Université de Montréal

# Deep networks training and generalization: insights from linearization

par

**Thomas George**

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Thèse présentée en vue de l'obtention du grade de
Philosophiæ Doctor (Ph.D.)
en Informatique

January 31, 2023

# Université de Montréal

Faculté des arts et des sciences

Cette thèse intitulée

## Deep networks training and generalization: insights from linearization

présentée par

## Thomas George

a été évaluée par un jury composé des personnes suivantes :

*Simon Lacoste-Julien*
(président-rapporteur)

*Pascal Vincent*
(directeur de recherche)

*Guillaume Lajoie*
(codirecteur)

*Guillaume Rabusseau*
(membre du jury)

*Andrew Saxe*
(examinateur externe)

(représentant du doyen de la FESP)

Ce qui compte, c'est pas l'arrivée, c'est la quête.

---

Orelsan, *La quête*

# Résumé

Bien qu'ils soient capables de représenter des fonctions très complexes, les réseaux de neurones profonds sont entraînés à l'aide de variations autour de la descente de gradient, un algorithme qui est basé sur une simple linéarisation de la fonction de coût à chaque itération lors de l'entrainement. Dans cette thèse, nous soutenons qu'une approche prometteuse pour élaborer une théorie générale qui expliquerait la généralisation des réseaux de neurones, est de s'inspirer d'une analogie avec les modèles linéaires, en étudiant le développement de Taylor au premier ordre qui relie des pas dans l'espace des paramètres à des modifications dans l'espace des fonctions.

Cette thèse par article comprend 3 articles ainsi qu'une bibliothèque logicielle. La bibliothèque NNGeometry (chapitre 3) sert de fil rouge à l'ensemble des projets, et introduit une Interface de Programmation Applicative (API) simple pour étudier la dynamique d'entrainement linéarisée de réseaux de neurones, en exploitant des méthodes récentes ainsi que de nouvelles accélérations algorithmiques. Dans l'article EKFAC (chapitre 4), nous proposons une approchée de la Matrice d'Information de Fisher (FIM), utilisée dans l'algorithme d'optimisation du gradient naturel. Dans l'article Lazy vs Hasty (chapitre 5), nous comparons la fonction obtenue par dynamique d'entrainement linéarisée (par exemple dans le régime limite du noyau tangent (NTK) à largeur infinie), au régime d'entrainement réel, en utilisant des groupes d'exemples classés selon différentes notions de difficulté. Dans l'article NTK alignment (chapitre 6), nous révélons un effet de régularisation implicite qui découle de l'alignement du NTK au noyau cible, au fur et à mesure que l'entrainement progresse.

**Mots-clés.** Apprentissage profond, réseaux de neurones, généralisation, optimisation, théorie de l'apprentissage

# Abstract

Despite being able to represent very complex functions, deep artificial neural networks are trained using variants of the basic gradient descent algorithm, which relies on linearization of the loss at each iteration during training. In this thesis, we argue that a promising way to tackle the challenge of elaborating a comprehensive theory explaining generalization in deep networks, is to take advantage of an analogy with linear models, by studying the first order Taylor expansion that maps parameter space updates to function space progress.

This thesis by publication is made of 3 papers and a software library. The library NNGeometry (chapter 3) serves as a common thread for all projects, and introduces a simple Application Programming Interface (API) to study the linearized training dynamics of deep networks using recent methods and contributed algorithmic accelerations. In the EKFAC paper (chapter 4), we propose an approximate to the Fisher Information Matrix (FIM), used in the natural gradient optimization algorithm. In the Lazy vs Hasty paper (chapter 5), we compare the function obtained while training using a linearized dynamics (e.g. in the infinite width Neural Tangent Kernel (NTK) limit regime), to the actual training regime, by means of examples grouped using different notions of difficulty. In the NTK alignment paper (chapter 6), we reveal an implicit regularization effect arising from the alignment of the NTK to the target kernel as training progresses.

**Keywords.** Deep learning, neural networks, generalization, optimization, learning theory

# Contents

# List of Tables

# List of Figures

# List of acronyms and abbreviations

| | |
|---|---|
| AI | Artificial Intelligence |
| API | Application Programming Interface |
| BN | Batch Normalization (Ioffe & Szegedy, 2015) |
| C-scores | Consistency scores (Jiang et al., 2021) |
| CIFAR | Canadian Institute For Advanced Research |
| CKA | Centered Kernel Alignment (Cristianini et al., 2001) |
| GPT | Generative Pre-trained Transformer (Brown et al., 2020) |
| GPU | Graphical Processing Unit is a hardware component with support for vectorized computations |
| EKFAC | Eigenvalue corrected Kronecker-Factored Approximate Curvature (George et al., 2018) |
| FIM | Fisher Information Matrix |

| | |
|---|---|
| ImageNet | ImageNet is a popular challenge based on image classification and segmentation |
| KFAC | Kronecker-Factored Approximate Curvature (Martens & Grosse, 2015) |
| KFE | Kronecker-Factored Eigenbasis (George et al., 2018) |
| KL divergence | the Kullback-Leibler divergence is a measure of how different are two probability distributions |
| MLP | Multi-Layer Perceptron |
| MNIST | Mixed National Institute of Standards and Technology refers to the dataset of handwritten digits (LeCun et al., 2010) |
| NNGeometry | Neural Network Geometry (George, 2021) |
| NTK | Neural Tangent Kernel (Jacot et al., 2018) |
| ResNet | Residual Networks (He et al., 2016a) |
| ReLU | Rectified Linear Units |
| RKHS | Reproducing Kernel Hilbert Space |

| | |
|---|---|
| SGD | Stochastic Gradient Descent |
| SVD | Singular Value Decomposition |
| SVM | Support Vector Machine |
| VGG | Visual Geometry Group, usually refers to a convolutional network architecture (Simonyan & Zisserman, 2015) |

# Acknowledgements

Je voudrais tout d'abord remercier Pascal Vincent pour m'avoir fait confiance et m'avoir guidé pendant mes années de doctorat, ainsi que Guillaume Lajoie qui a repris le flambeau de la co-supervision avec brio. Pour leurs précieux conseils et leurs encouragements, merci.

A warm thank you to Andrew Saxe, Guillaume Rabusseau, and Simon Lacoste-Julien for agreeing to be part of my thesis committee and for taking the time to review this manuscript.

Pour les discussions intéressantes et fructueuses, le travail parfois acharné, les bons moments et les plus difficiles, merci à mes co-auteurs Nicolas Ballas, Aristide Baratin, Xavier Bouthillier, Pascal Vincent, Guillaume Lajoie, R Devon Hjelm, Simon Lacoste-Julien, César Laurent. Plus généralement, j'ai croisé des personnes formidables pendant ces années au Mila. Pour les aventures dans le grand ouest, merci à Alexandre, César, Gabriel, Gauthier et Thomas. Pour m'avoir aidé à reconnaitre les fascistes des libéraux, merci à Camille, Faruk, Florian et Lluís. Pour ces discussions sur la philosophie pascalienne, merci à Ahmed, César, Florian, Hugo, Tom, Vincent et Xavier. Pour les heures passées à préparer les labs et à corriger les copies, merci à Michael et Olivier. Pour les bonnes idées de recherche, merci à Breandan et Mohammad. Merci aussi à Guillaume Rabusseau et Ioannis Mitliagkas de m'avoir donné l'opportunité d'enseigner.

