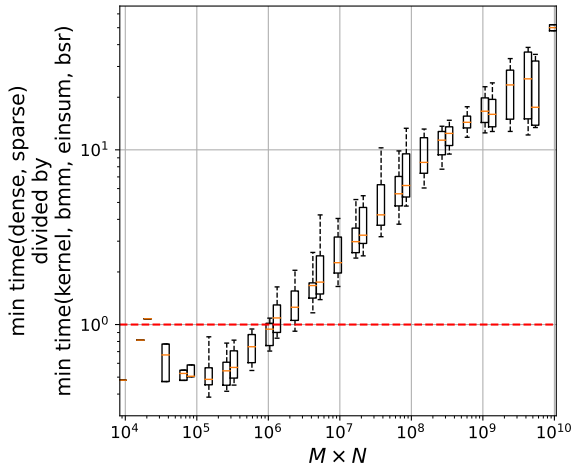


Figure D.5: Speed-up factor of min time(`kernel`, `bmm`, `bsr`, `einsum`) compared to min time(`dense`, `sparse`) vs. the matrix size $d_{\text{out}} \times d_{\text{in}}$. Experiments are carried in *half-precision*.

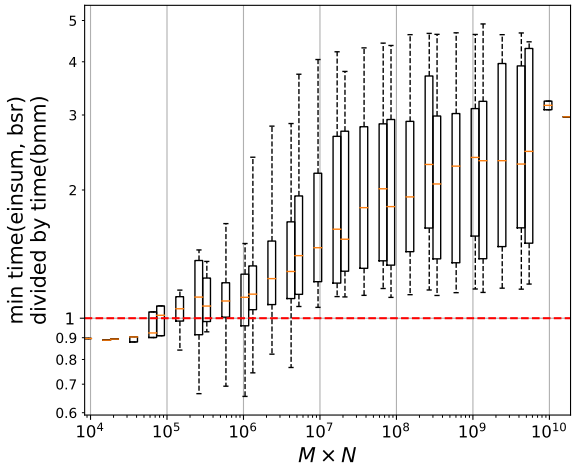


Figure D.6: Speed-up factor of time(`bmm`) compared to min time(`einsum`, `bsr`) vs. the matrix size $d_{\text{out}} \times d_{\text{in}}$. Experiments are carried in *half-precision*.

D.3 Details on perfect shuffle permutations

The goal is to prove Equation (5.1), which we recall here for convenience:

$$\mathbf{S}_\pi = \underbrace{(\mathbf{I}_a \otimes \mathbf{P}_{b,d})}_{:=\mathbf{P}} \underbrace{(\mathbf{I}_{ad} \otimes \mathbf{1}_{b\times c})}_{=\mathbf{S}_{\tilde{\pi}}} \underbrace{(\mathbf{I}_a \otimes \mathbf{P}_{c,d})^\top}_{:=\mathbf{Q}} = \mathbf{P}\mathbf{S}_{\tilde{\pi}}\mathbf{Q},$$

where the matrix $\mathbf{P}_{p,q}$ is the so-called (p, q) perfect shuffle permutation introduced below. To prove this formula, we will use the next lemma.

Lemma D.3.1. *For any positive integers b, c, d:*

$$\mathbf{P}_{b,d}{}^\top(\mathbf{1}_{b\times c} \otimes \mathbf{I}_d)\mathbf{P}_{c,d} = \mathbf{I}_d \otimes \mathbf{1}_{b\times c},$$

where $\mathbf{P}_{p,q}$ denotes the (p, q) perfect shuffle *of $r := pq$ [Van Loan, 2000], which is the permutation matrix of size $r \times r$ defined as:*

$$\mathbf{P}_{p,q} := \begin{pmatrix} \mathbf{I}_r[R_0, :] \\ \mathbf{I}_r[R_1, :] \\ \vdots \\ \mathbf{I}_r[R_{q-1}, :] \end{pmatrix}, \tag{D.1}$$

where $R_i := \{i + qj \,|\, j \in [\![0, p-1]\!]\}$ for $i \in [\![0, q-1]\!]$.

Proof of Lemma D.3.1. This is a direct consequence of a more general result claiming that the Kronecker product commutes up to some perfect shuffle permutation matrices [Van Loan, 2000, Section 1]. □

We now turn to the proof of Equation (5.1).

Proof of Equation (5.1). By definition, $\mathbf{S}_\pi = \mathbf{I}_a \otimes \mathbf{1}_{b\times c} \otimes \mathbf{I}_d$ when $\pi = (a, b, c, d)$. By Lemma D.3.1,

$$\mathbf{S}_\pi = \mathbf{I}_a \otimes \mathbf{1}_{b\times c} \otimes \mathbf{I}_d = \mathbf{I}_a \otimes \left(\mathbf{P}_{b,d}(\mathbf{I}_d \otimes \mathbf{1}_{b\times c})\mathbf{P}_{c,d}{}^\top\right).$$

By the equality $(\mathbf{AB})\otimes(\mathbf{CD}) = (\mathbf{A}\otimes\mathbf{C})(\mathbf{B}\otimes\mathbf{D})$ for any matrices $\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}$ of compatible sizes, we get the result:

$$\begin{aligned}
\mathbf{S}_\pi &= \mathbf{I}_a \otimes \left(\mathbf{P}_{b,d}(\mathbf{I}_d \otimes \mathbf{1}_{b\times c})\mathbf{P}_{c,d}{}^\top\right) \\
&= (\mathbf{I}_a \otimes \mathbf{P}_{b,d}) \left(\mathbf{I}_a \otimes \left((\mathbf{I}_d \otimes \mathbf{1}_{b\times c})\mathbf{P}_{c,d}{}^\top\right)\right) \\
&= (\mathbf{I}_a \otimes \mathbf{P}_{b,d})(\mathbf{I}_a \otimes \mathbf{I}_d \otimes \mathbf{1}_{b\times c})(\mathbf{I}_a \otimes \mathbf{P}_{c,d}{}^\top) \\
&= (\mathbf{I}_a \otimes \mathbf{P}_{b,d})(\mathbf{I}_{ad} \otimes \mathbf{1}_{b\times c})(\mathbf{I}_a \otimes \mathbf{P}_{c,d}{}^\top).
\end{aligned}$$

□

D.4 Implementations

This section gives details on the baseline GPU implementations presented in Section 5.1.2 (Appendix D.4.1) and on the new kernel presented in Section 5.3 (Appendix D.4.2).

D.4.1 Details on baseline GPU implementations

To keep it short, we only give the code in the case of the *batch-size-first* memory layout (except for `dense` and `sparse` where the codes are small). The case of *batch-size-last* can simply be obtained by inverting the first and last positions in all tensor reshapings.

`einsum` implementation. This implementation uses tensor contractions with the high-performance `einops` library. The $abcd$ nonzero entries of the Kronecker-sparse matrix $\mathbf{K}$ (Figure 5.2) are stored in a `PyTorch` 4D-tensor `K_einsum` of shape (a, b, c, d). The implementation uses Einstein notations.

```
def kronecker_einsum(X_bsf, K_einsum):
    X_perm = einops.rearrange(X_bsf, "... (a c d) -> ... a c d",
       a=a, c=c, d=d)
    Y_perm = einops.einsum(X_perm, K_einsum, "... a c d, a b c d
       -> ... a b d")
    Y_bsf = einops.rearrange(Y_perm, "... a b d-> ... (a b d)")
    return Y_bsf
```

The second line of this code does at the same time all the matrix multiplications $\mathbf{Y}[:, \texttt{row}] \leftarrow \mathbf{X}[:, \texttt{col}]\mathbf{K}^\top[\texttt{col}, \texttt{row}]$ for all the pairs $(\texttt{row}, \texttt{col})$ in Algorithm 5.1.1.

`bsr` implementation. This is an implementation of Algorithm 5.1.2 using the high-performance Block compressed Sparse Row (BSR) `PyTorch` library. The matrix $\tilde{\mathbf{K}}$ is stored as a tensor `K_bsr` stored in the BSR format.

```
def kronecker_bsr(X_bsf, K_bsr):
    batch_size = X_bsf.shape[0]
    X_perm = (
        X_bsf.view(batch_size, a, c, d)
        .transpose(-1, -2)
        .reshape(batch_size, a * c * d)
    )
    Y_perm = torch.nn.functional.linear(
        X_perm, K_bsr
    )
    Y_bsf = (
        Y_perm.view(batch_size, a, d, b)
        .transpose(-1, -2)
        .reshape(batch_size, a * b * d)
    )
    return Y_bsf
```

`bmm` implementation. This is an implementation of Algorithm 5.1.2 using the high-performance Block compressed Sparse Row (BSR) `PyTorch` library. The matrix $\tilde{\mathbf{K}}$ is stored as a tensor `K_bsr` stored in the BSR format. This implementation using `torch.bmm`, which is based on high-performance batched matrix multiplication NVIDIA routines. The non-zero entries of $\tilde{\mathbf{K}}$ are stored in a four-dimensional `PyTorch` tensor `K_bmm` of shape $(a * d, b, c)$.

```
def kronecker_bmm(X_bsf, K_bmm):
    batch_size = X_bsf.shape[0]
    X_perm = (
        X_bsf.view(batch_size, a, c, d)
        .transpose(-1, -2)
        .reshape(batch_size, a * d, c).
        contiguous().
        transpose(0, 1)
    )
    Y_perm = torch.empty(batch_size, a * d, b, device=x.device,
       dtype=x.dtype).transpose(0, 1)
    Y_perm = torch.bmm(X_perm, K_bmm.transpose(-1, -2))
    Y_bsf = (
        Y_perm.transpose(0, 1)
        .reshape(batch_size, a, d, b)
        .transpose(-1, -2)
        .reshape(batch_size, a * b * d)
    )
    return
```