En dehors du Mila, j'ai eu la chance de partager des moments avec des personnes exceptionnelles à Montréal ou ailleurs. Pour m'avoir aidé à braver le climat extrême montréalais, merci à Alex, Alix, Alice, Arthur, Baptiste, Chan, Charlotte, Claire, Claire, David, Doudi, Greg, Guigui, Loulou, Lucie, Manon, Mathilde, Sophie, Suzie, Ted, Twixo, Valou et Jean-Michel pour la BO. Pour les années aux Mines et après, merci à Alberto, Charles, CaJo, Cheimi, Clément, Coline, Florent, Léa, Lénaïc. En préparation de mon expatriation québécoise, j'ai aussi pu compter sur le soutien de Dany, Greg, Greg, Jens, Kristin, Pauline, Noémie, Victor qui ont goûté à mes balbutiements d'anglais à Bristol en Angleterre. Auparavant, mes études supérieures ont débuté sur les chapeaux de roues en compagnie d'Ahmed, Clément, Guilhem, Mohand, Rachel. Sans oublier les racines malakoffiotes et les électrons libres, merci à Alex, Alex, Alex, Alice, Anna, Antoine, Baptiste, Caro, Claire, Damien, Jason, Léo, Manue, Marie, Mehdi, Piche, Romain, Sabine, Thomas, et bien sûr Denise pour la bolo. Merci à mes coéquipiers à l'USMM, puis à Martigua, et enfin aux Celtiques de Montréal qui

ont propulsé ma carrière de handballeur à son apogée avec le titre de champion du Canada en 2016.

Merci à ces professeurs qui comptent pour moi car ils ont éclairé mes choix lorsque des embranchements se sont présentés dans ma vie: Pascal Fautrero, Annie Le Bourhis, Sylvie Gillot, Yves Berthaud, Nicolas Petit, Yoshua Bengio.

Pour leur soutien et leur amour, merci à Christine, Paul-Louis (George, 2005), Clément, Michèle, Nicole, François et Jacques, à mes oncles et tantes, à mes cousines et cousins chéris, et bien sûr merci à Lisa et à sa famille.

**Merci à vous tous.** Chacun à votre manière, vous êtes les co-auteurs de ce bout de vie.

# Chapter 1

# Introduction

I first heard about artificial neural networks during my undergraduate studies in 2012 at École des Mines in Paris. I implemented gradient descent on a hard-coded 2-layers multilayer perceptron (MLP) in C, in order to solve the task MNIST (LeCun et al., 2010), a 10-fold classification problem of grayscale images of handwritten digits. Since then, *deep learning*, as it is now called, underwent dramatic progress, driven by the incremental discovery of new network architectures, the increase in compute and storage capabilities, the availability of data, and the growing interest of private companies and publicly funded research labs. Between 2015 when I first knocked at the door of the Lisa[1] lab, and 2022 now that I am finishing my PhD at Mila's brand-new facilities, the community had grown at least 5-fold, as can e.g. be measured by the number of papers published at NeurIPS, one of the leading conference about deep learning.

This thesis relates my contribution to the field these last 5 years.

## 1.1. High level introduction and motivation

Nowadays, deep learning is used in many commercial applications, where it is often confounded with Artificial Intelligence (AI) in the general public. The most impressive success of deep learning started with image classification, where convolutional architectures such as AlexNet (Krizhevsky et al., 2017), VGG (Simonyan & Zisserman, 2015), and ResNet (He et al., 2015) successfully improved the state of the art on the popular challenge ImageNet (Deng et al., 2009). Since then, deep learning techniques have also conquered automatic text translation (Sutskever et al., 2014), prediction of protein folding structure (Senior et al., 2020), general purpose dialog systems (Brown et al., 2020), game engines such as chess and go (Silver et al., 2018), image generation from a text description (Ramesh et al., 2022) and many other applications.

[1]The Laboratoire d'Informatique des Systèmes Adaptatifs (LISA) was a lab at Université de Montréal before it merged with other Montréal labs to become Mila.

In short, deep learning (Goodfellow et al., 2016) is inspired by biological neurons, with the intuition that arranging many basic units (neurons) in a large network (the neural network) can produce versatile functions able to solve a variety of tasks. These models are tuned using a database of annotated examples, in an analogous way as a child that is taught by their parents to distinguish animal species. This procedure is often referred to as *training* or *learning*. A formal, more precise description of deep learning can be found in Chapter 2.

The tremendous success of deep learning is mostly based on experiments and heuristics. In fact, most improvements at solving tasks with deep learning are often the result of common sense in tweaking the network architecture, and trial and error until a satisfactory predictor can be trained. This empirical progress has far outpaced theory, despite significant effort by the deep learning theory community (e.g. Kawaguchi et al., 2017). A comprehensive theoretical framework has yet to emerge, even for simple datasets and network architectures. This thesis contributes to this goal.

A question remains whether it is even relevant to seek for a theoretical framework explaining deep learning: Even with the lack of it, deep models *work*, that is, on many tasks they give good predictions on previously unseen examples. To some extent, the state of the field is comparable to the steam machine invented in the $18^{\text{th}}$ century, which has preceded the discovery of theoretical principles describing its inner working (i.e. thermodynamics) in the $19^{\text{nth}}$ century.

Leaving aside the purely intellectual challenge of trying to understand how things work that arguably drives every researcher, there are some compelling practical motivations for a comprehensive theoretical understanding of deep learning. In critical applications, such as automated medical procedures or self-driving cars, it is essential to come with generalization guarantees, that quantify how effective an algorithm trained on some data will be when used on previously unseen data. This is particularly true in the out-of-distribution setup, where the system is deployed in a different environment than the one used for training (e.g. a self-driving car trained in North America will likely be erratic if used in Europe because typical traffic signs, car models and crossings are different). Another motivation for advancing theory is to obtain principled approaches for neural network architecture design and training procedures. This is especially relevant as deep learning is becoming ubiquitous, requiring more and more compute capacities in large datacenters, which are very energy-intensive. Lately, deep learning has been following a trend of increasingly bigger models (e.g. $10^{11}$ parameters for GPT-3, Brown et al., 2020), which could be replaced by smaller, more energy efficient models. On a different subject, a concerning failure mode of deep models for image classification is their tendency to treat different subgroups of people differently (e.g. based on their skin color or their gender, as discussed in Buolamwini & Gebru, 2018), or more generally their lack of interpretability.

To tackle the challenge of improving our understanding of deep learning theory, we start from a striking observation: contrasting with the versatility and the complexity of the functions obtained at the end of the training procedure, the optimization algorithm at the heart of most deep learning tasks is in fact very simple: *gradient descent* relies on the linearization of the training loss at every iteration (Rumelhart et al., 1985). This thesis studies a slightly different concept: linearization of the predictor. The works presented focuses on 2 questions:

(1) How to train deep models fast?

(2) What are the principles that govern generalization, *i.e.* how good is a given network at prediction on new examples that were not used for learning?

## 1.2. Challenges

In our attempt at answering these questions, we face the following challenges.

(i) *Failure of traditional statistical learning theory approaches*
In most pre-2017 machine learning textbooks (e.g. Bishop, 2006), statistical learning theory predicts a bias-variance trade-off, where learning a model from a higher capacity function class often results in worse generalization performance. But neural networks have been proven to be capable of approximating any sufficiently regular function (Hornik et al., 1989), and can perfectly fit even a high proportion of incorrectly labeled examples (Zhang et al., 2017a). From this perspective, the high capacity of classes of neural networks contradicts with their impressive generalization ability. During the time of this thesis, this apparent paradox is now no longer considered true (Neal et al., 2019; Belkin et al., 2018). A double descent phenomenon where generalization starts improving again passed a certain threshold in model capacity has been discussed by several authors (Advani & Saxe, 2017; Belkin et al., 2019; Nakkiran et al., 2020), and other examples of high capacity classes of models have been put forth (Belkin et al., 2018; Wyner et al., 2017). The focus has now shifted to the search for implicit bias mechanisms (Neyshabur et al., 2014): among all functions that can be described by a given architecture, which are the functions that are actually learned in a limited amount of time given an optimization algorithm and possibly some form of regularization on the parameters (e.g. weight decay, Hanson & Pratt, 1988) or the data (e.g. the Mixup data augmentation technique, Zhang et al., 2017b).

(ii) *Characterization of the function obtained after training a single network*
Whereas challenge (i) relates to the difficulty of characterizing whole function classes described by given neural network architectures, the complexity of predicting properties of even a single realization from one of these classes a priori is also noteworthy: given a fixed training dataset and starting from given initial parameters, there is

currently no universally accepted accurate modeling of the final solution that does not require fully training the model. This contrasts to e.g. the ridge regression, that admits a closed-form solution. The empirical risk minimization problem of deep learning is non-convex and non-smooth, can have infinitely many minima, and the training dynamics are non-linear. Some simplified settings, however, are amenable to analytical treatment, such as deep linear networks with no activation functions (Saxe et al., 2014; Gidel et al., 2019), large width networks (Du et al., 2019b) up to infinite-width limiting regimes (Jacot et al., 2018; Sirignano & Spiliopoulos, 2020) or using other modelling inspired by kernel methods (Mairal et al., 2014).

(iii) *High dimensionality of the optimization problem*

The empirical risk minimization problem is the mathematical formalization of the process of learning with deep networks. It is a sum over $N$ examples in the training dataset (typically $N$ is in the order $10^4$), of a loss that depends on $P$ parameters (typically $P$ is in the order $10^7$–$10^{10}$), where $N$ and $P$ are large and pose different challenges. We usually address the sum over $N$ by computing stochastic estimates of the required quantities (i.e. the gradients of the loss) using mini-batches of a few examples (Cotter et al., 2011), and possibly some variance reduction method (Johnson & Zhang, 2013; Defazio et al., 2014; Mairal, 2015; Schmidt et al., 2017). The large number of parameters raises a different issue. Optimization methods that counteract ill-conditioned curvature (e.g. the Generalized Gauss-Newton approximation of Newton's method, Schraudolph, 2001) require manipulating large $P \times P$ matrices, which are impractical to invert or even store in memory for typical values of $P$. Practitioners usually resort to diagonal methods such as RMSProp (Tieleman & Hinton, 2012) or Adam (Kingma & Ba, 2015), or use more sophisticated low-rank methods such as $\ell$-BFGS (Liu & Nocedal, 1989).

(iv) *Incorporating the structure of the data*

The input space of neural network functions generally has high cardinality (e.g. even tiny $32 \times 32$ pixels images give an input space of 1024 dimensions). In this space, training examples are scarce: without any additional assumption on the structure of the data, doing machine learning on this data would be hopeless. Modelling this structure is however an additional challenge, especially relevant as deep learning is deemed to be learning its feature representations from data (Olah et al., 2017). Training examples also play different roles, with a skewed distribution of a head that contains the most representative examples, and a tail with examples which are merely memorized (Feldman & Zhang, 2020). Some examples carry some spuriously correlated features that can hinder generalization (Sagawa et al., 2020b), and in the meantime deep networks are able to perfectly fit incorrectly labeled examples while still generalizing well (Zhang et al., 2017a).

## 1.3. Proposed approach

Our proposed approach, and the common thread of this thesis, is the engineer's first instinct when faced with a function that is too complex to analyze: *linearization*. Training a deep networks consists in starting from initial parameters $\mathbf{w}_0$, then running a learning algorithm that iterates through a sequence of parameter values $\{\mathbf{w}_t\}_{0\leq t<T}$ until a satisfactory solution $\mathbf{w}_T$ is found. We write the function obtained at the end of the training procedure as a sum of function iterates:

$$f_{\mathbf{w}_T} = f_{\mathbf{w}_0} + \sum_{t=0}^{T-1} \underbrace{\left(f_{\mathbf{w}_{t+1}} - f_{\mathbf{w}_t}\right)}_{:=g_t} \tag{1.3.1}$$

$g_t$ is the function difference resulting from the step $\delta\mathbf{w}_t$ in parameter space going from $\mathbf{w}_t$ to $\mathbf{w}_{t+1}$. Each of these function iterates is then locally approximated using their first order Taylor expansion in $\mathbf{w}$:

$$g_t(x) = \langle \delta\mathbf{w}_t, \Phi_{\mathbf{w}_t}(x)\rangle + O(\|\delta\mathbf{w}_t\|^2) \tag{1.3.2}$$

This local model is reminiscent of a linear model (e.g. SVM or Logistic Regression) with features obtained by the mapping $\Phi_{\mathbf{w}_t}$ from the input space to a feature space. We focus on studying the series of *tangent features* $\{\Phi_{\mathbf{w}_t}\}_{0\leq t<T}$ as training progresses. Our claim, illustrated in 3 different cases throughout this thesis, is that **local properties of the tangent features lead to global insights on the outcome**.

## 1.4. Summary of contributions

This thesis started with the optimization project EKFAC (Chapter 4) where our goal was to accelerate deep models training. During development of this project, we extensively used the Fisher Information Matrix (FIM) $F = \mathbb{E}_x\left[\Phi_{\mathbf{w}_t}(x)\,\Phi_{\mathbf{w}_t}(x)^\top\right]$, which we later figured was a sibling object to the Neural Tangent Kernel (NTK) $k(x,x') = \langle \Phi_{\mathbf{w}_t}(x), \Phi_{\mathbf{w}_t}(x')\rangle$ subsequently introduced by Jacot et al. (2018). These give a common thread to the thesis, that can be summarized as linearization of the training dynamics with respect to the tunable parameters of deep models.

**1st paper, Chapter 3.** Our first contribution is a software library, called NNGeometry, that proposes an Application Programming Interface (API) on top of PyTorch (Paszke et al., 2019b) to manipulate FIMs, NTKs and Jacobians used in the other projects. The main challenge is the dimensionality of these matrices or tensors, that would naively require too much memory and too much compute to handle, even with modern GPUs. In order to address this challenge, we rely on previously published approximate techniques (Martens et al., 2018; Ollivier, 2015; George et al., 2018) that can be accessed through a unified interface, and we introduce new algorithmic enhancements to perform operations implicitly, allowing to use

NNGeometry even on large networks. A second challenge is that PyTorch, as well as most other deep learning frameworks (Bergstra et al., 2011; Abadi et al., 2016), are specialized to the very task of training deep networks, whereas in order to compute FIMs, NTKs and Jacobians, we need to hack their internal mechanisms.