`dense` implementation. This ignores the sparsity of the Kronecker-sparse matrix $\mathbf{K}$, that is stored as a dense matrix in a 2d-tensor `K_dense`.

batch-size-first: `torch.nn.functional.linear(X_bsf,K_dense)`

batch-size-last: `torch.matmul(K_dense,X_bsl)`

The implementation in *batch-size-first* is the default PyTorch implementation of a forward-pass of a linear layer. For *batch-size-last*, we had to choose an implementation since Pytorch uses *batch-size-first* by default. We made our choice based on a small benchmark of different alternatives.

`sparse` implementation. This exploits the sparsity of the Kronecker-sparse matrix $\mathbf{K}$ but not its structure (recall that the support are not arbitrary, they are structured since they must be expressed as Kronecker products, see Definition 5.1.1).

batch-size-first: `torch.nn.functional.linear(X_bsf,K_csr)`

batch-size-last: `torch.matmul(K_csr,X_bsl)`

D.4.2 Details on the kernel implementation

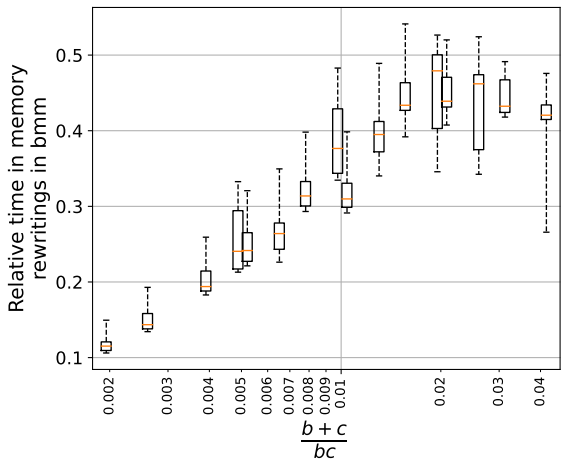
Classical optimizations that we build upon. The proposed implementation use *vectorization* as soon as an operation can be vectorized. Concretely, the float4 and half2 vector types are used to mutualize read/write operations [NVIDIA, 2023b,a, 2024, Boehm,

2022]. An *epilogue* [NVIDIA, 2023a] is also implemented to avoid writing in global memory in a disorganized way. Indeed, after having accumulated the output in registers, each thread has specific rows and columns of the output to write to global memory, and may finish its computation before the others. To avoid that, the epilogue starts to write in the shared memory, in a disorganized way, and then organize the writing from shared to global memory. Another implemented optimization is *double buffering* [NVIDIA, 2023b,a, Boehm, 2022, Li et al., 2019]: a thread block is always both computing the output of a tile, and loading the next tile from global to shared memory. This allows us to hide some latency that arises when loading from the global memory.

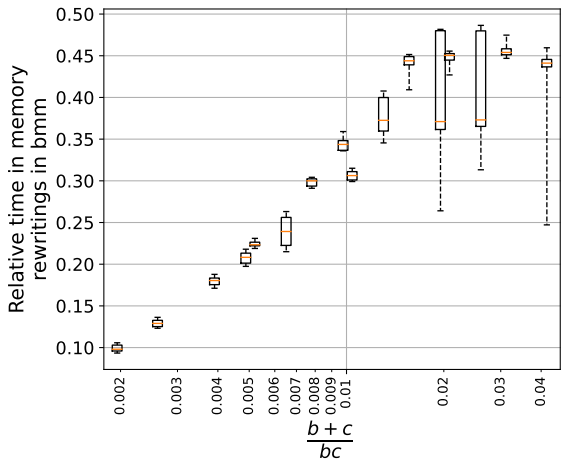
Note that as with any CUDA kernel, the constants (such as the number of threads) need to be tailored to each specific case of use —here, each Kronecker sparsity pattern $\pi = (a, b, c, d)$— and to each GPU.

```python
import itertools

batch_size = 25_088
size_limit = 2_147_483_647

a_list = [1, 2, 3, 4, 6, 8, 12, 16, 24, 32, 48, 64, 96, 128]
b_list = [48, 64, 96, 128, 192, 256, 384, 512, 768, 1024]
c_list = [48, 64, 96, 128, 192, 256, 384, 512, 768, 1024]
d_list1 = [1, 2, 3, 4, 6, 8, 12, 16, 24, 32, 48, 64, 96, 128]
d_list2 = [4, 16, 64]

def get_patterns_benchmark():
    patterns_list = []

    def add_pattern(a, b, c, d):
        if batch_size * a * c * d <= size_limit and \
           batch_size * a * b * d <= size_limit and \
           a * b * c * d <= size_limit:
            patterns_list.append((a, b, c, d))

    for b, c, d in itertools.product(b_list, c_list, d_list1):
        a = 1
        if (b == c or b == 4 * c or c == 4 * b):
            add_pattern(a, b, c, d)

    for a, b, c, d in itertools.product(a_list, b_list, c_list,
       d_list2):
        if a != 1 and \
           (b, c) not in [(1024, 256), (256, 1024), (128, 512),
              (512, 128), (64, 256), (256, 64)] and \
           (b == c or b == 4 * c or c == 4 * b):
            add_pattern(a, b, c, d)

    return patterns_list
```

Figure D.1: Python code to generate the patterns benchmarked for the execution time in the numerical experiments of Section 5.4.