**2nd paper, Chapter 4.** We introduce an approximate to the FIM, called EKFAC, that drastically reduces the memory and the computational requirements to perform standard linear algebra operations. Compared to the previously published method KFAC (Martens & Grosse, 2015; Grosse & Martens, 2016) from which it is inspired, EKFAC gives a more accurate estimate of the FIM, and allows for fast re-estimation when parameters vary slowly. We use EKFAC as a natural gradient (Amari, 1998) optimizer in order to accelerate training. In a variety of setups, EKFAC improves over other benchmarked methods in terms of progress per iteration, and progress per elapsed time.

**3rd paper, Chapter 5.** In a variety of settings, starting from the same initialization, we empirically compare the training dynamics of 2 regimes of interest: i) the full standard deep learning training regime for which we do not yet have tools for a precise analysis even for simple 2-layers networks with non-linearities and ii) the linearized regime at initialization using the feature map $\Phi_{\mathbf{w}_0}$. This linearized regime ii) boils down to a kernelized linear model that is amenable to analysis, for underparameterized models (e.g. in the popular textbook Bishop, 2006) or more recently in the overparameterized regime (e.g. in the review by Bartlett et al., 2021). We propose to normalize training progress and compare the trajectory of both regimes in the function space (as opposed to the parameter space), by evaluating the functions using examples ranked by different notions of difficulty, instead of performing a Fourier analysis as in Rahaman et al. (2019); Zhang et al. (2021). This reveals a comparatively more pronounced sequentialization for the non-linear regime, where easy and more representative examples are prioritized while more difficult and rarer examples are essentially ignored during a first phase of training, suggesting an implicit bias of deep learning towards directions supported by these most representative examples. We reproduce these empirical findings in a simplified 2-layers parameterization that is amenable to analytical treatment in 3 setups that mimic our larger scale experiments.

**4th paper, Chapter 6.** We analyze the time-varying NTK or equivalently the tangent features, throughout the course of training. Using basic algebraic observations, we link the spectrum of the NTK to the evolution of the learned function. We empirically observe that the effective rank of the NTK diminishes as training progresses, which we interpret as a selection of a few number of directions in the function space. Further, we observe that the NTK aligns to the target kernel, when evaluated on both the training or test datasets. This alignment has long been used as a criterion for selecting kernels that generalize well in kernel learning (Cristianini et al., 2001). The increase in alignment suggests an implicit

regularization mechanism, that we exploit by deriving a new Rademacher complexity measure for families of functions built from a sequence of kernels. We perform a sensitivity analysis of this measure as we train different models with a varying number of parameters/a varying amount of noise in the training dataset, and we observe that it correlates well with the generalization.

## 1.5. Other contributions not included in the thesis

During my PhD at Mila, I also contributed to the following projects, that are not part of this thesis:

- In *Revisiting loss modelling for unstructured pruning* (Laurent et al., 2020), we evaluated pruning methods for deep networks that select which parameters to prune using saliency scores based on minimally changing the training loss after pruning.
- In *Continual learning in deep networks: an analysis of the last layer* (Lesort et al., 2021), we studied the influence of the parameterization of the last layer on the ability of deep learning networks to learn tasks sequentially in continual learning setups.
- In *The dynamics of functional diversity throughout neural network training* (Zamparo et al., 2021), we studied the role of different sources of randomness (different initialization, different mini-batch order, ...) in finding diverse trained models, in the context of (deep) ensembles.

# Chapter 2

# Brief introduction to machine learning and deep networks

In this section, we briefly present the concepts and notation that we will be using throughout the document. This section is intentionally left short and compact since we expect the reader to be already familiar with machine learning and deep networks. It serves as a refresher and a presentation of notations.

## 2.1. Supervised machine learning

Supervised machine learning consists in estimating an unknown function $f : \mathcal{X} \to \mathcal{Y}$ from an input space $\mathcal{X}$ to an output space $\mathcal{Y}$, given a dataset $\mathcal{D}$ of (possibly noisy) realizations of $f$ at given values of $x \in \mathcal{X}$, generally independently sampled from a probability density $p(x)$. $\mathcal{D} = \{(x_i, y_i = f(x_i)) : i \in [\![1,n]\!]\}$ is called the *training set*.

**Example 1.** In the CIFAR10 image classification task (Krizhevsky & Hinton, 2009), $\mathcal{X}$ is the space of $32 \times 32 \times 3$ images arranged as a 3d tensors (2 spatial dimensions and 3 channels for RGB layers), $\mathcal{Y}$ is the discrete space of 10 classes $\{\text{airplane}, \text{automobile}, \text{bird}, \text{cat}, \text{deer}, \text{dog}, \text{frog}, \text{horse}, \text{ship}, \text{truck}\}$. $p(x)$ would be a probability density that puts mass on real images belonging to the classes in $\mathcal{Y}$ and 0 elsewhere. In practice $p(x)$ is never explicitly used since modelling it is a challenge by itself[1]. We instead have access to the dataset of samples $\mathcal{D}$ of 60.000 pictures of real objects with their associated true class. The task is to find $f$ that maps an image to its true class.

**Expected risk.** Given a family of functions $\mathcal{F}$ called the *hypothesis space*, our criterion for choosing a candidate function $f^*$ is to seek for a minimizer of the *expected risk* $\mathcal{L}$ obtained

[1] in some sense $p(x)$ is given by the laws of physics that govern the world.

using a loss function $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}^+$ that depends on the task:

$$f^* = \arg\min_{\hat{f} \in \mathcal{H}} \underbrace{\mathbb{E}_{x \sim p(x)} \left[ \ell \left( \hat{f}(x), f(x) \right) \right]}_{:= \mathcal{L}\left(\hat{f}\right) \text{ (expected risk)}} \tag{2.1.1}$$

We will refer to the process of finding $\hat{f}$ that minimizes the expected risk $\mathcal{L}\left(\hat{f}\right)$ as *training* or *learning*.

**Empirical risk.** In real world applications we do not know the true distribution $p(x)$ but we only have access to a limited number of realizations $\{f(x_i)\}_{i \in [\![1,n]\!]}$ arranged in the dataset $\mathcal{D}$. We are thus unable to compute the expectation over $x$ that appears in the expected risk, and we estimate it using a quantity called the *empirical risk*:

$$L\left(\hat{f}\right) \coloneqq \frac{1}{n} \sum_{i=1}^{n} \ell \left( \hat{f}(x_i), y_i \right) \tag{2.1.2}$$

Our main interest is not to learn by heart the examples of the dataset $\mathcal{D}$ : we want our candidate function to be able to generalize to unseen values in the input space $\mathcal{X}$. A good candidate $\hat{f}$ should simultaneously minimize the empirical risk and make sure that the discrepancy between the expected risk and the empirical risk is reasonably small. This discrepancy $\mathcal{L}\left(\hat{f}\right) - L\left(\hat{f}\right)$ is called the *generalization gap*.

However, even supposing that the true function $f$ belongs to the hypothesis space $\mathcal{F}$, then we are not guaranteed to recover it. In practice, the problem is often under specified (D'Amour et al., 2020): $\mathcal{F}$ possibly includes infinitely many functions that perfectly minimize the loss on the training examples but give different predictions outside the training dataset. Depending on the choice of minimizer made by the learning algorithm, we get a different prediction accuracy on unseen examples not included in the training set.