(a) Batch-size-first (same as Figure 5.5).



(b) Batch-size-last.

Figure D.2: Estimated relative time spent on memory rewritings in `bmm` for the multiplication with $\mathbf{K} \in \Sigma^{\pi}$, for several $\pi = (a, b, c, d)$. We regroup patterns by their value of $(b + c)/(bc)$, and plot a boxplot to summarize the corresponding measurements.

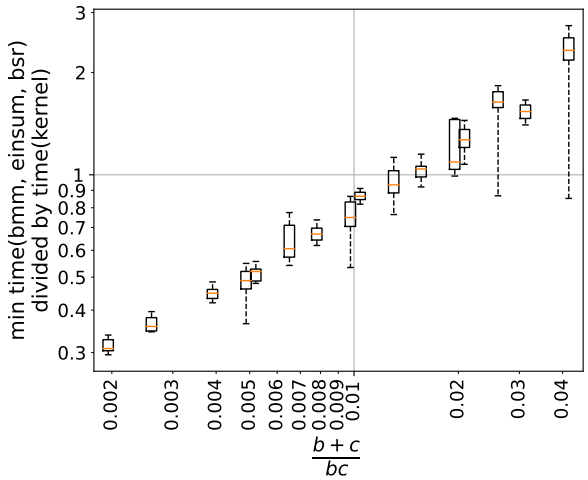
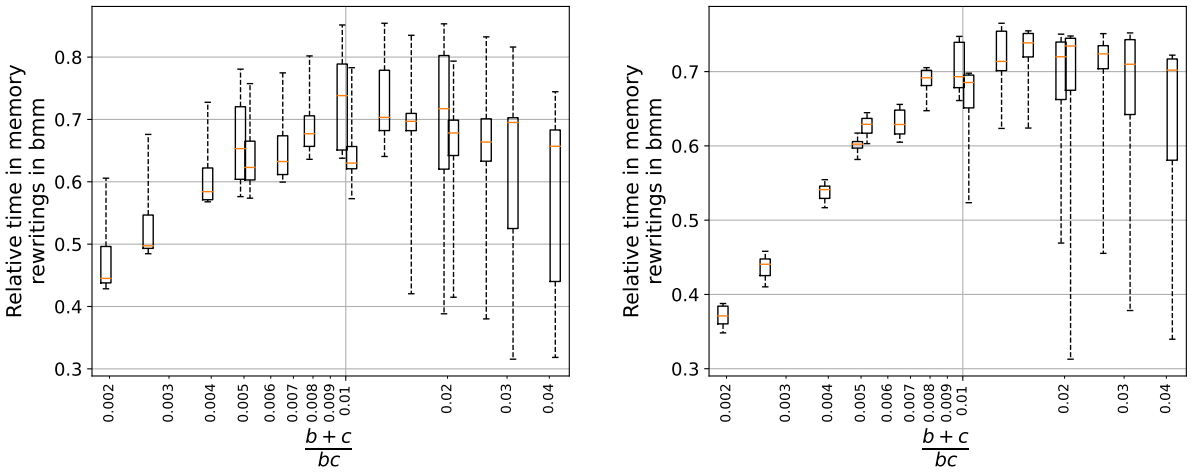


Figure D.3: Speedup factor of `kernel` compared to min(`bmm`, `einsum`, `bsr`) in *half-precision*. For each implementation, we take the minimum time between the *batch-size-first* and the *batch-size-last* memory layout. We regroup the (a, b, c, d) patterns by their value of $(b + c)/(bc)$, and use a boxplot to summarize the corresponding measurements. Experiments are carried in *half-precision*.



(a) Batch-size-first. (b) Batch-size-last.

Figure D.4: Estimated relative time spent on memory rewritings in `bmm` for the multiplication with $\mathbf{K} \in \Sigma^{\pi}$, for several $\pi = (a, b, c, d)$. We regroup patterns by their value of $(b + c)/(bc)$, and plot a boxplot to summarize the corresponding measurements. Experiments are carried in *half-precision*.