**Parametric models.** In this document we will focus on parametric functions $f_{\mathbf{w}}$ where $\mathbf{w}$ is a vector of parameters of dimension $d$. The hypothesis class is the family of such functions $\mathcal{F} = \{f_{\mathbf{w}} : \mathbf{w} \in \mathcal{C}\}$ where $\mathcal{C} \subset \mathbb{R}^d$ defines the set of accepted values for $\mathbf{w}$. Instead of manipulating raw functions, we can now explore the hypothesis space by varying $\mathbf{w}$. We define a *cost function*: $C : \mathbb{R}^d \to \mathbb{R}^+$ that maps a vector $\mathbf{w}$ to the empirical risk of the corresponding function $f_{\mathbf{w}}$:

$$C(\mathbf{w}) \coloneqq L(f_{\mathbf{w}}) \tag{2.1.3}$$

In the case of parametric functions, supervised machine learning algorithms minimize the empirical risk $L$ by varying $\mathbf{w}$ using *optimization* techniques on the cost function $C$, while making sure that the generalization gap stays small. One way of controlling this generalization gap, is to *regularize* the optimization algorithm, by biasing it toward certain solutions, e.g. that impose some regularity on the function.

## 2.2. Linear models and RKHS

Linear models present a first parameterization both very expressive in that they are able to represent a diversity of functions, and also simpler to analyze using well understood theoretical tools.

Linear models are functions $f_{\mathbf{w}}(x) = \langle \mathbf{w}, \phi(x) \rangle$ written as an inner product between a parameter vector $\mathbf{w} \in \mathbb{R}^p$ and a data representation or *feature map* $\phi : \mathcal{X} \to \mathbb{R}^p$ that maps input examples to a feature space.

**Example 2** (Ridge regression)**.** Given a feature function $\phi : \mathcal{X} \to \mathbb{R}^d$ that maps input vectors $x$ to their data representation $\phi(x)$, we define our hypothesis space as the family of functions that can be written as an inner product between a parameter vector $\mathbf{w} \in \mathbb{R}^d$ and the feature map $\phi$: $\mathcal{F} = \left\{ f_{\mathbf{w}} = \langle \mathbf{w}, \phi \rangle, \mathbf{w} \in \mathbb{R}^d \right\}$. In the case of regression, $\mathcal{Y}$ identifies as $\mathbb{R}$ and the loss function $\ell$ is the squarred error between a predicted value and the true value: for a sample $(x_i, y_i) \in \mathcal{D}$, $\ell(f_{\mathbf{w}}(x_i), y_i) = (f_{\mathbf{w}}(x_i) - y_i)^2$. We regularize by penalizing the euclidean norm of $\mathbf{w}$ (Krogh & Hertz, 1992). We obtain the optimal value of $\mathbf{w}$ that minimizes the cost function in closed-form:

$$\mathbf{w}^* \coloneqq \arg\min_{\mathbf{w}} C(\mathbf{w}) + \lambda \|\mathbf{w}\|_2^2 = \left( \Phi^\top \Phi + \lambda n I \right)^{-1} \Phi^\top \mathbf{y} \tag{2.2.1}$$

where $\Phi = \begin{pmatrix} - \phi(x_1)^\top - \\ \vdots \\ - \phi(x_n)^\top - \end{pmatrix}$ is called the *design matrix* of features for all examples in $\mathcal{D}$, and $\mathbf{y} = (y_1, \ldots, y_n)^\top$ is their corresponding true output.

**Kernels.** Instead of working with parameter vectors, it is sometimes more convenient to write $\mathbf{w}$ as a linear combination of the features given by training set examples weighted by $\mathbf{a} \in \mathbb{R}^n$:

$$\mathbf{w} = \Phi^\top \mathbf{a} \tag{2.2.2}$$

We define $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ the *kernel* function $k(x, x') = \langle \phi(x), \phi(x') \rangle$ and we can rewrite $f_{\mathbf{w}}$ as a linear combination of $k$ applied to training set examples:

$$f_{\mathbf{w}}(x) = \sum_{i=1}^{n} a_i \langle \phi(x_i), \phi(x) \rangle = \sum_{i=1}^{n} a_i k(x_i, x) \tag{2.2.3}$$

Kernels can be directly defined without explicitely computing $\phi(x)$. This is particularly useful when the dimension of the parameter space is larger than the number of training examples (in fact it can even be $\infty$).

**RKHS.** For a given symmetric positive semidefinite kernel $k$, the space of functions

$$\mathcal{F} = \left\{ f(\cdot) = \sum_{i=1}^{n} c_i k(x_i, \cdot), n \in \mathbb{N}, \forall i \in [\![1, n]\!], c_i \in \mathbb{R}, x_i \in \mathcal{X} \right\} \tag{2.2.4}$$

is called the Reproducing Kernel Hilbert Space (RKHS) associated with $k$. Mercer's theorem states that there exist an orthonormal basis $\{u_j\}$ of $\mathcal{F}$ and nonnegative eigenvalues $\{\lambda_i\}$ such that:

$$k\left(x,x'\right) = \sum_{j=1}^{+\infty} \lambda_j u_j\left(x\right) u_j\left(x'\right) \tag{2.2.5}$$

Given a kernel, we can thus get back to defining feature maps $\phi : \mathcal{X} \to \mathbb{R}^d$ using the eigenfunctions $\phi_j = \sqrt{\lambda_j} u_j$. For a function $f \in \mathcal{F}$, we define the RKHS norm $\|\cdot\|_{RKHS}$ by means of the inner product $\|f\|^2_{RKHS} = \langle f,f\rangle_{RKHS}$ and in particular for parametric functions $f_{\mathbf{w}} = \sum_j w_j u_j$, :

$$\|f_{\mathbf{w}}\|^2_{RKHS} = \langle f_{\mathbf{w}},f_{\mathbf{w}}\rangle \tag{2.2.6}$$

$$= \left\langle \sum_j w_j u_j, \sum_j w_j u_j \right\rangle \tag{2.2.7}$$

$$= \sum_j w_j^2 \tag{2.2.8}$$

For a function $f_{\mathbf{w}} = \langle \mathbf{w},\phi\rangle$, and given 2 points $x$ and $x'$ in $\mathcal{X}$, Cauchy-Schwarz's inequality allows us to use the RKHS norm and get an upper bound on the variation of $f_{\mathbf{w}}$:

$$\left|f_{\mathbf{w}}\left(x\right) - f_{\mathbf{w}}\left(x'\right)\right| \quad \leq \quad \|\mathbf{w}\|_{\mathcal{H}} \left\|\phi\left(x\right) - \phi\left(x'\right)\right\|_{\mathcal{H}} \tag{2.2.9}$$

In particular, we seek candidate functions $f$ that are sensitive to variations in relevant feature directions while ignoring irrelevant ones. From this we see that for a given $\phi$, regularizing the RKHS norm $\|\mathbf{w}\|_{\mathcal{H}}$will regularize the function $f_{\mathbf{w}}$.

**Example 3** (Ridge regression continued)**.** Using the same setup as in example 2, we can write the solution of the ridge regression algorithm using the kernel function:

$$f^*\left(\cdot\right) = \sum_{i=1}^{n} c_i k\left(x_i,\cdot\right) \text{ with } \mathbf{c} = \left(K + \lambda n I\right)^{-1} \mathbf{y} \tag{2.2.10}$$

where $K_{ij} = k\left(x_i,x_j\right)$ is a $n \times n$ matrix called the *Gram matrix*.

## 2.3. Deep networks

In the last section, the hypothesis space contained linear models of a parameter $\mathbf{w}$. Neural networks, being able to represent more complex functions without requiring to hand-design features, have proven very successful at solving many different tasks. We now describe this family of functions.