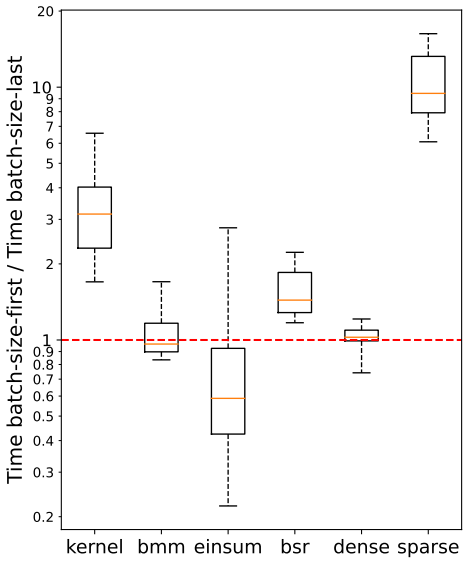


Figure D.7: Boxplots of the ratio $\frac{\text{time of } \textit{batch-size-first}}{\text{time of } \textit{batch-size-last}}$ in *half-precision*.

Appendix E

Supplementary material for Chapter 7

E.1 Details on the experiments

This section provides additional details on the experiments of Section 7.2 and Section 3.5. I start by describing the general setup of the experiments, and then provide specific details for each chapter in Appendix E.1.1 and Appendix E.1.2. Let me recall that the code is available at `github.com/agonon/pathnorm_toolkit`.

Model and data. I train a dense ResNet18 [He et al., 2016] on ImageNet-1k, using 99% of the 1,281,167 images of the training set for training, the other 1% for validation. The PyTorch code for normalization at inference is standard:

```
inference_normalization = transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize(
            mean=[0.485, 0.456, 0.406],
            std=[0.229, 0.224,  0.225]
            ),
    ])
```

Optimization. I use SGD for 90 epochs, learning rate 0.1, weight-decay 0.0001, batch size 1024, and a multi-step scheduler where the learning rate is divided by 10 at epochs 30, 60 and 80. The epoch out of the 90 ones with maximum validation top-1 accuracy is considered as the final epoch. Doing 90 epochs took me about 18 hours on a single A100-40GB GPU.

Pruning. At the end of the training phase, I prune (i.e. set to zero) $p\%$ of the remaining weights of each convolutional layer, and $\frac{p}{2}\%$ of the final fully connected layer for a layerwise method. For a global pruning method, I prune the same amount of weights but globally. I save the mask and rewind the weights to their values after the first 5 epochs of the dense network, and train for 85 remaining epochs. Section 3.5 discusses results obtained by applying this pruning method *once*, while Section 7.2 discusses results relative to a

varying number k of iterations corresponding to Figure E.2. This exactly corresponds to the hyperparameters and pruning algorithm of the lottery ticket literature [Frankle et al., 2021].

E.1.1 Details specific to Section 7.2

Details for Table 7.1. I estimate $B = 2.6400001$ by taking the maximum of the L^∞ norms of the training images normalized for inference.

For Table 7.1, I consider ResNets. They have a single max-pooling layer of kernel size 3×3 so that $K = 9$. The depth is $D = 3 + \#$ basic blocks $\times$ # conv per basic block, where 3 accounts for the conv1 layer, the average-pooling layer, the fc layer, and the rest accounts for all the convolutional layers in the basic blocks. Note that the average-pooling layer can be incorporated into the next fc layer as it only contains identity neurons, see the paragraph "Improving Theorem 4.4.1" right after Theorem 4.4.1, so we can actually consider $D = 2 + \#$ basic blocks $\times$ # conv per basic block. Table E.1 details the relevant values related to basic blocks.

Table E.1: Number of basic blocks, of convolutional layer per basic blocks and associated D for ResNets [He et al., 2016, Table 1].

ResNet	18	34	50	101	152
# basic blocks	8	16		33	50
# conv per basic block	2		3		
D	18	34	50	101	152

Pretrained ResNets. The PyTorch pretrained weights that have been selected are the ones with the best performance: `ResNetX_Weights.IMAGENET1K_V1` for ResNets 18 and 34, and `ResNetX_Weights.IMAGENET1K_V2` otherwise.

Choice of $\gamma > 0$ for Theorem 4.1.2. In Equation (4.14), note that there is a trade-off when choosing $\gamma > 0$. Indeed, the first term of the right-hand side is non-decreasing with γ while the second one is non-increasing. The first term is simply the proportion of datapoints that are not correctly classified with a margin at least equal to γ. Defining the margin of input i on parameters θ to be $R_\theta(\mathbf{X}_i)_{\mathbf{Y}_i} - \arg\max_{c \neq \mathbf{Y}_i} R_\theta(\mathbf{X}_i)_c$, this means that the first term is (approximately) equal to q if $\gamma = \gamma(q)$ is the q-quantile of the distribution of the margins over the training set.