### 2.3.1. Neural networks

Neural networks are a family of parametric functions arranged as a composition of functions called *layers*:

$$f_{\mathbf{w}}(x) \;=\; g^{(D)}_{\mathbf{w}_D} \circ g^{(D-1)}_{\mathbf{w}_{D-1}} \circ \ldots \circ g^{(1)}_{\mathbf{w}_1}(x) \tag{2.3.1}$$

$D$ is called the *depth*: it is the number of such composed functions.

For the current document, we will restrict our study to feed-forward networks. For layer $i$, we define the following elements: $a^{(i)}$ is the activation, $\sigma^{(i)}$ is the activation function or non-linearity, $W^{(i)}$ is the weight matrix and we append a scalar constant 1 to the vector $a^{(i-1)}$ to form $\tilde{a}^{(i-1)} = \begin{pmatrix} a^{(i-1)} \\ 1 \end{pmatrix}$ in order to add a bias term given by the last row of the matrix $W^{(i)}$ (sometimes called *homogeneous* notation). Each layer maps activation $a^{(i-1)}$ from the previous layer, to a new transformed activation $a^{(i)}$, given by the following expression:

$$a^{(i)} = g^{(i)}_{\mathbf{w}_i}\left(a^{(i-1)}\right) = \sigma^{(i)}\left(s^{(i)}\right) = \sigma^{(i)}\left(W^{(i)}\begin{pmatrix} a^{(i-1)} \\ 1 \end{pmatrix}\right) = \sigma^{(i)}\left(W^{(i)}\tilde{a}^{(i-1)}\right) \tag{2.3.2}$$

We identify $a^{(0)}$ as the input vector $x \in \mathcal{X}$. Each activation $a^{(i)}$ gives a new representation of $x$ (Bengio et al., 2013). After training a deep networks at solving a task, we observe in practice that these representations become more and more useful for solving the task at hand. See for instance Olah et al. (2017) for a visual presentation of such representations for an image classification task.

This arrangement of linear mappings interleaved with non-linear transformations yields a particular structure in the geometry of the function space, that we describe more precisely and exploit in Chapter 4.

**Overparameterization.** In current real applications, the depth $D$ ranges from 1 to the order of 1000 (He et al., 2016b), and the size of the vector of parameters $\mathbf{w}$ can be as large as in the order of $d = 10^8$ (Simonyan & Zisserman, 2015), hence the designation *deep* networks. By contrast, the training dataset $\mathcal{D}$ typically contains a number of examples in the order of $10^4$. Given the expressiveness of deep networks with as many parameters (Hornik et al., 1989), the hypothesis class is large enough to include deep networks that are able to learn the training examples by heart while behaving erratically for unseen examples. In practice, trained networks give surprisingly good predictions. The reason for their success is not yet fully understood. We contribute some insights in Chapters 5 and 6.

### 2.3.2. Gradient descent

Deep networks are usually trained using (variants of) the gradient descent algorithm on the scalar cost $C(\mathbf{w})$. The parameters are iteratively updated in order to minimize the empirical risk where each update is given by the *steepest descent* direction of the empirical risk with respect to the parameters:

$$\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} - \eta \nabla C\left(\mathbf{w}^{(t)}\right) \tag{2.3.3}$$

where $\eta$ is the scalar learning rate. The gradients $\nabla C\left(\mathbf{w}^{(t)}\right)$ are obtained using an efficient implementation of the chain rule called the backpropagation algorithm (Rumelhart et al., 1985).

We here implicitly hid the choice of the Euclidean distance in the parameter space in order to define gradients. In 4.2.1 in Chapter 4, we further discuss this choice and describe the natural gradient algorithm that compute gradients using more relevant distances.

In the case of neural networks, the cost function $C$ is non-convex. However, in a specific large width limit regime, some recent results proved that gradient descent is able to converge to a global minimum of the cost function $C$ (Du et al., 2019b,a).

## 2.4. Geometry of function spaces

Throughout the process of optimizing the empirical risk, the steps taken by the algorithm follow a path in the parameter space given by the successive values $\mathbf{w}^{(0)}$,$\mathbf{w}^{(1)}$,$\dots$ $\mathbf{w}^{(T)}$. Each of these parameter values correspond to a function $f_{\mathbf{w}^{(0)}}$,$f_{\mathbf{w}^{(1)}}$,$\dots$,$f_{\mathbf{w}^{(T)}}$ in function space. In the following chapters, we will be interested in characterizing the distances between these functions, as measured using a meaningful notion of distance. We first give an intuition of what is meant by *meaningful* by illustrating this point in the following example.

### 2.4.1. Illustrative example: 2 normal distributions

**Example 4.** Let us consider the family of probability density functions of 1-d normal distributions parametrized by a mean $\mu \in \mathbb{R}$ and a variance $\sigma^2 \in \mathbb{R}^+$:

$$f_{\mu,\sigma^2}(x) = \mathcal{N}\left(x|\mu,\sigma^2\right) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \tag{2.4.1}$$

We consider 4 elements of this family $f^A = f_{0,1}$,$f^B = f_{0,2}$,$f^C = f_{-\frac{1}{2},1-\sqrt{\frac{3}{4}}}$,$f^D = f_{\frac{1}{2},1-\sqrt{\frac{3}{4}}}$ (figure 1). Let us first observe that in the parameter space, we can use the canonical euclidean metric and measure distances between these functions:

$$d_{\text{parameter}}\left(f^A,f^B\right) = 1 \text{ and } d_{\text{parameter}}\left(f^C,f^D\right) = 1 \tag{2.4.2}$$

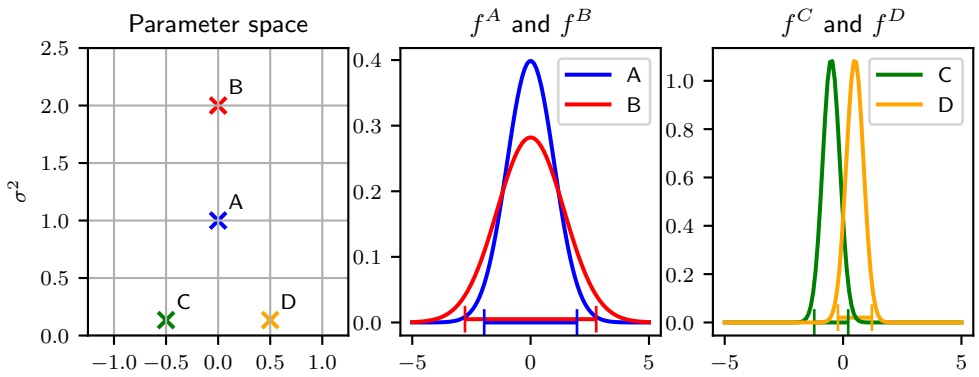



**Figure 1.** in the **(middle)** and **(right)** figures we plot the probability density functions of 4 normal distributions and corresponding 95% quantile. In the **(left)** figure the same functions are represented in parameter space. However equally distant in the parameter space, we can observe that distributions in the left plot are very similar whereas the distributions in the middle plot are very different.

But if we instead look at their graphs (figure 1 middle and right), we see that the probability distributions that they represent are very different: points sampled from $f^A$ and $f^B$ will likely be concentrated around 0 and their respective 95% quantile will largely overlap, but points sampled from $f^C$ and $f^D$ will easily be distinguished since their 95% quantile do not overlap much.