Note that since the second term in Equation (4.14) is of order $1/\sqrt{n}$, it would be desirable to choose the $1/\sqrt{n}$-quantile (up to a constant) for γ. However, this is not possible in practice as soon as the training top-1 accuracy is too large compared to $1/\sqrt{n}$ (eg. on ImageNet). Indeed, if the training top-1 error is equal to $e \in [0, 1]$, then at least a proportion e of the data margins should be negative[1] so that any q-quantile with $q < e$ is negative and cannot be considered for Theorem 4.1.2

The distribution of the margins on the training set of ImageNet can be found in Figure E.1. The maximum training margin is roughly of size 30, which is insufficient to

[1] A data margin is negative if and only if it is misclassified.

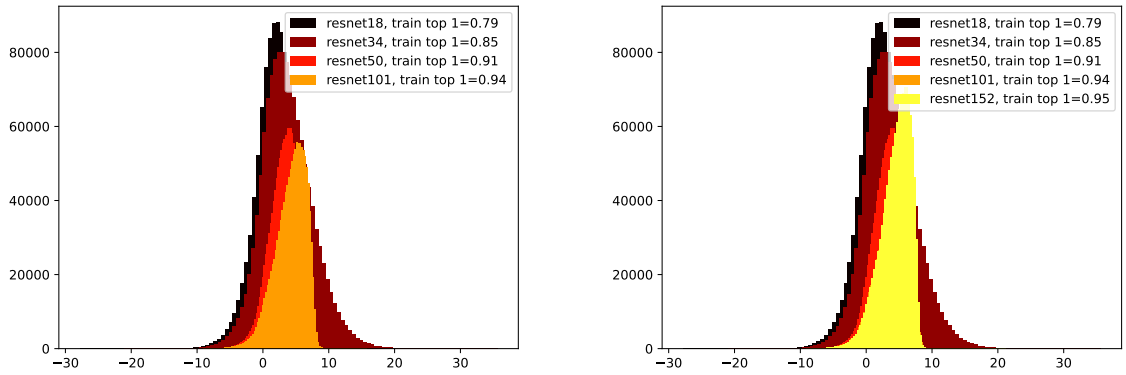


Figure E.1: Distribution of the margins on the training set of ImageNet, with the pretrained ResNets available on PyTorch.

compensate the size of the ℓ^1-path-norm of pretrained ResNets reported in Table 7.2. For $\gamma > 30$, the first term of the right-hand side of Theorem 4.1.2 is greater than one, so that the bound is not informative. This shows that there is no possible choice for $\gamma > 0$ that makes the bound informative on these pretrained ResNets. Table E.2 reports a quantile for these pretrained ResNets.

Table E.2: The q-quantile $\gamma(q)$ for $q = \frac{1}{3}e + \frac{2}{3}$, with e being the top-1 error, on ImageNet, of pretrained ResNets available on PyTorch.

ResNet	18	34	50	101	152
$\gamma(q)$	5.0	5.6	4.2	5.6	5.8

Details for increasing the train size. Instead of training on 99% of ImageNet ($n = 1268355$), I trained a ResNet18 on $n/2^k$ samples drawn at random, for $1 \leqslant k \leqslant 5$. For each given k, the results are averaged over 3 seeds. The hyperparameters are the same as for sparse networks (except that I do not perform any pruning here): 90 epochs etc. Results are in Figure E.3.

E.1.2 Details specific to Section 3.5

Random rescaling. Consider a pair of consecutive convolutional layers in the same basic block of the ResNet18 architecture, for instance the ones of the first basic block: `model.layer1[0].conv1` and `model.layer1[0].conv2` in PyTorch, with `model` being the ResNet18. Denote by C the number of output channels of the first convolutional layer, which is also the number of input channels of the second one. For each channel $c \in [\![1, C]\!]$, I choose uniformly at random a rescaling factor $\lambda \in \{1, 128, 4096\}$ and multiply the output channel c of the first convolutional layer by λ, and divide the input channel c of the second convolutional layer by λ. In order to preserve the input-output relationship, I also multiply by λ the running mean and the bias of the batch normalization layer that