From this example we see that distances in parameter space are not meaningful for measuring distances between probability distributions. We instead resort to distances measured on the function space in the next section.

### 2.4.2. Riemannian metrics on parametric function manifolds

$L^2(p)$ is the space of square integrable functions for the measure $p$:

$$\forall f \in L^2(p), \int_x f(x)^2 p(x)\, dx < \infty \tag{2.4.3}$$

In our case, functions are obtained using an immersion $\mathbf{w} \mapsto f_{\mathbf{w}}$ from $\mathbb{R}^p$ to $L^2(p)$, so that $\mathcal{F}$ inherits a (sub-)manifold structure. The tangent space $\mathcal{T}_{\mathbf{w}}\mathcal{F}$ at the point $f_{\mathbf{w}}$ is the subspace of $L^2(p)$ spanned by the $p$ gradients (see figure 2).

$$\mathcal{T}_{\mathbf{w}}\mathcal{F} = \text{span}\left\{\frac{\partial f_{\mathbf{w}}}{\partial \mathbf{w}_i}, i \in [\![1,p]\!]\right\} \tag{2.4.4}$$

Using the tools of differential geometry, we can define a (Riemannian) metric for infinitesimal elements in $\mathcal{F}$. In particular for a small displacement $\delta\mathbf{w}$ in parameter space, we write $\delta f_{\mathbf{w}} = f_{\mathbf{w}+\delta\mathbf{w}} - f_{\mathbf{w}} = \frac{\partial f_{\mathbf{w}}}{\partial \mathbf{w}}\delta\mathbf{w} + o(\|\delta\mathbf{w}\|)$. Up to first order in $\delta\mathbf{w}$, $\delta f_{\mathbf{w}}$ will live in the tangent space $\mathcal{T}_{\mathbf{w}}\mathcal{F}$. This tangent space helps us define a local metric.

**2−norm in $L^2(p)$.** For general functions in $L^2(p)$, we can use the 2−norm

$$\|\delta f_{\mathbf{w}}\|_{L^2}^2 = \int_x \left(f_{\mathbf{w}+\delta\mathbf{w}}(x) - f_{\mathbf{w}}(x)\right)^2 p(x)\, dx \tag{2.4.5}$$

and define its corresponding metric in parameter space for infinitesimal steps:

$$\|\delta\mathbf{w}\|_{G_{\mathbf{w}}}^2 = \langle \delta\mathbf{w}, G_{\mathbf{w}}\delta\mathbf{w}\rangle \tag{2.4.6}$$

where $G_{\mathbf{w}}$ is the second moment of the partial derivatives of $f_{\mathbf{w}}$ with respect to its parameters:

$$G_{\mathbf{w}} = \mathbb{E}_{x\sim p(x)}\left[\frac{\partial f_{\mathbf{w}}(x)}{\partial \mathbf{w}}\left(\frac{\partial f_{\mathbf{w}}(x)}{\partial \mathbf{w}}\right)^{\top}\right] \tag{2.4.7}$$

For small increments $\delta\mathbf{w}$, we have the following equality:

$$\|\delta f_{\mathbf{w}}\|_{L^2}^2 = \|\delta\mathbf{w}\|_{G_{\mathbf{w}}}^2 + o\left(\|\delta\mathbf{w}\|_2^2\right) \tag{2.4.8}$$

Since we do not have access to the true distribution over $x$, we instead estimate $G_{\mathbf{w}}$ using examples from the training dataset:

$$\tilde{G}_{\mathbf{w}} = \frac{1}{n}\sum_{i=1}^{n} \frac{\partial f_{\mathbf{w}}(x_i)}{\partial \mathbf{w}}\left(\frac{\partial f_{\mathbf{w}}(x_i)}{\partial \mathbf{w}}\right)^{\top} \tag{2.4.9}$$

$\tilde{G}_{\mathbf{w}}$ is sometimes called the *empirical Fisher* since it resembles the Fisher Information Matrix that we describe below but Kunstner et al. (2019) and Thomas et al. (2019) make the argument that it is a misnomer so we will just call it the empirical $L^2$ metric.

**Probability distributions and their density functions.** If we identify our functions $f_{\mathbf{w}}(z) = p(z|\mathbf{w})$ as being probability density functions of distributions on the random variable $z$ then the canonical *natural* metric (Amari, 1985) is:

$$\|\delta f_{\mathbf{w}}\|_{F_{\mathbf{w}}}^2 = \langle \delta\mathbf{w}, F_{\mathbf{w}}\delta\mathbf{w}\rangle \tag{2.4.10}$$

where $F_{\mathbf{w}}$ is the Fisher Information Matrix (FIM).

$$F_{\mathbf{w}} = \mathbb{E}_{z\sim f_{\mathbf{w}}(z)}\left[\frac{\partial \log f_{\mathbf{w}}(z)}{\partial \mathbf{w}}\left(\frac{\partial \log f_{\mathbf{w}}(z)}{\partial \mathbf{w}}\right)^{\top}\right] \tag{2.4.11}$$

This metric is the second order Taylor series expansion of the KL divergence $\mathrm{KL}\left(f_{\mathbf{w}+\delta\mathbf{w}}(z)\,\|\,f_{\mathbf{w}}(z)\right)$ for quantifying how much does the probability distribution with density function $f_{\mathbf{w}+\delta\mathbf{w}}(z)$ differ from the probability distribution with density function $f_{\mathbf{w}}(z)$:

$$\mathrm{KL}\left(f_{\mathbf{w}+\delta\mathbf{w}}(z)\,\|\,f_{\mathbf{w}}(z)\right) = \frac{1}{2}\langle \delta\mathbf{w}, F_{\mathbf{w}}\delta\mathbf{w}\rangle + o\left(\|\delta\mathbf{w}\|^2\right) \tag{2.4.12}$$

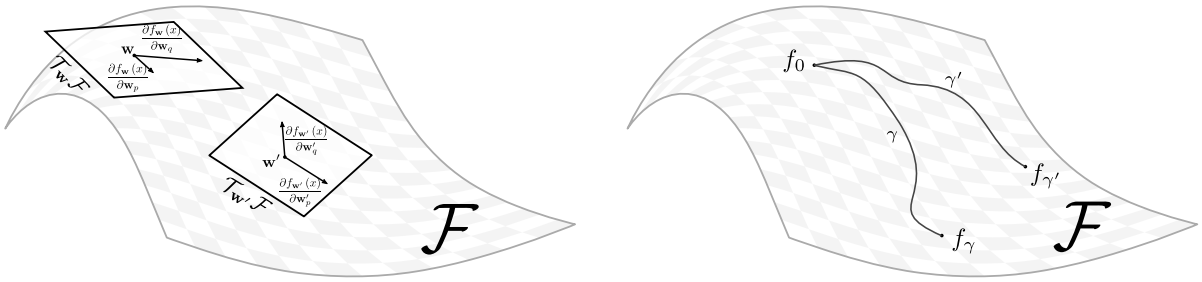



**Figure 2.** **Left**: Tangent spaces. **Right**: Different learning paths in the function manifold

**Empirical estimation for supervised learning.** In supervised learning, we instead identify our functions $f_{\mathbf{w}}(x)$ as conditional probability densities $p(y|x)$[2]. In this case, the formalism above also applies with $p(z) = p(y|x,\mathbf{w})\,p(x)$. The distribution over $x$ is estimated using its empirical estimator:

$$\tilde{F}_{\mathbf{w}} = \frac{1}{n}\sum_{i=1}^{n} \mathbb{E}_{y\sim p(y|x,\mathbf{w})}\left[\frac{\partial \log f_{\mathbf{w}}(x)}{\partial \mathbf{w}}\left(\frac{\partial \log f_{\mathbf{w}}(x)}{\partial \mathbf{w}}\right)^{\top}\right] \tag{2.4.13}$$

This expression further simplifies for classification and regression tasks (Pascanu & Bengio, 2013)

## 2.5. Rademacher complexity: a measure of the capacity of function classes

By characterizing the complexity of a function class, we obtain guarantees on the generalization properties of functions chosen from this function class. Other measures of complexity can also be used, but we only focus on Rademacher complexity in this document. A more thorough discussion can be found in Boucheron et al. (2004).