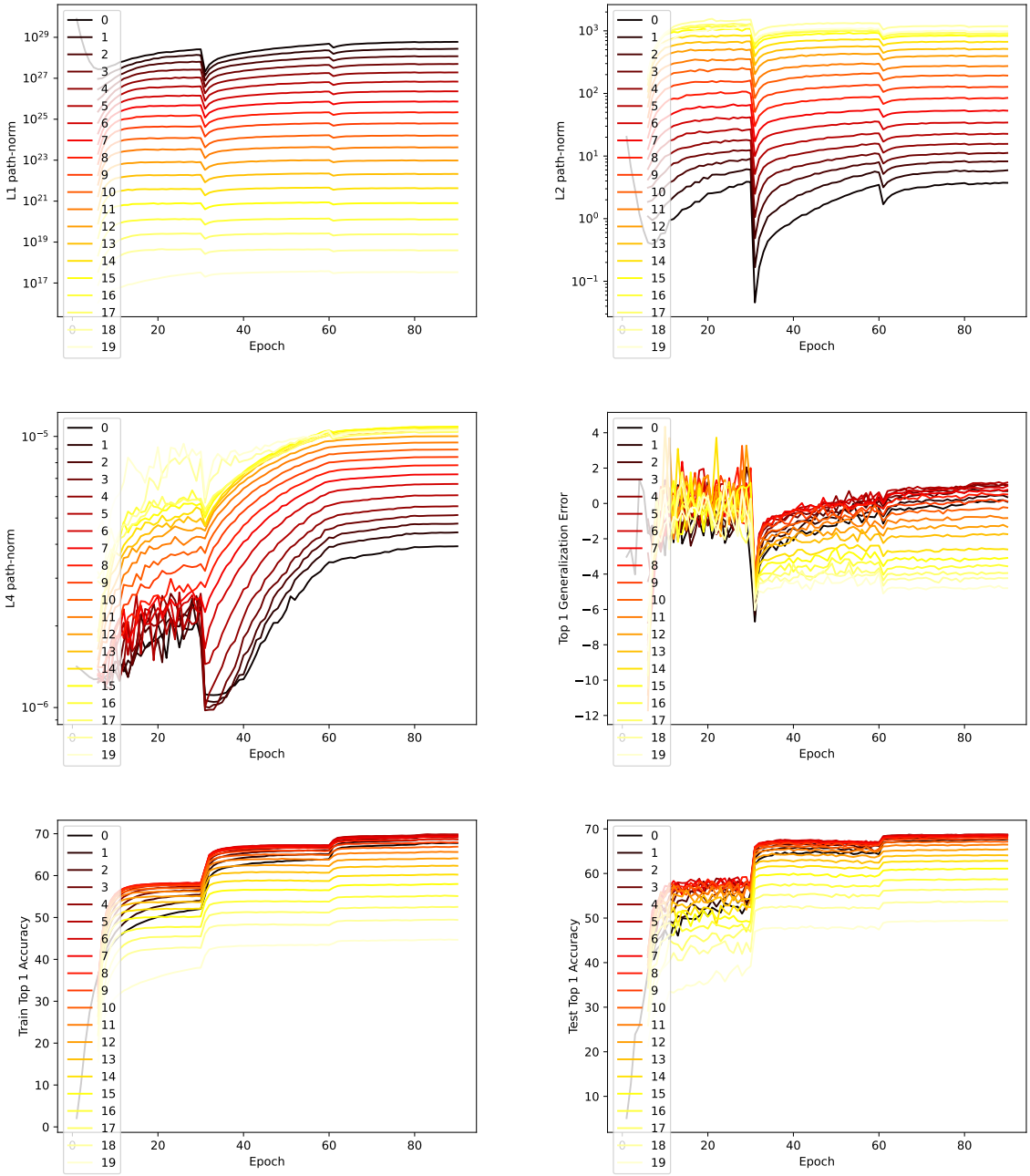


Figure E.2: ℓ^q-path-norm ($q = 1, 2, 4$), test top-1 accuracy, training top-1 accuracy, and the top-1 generalization error (difference between test top-1 and train top-1) during the training of a ResNet18 on ImageNet. The pruning iteration is indicated in legend, with 0 corresponding to the dense network. The color also indicates the degree of sparsity: from dense (black) to extremely sparse (yellow).

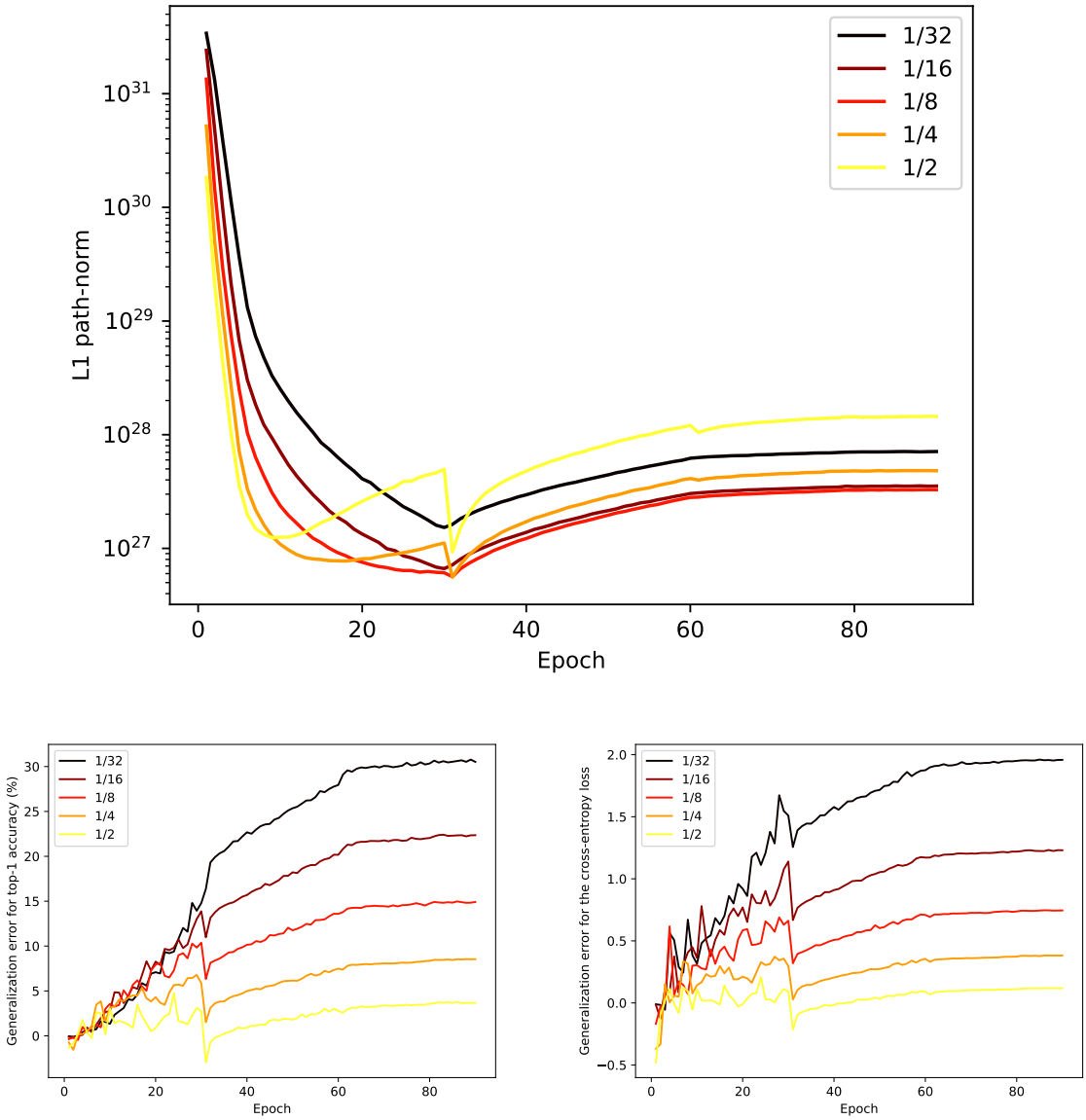


Figure E.3: ℓ^1-path-norm, and empirical generalization errors for both the top-1 accuracy and the cross-entropy during the training of a ResNet18 on a subset of the training images of ImageNet. The legend indicates the size of the subset considered, e.g., $1/m$ corresponds to $1/m$ of 99% of ImageNet, leaving the other 1% out for validation. The color also indicates the size of the subset: from small (black) to large (yellow).

is in between (`model.layer1[0].bn1` in the previous example). Here is an illustrative Python code (that should be applied to the correct layer weights as described above):

```
    factors = np.array([1, 128, 4096])

    out_channels1, _, _, _ = weights_conv1.shape

    for out in range(out_channels1):
                factor = np.random.choice(factors)
                weights_conv1[out, :, :, :] *= factor
                weights_conv2[:, out, :, :] /= factor
                running_mean[out] *= factor
                bias[out] *= factor
```

E.2 Existing evaluations of the path-norms in the literature

I could not find reported numerical values of the path-norm except for toy examples [Dziugaite, 2018, Furusho, 2020, Zheng et al., 2019]. The details are given below. Remember also that before Theorem 3.1.1, there were no known formula to compute the path-norm on modern networks (previous formulas were false if directly extended in the presence of max-pooling neurons).

Details on previous numerical evaluations of path-norms. In Dziugaite [2018, Section 2.9.1] is reported numerical evaluations after 5 epochs of SGD on a one hidden layer network trained on a binary variant of MNIST. Furusho [2020, Figure 9 and Section 3.3.1] deals with 1d regression with 5 layers and 100 width. Experiments in Zheng et al. [2019] are on MNIST. Note that it is not clear whether the path-norm used in Zheng et al. [2019] corresponds to the one defined in Definition 2.3.3. Indeed, the references given in Zheng et al. [2019] for the definition of the path-norm are both Neyshabur et al. [2015] and Neyshabur et al. [2017], but these two papers have two different definitions of the path-norm. The one in Neyshabur et al. [2015] corresponds to the norm of the path-lifting as defined in Definition 2.3.3 (but in simpler settings: no pooling etc.), while the one in Neyshabur et al. [2017] corresponds to the latter divided by the margin of the estimator.

For completeness, let me also mention that it is reported in Dziugaite et al. [2020], Jiang et al. [2020] whether the path-norm correlates with the empirical generalization error or not, but there is no report of numerical values of the path-norm. In Neyshabur et al. [2017] are reported the quotient of path-norms with margins, but not the path-norms alone.