The Rademacher complexity measures the richness of a hypothesis class, as the expected value of the maximum correlation between a random vector variable $\sigma$ sampled uniformly from $\{-1,1\}^n$ and a function in $\mathcal{F}$, evaluated at $n$ datapoints drawn from $p(x)$:

**Definition 1** (Rademacher complexity)**.** Given a random dataset $\mathcal{D} = \{x_1,\ldots,x_n\}$ of $n$ elements sampled from $p(x)$, and a uniformly sampled random variable $\sigma \in \{-1,1\}^n$, we define the Rademacher complexity of $\mathcal{F}$ as:

$$\mathcal{R}_n(\mathcal{F},p) = \mathbb{E}_{\mathcal{D},\sigma}\left[\frac{1}{n}\sup_{f\in\mathcal{H}}\left|\sum_{i=1}^{n}\sigma_i f(x_i)\right|\right] \tag{2.5.1}$$

[2]For regression, $p(y|x,\mathbf{w}) = \mathcal{N}(y|f_{\mathbf{w}}(x),1)$ and for classification $p(y=c|x,\mathbf{w}) = (f_{\mathbf{w}}(x))_c$ for all classes $c$. For a thorough discussion, see Bishop (2006), chapter 3.1.1 for regression and 4.2 for classification.

Using the Rademacher complexity, we get an upper bound on the generalization gap for any function in $\mathcal{F}$:

**Proposition 2.5.1** (Rademacher complexity based generalization bound)**.** *With probability at least $1-\delta$ over the choice of the random dataset $\mathcal{D}$ of size $n$, every function $f \in \mathcal{F}$ satisfies:*

$$\mathcal{L}(f) - L(f) \leq 2\mathcal{R}_n(\mathcal{F}) + \sqrt{\frac{\log(1/\delta)}{2n}} \tag{2.5.2}$$

Computing the Rademacher complexity for a given machine learning would require knowing the true distribution $p(x)$. We only have access to samples from $p(x)$, so we use an estimator of the Rademacher complexity called the empirical Rademacher complexity:

**Definition 2** (Empirical Rademacher complexity)**.** For a given datasets $\mathcal{D}$ of $n$ elements sampled from $p(x)$, and uniformly sampled random variable $\sigma \in \{-1,1\}^n$, we define the empirical Rademacher complexity of $\mathcal{H}$ as:

$$\hat{\mathcal{R}}_n(\mathcal{F},\mathcal{D}) = \mathbb{E}_\sigma\left[\frac{1}{n}\sup_{f\in\mathcal{H}}\left|\sum_{i=1}^{n}\sigma_i f(x_i)\right| \middle| \mathcal{D}\right] \tag{2.5.3}$$

In the following we will consider the dependency on the dataset $\mathcal{D}$ implicit and simplify the notation to $\hat{\mathcal{R}}_n(\mathcal{F}) = \hat{\mathcal{R}}_n(\mathcal{F},\mathcal{D})$. This estimator also gives us a bound on the generalization gap, very similar to the one obtained with the Rademacher complexity, but slightly less tight.

**Proposition 2.5.2** (Empirical Rademacher based generalization bound)**.** *With probability at least $1-\delta$ over the choice of the random dataset $\mathcal{D}$ of size $n$, every function $f \in \mathcal{F}$ satisfies:*

$$\mathcal{L}(f) - L(f) \leq 2\hat{\mathcal{R}}_n(\mathcal{F}) + \sqrt{\frac{\log(2/\delta)}{n}} \tag{2.5.4}$$

This concludes our general background chapter. We now turn to our contributed work.

# Chapter 3

# NNGeometry: Easy and Fast Fisher Information Matrices and Neural Tangent Kernels in PyTorch

## Prologue

In this chapter, we focus on the practicalities in linearizing the training dynamics of neural networks with respect to displacements in parameter space. The main challenge is the dimensionality of this vector space, which can be up to $10^7$ even on modest architectures. We give examples of techniques using deep networks that rely on some form of linearization, sometimes without explicitly acknowledging it, with the common challenge that it involves computing jacobian matrices whose size can be overwhelming. Practical solutions have been proposed in previous literature, which are here unified in a common Application Programming Interface (API) in Python. In addition, we contribute new algorithmic tricks that make it possible to perform operations on very large matrices implicitly.

***Article Details.*** *NNGeometry: Easy and Fast Fisher Information Matrices and Neural Tangent Kernels in PyTorch.* Thomas George (George, 2021) I gave a presentation of NNGeometry during the PyTorch Ecosystem Day event in 2021. The library is available at `https://github.com/tfjgeorge/nngeometry/`, the documentation can be found at `https://nngeometry.readthedocs.io/en/latest/`.

**Context.** After working with Fisher Information Matrices (FIM) for the EKFAC project (George et al., 2018), I turned to computing finite-width Neural Tangent Kernels (NTK) during training in the NTK alignment project (Baratin et al., 2021). Both are very close sibling objects in that they share the same non-zero spectrum, even if they are being used by different research communities. I felt that a simple and common API for working with FIMs and NTKs was missing, as I was reusing code parts between several projects.

In a more general context, the NTK paper of Jacot et al. (2018) popularized the idea that kernelized linear models could help analyze deep neural networks in the deep learning theory community. In particular, subsequent works analyzed generalization in overparameterized linear models, using the argument that in certain specific infinite-width limiting regimes, deep network training dynamics are fully explained by the training dynamics of a linear model using the neural tangent kernel.

On the other hand, FIMs or equivalent are used in several unrelated subfields of deep learning such as continual learning (Kirkpatrick et al., 2017), pruning (Singh & Alistarh, 2020) or Bayesian deep learning (Maddox et al., 2019). We felt that these could benefit for more efficient representations such as KFAC (Martens & Grosse, 2015) or EKFAC (George et al., 2018), but that it would be burdensome to implement a codebase specialized to their use. It felt more natural to split the task into: i) propose a technique that use the FIM and ii) choose the best representation of this FIM that trades off accuracy and efficiency.

**Developments.** NNGeometry seems to be moderately popular as measured by Github indicators: as of this writing, 131 stars, 15 forks and around 10 unique clones weekly. It is difficult to precisely measure its audience and usage, but we are quite confident that it has saved time to many deep learning practitioners.

Since the release of NNGeometry, the PyTorch project has started releasing its functorch core library that makes implementation of NNGeometry features easier or in some cases redundant. Concurrently to NNGeometry that is built on top of PyTorch, the developers of the Neural Tangents library (Novak et al., 2019) backed by Google and built on top of Jax independently started implementing similar functionalities for computing implicit operations with Jacobians and finite-width NTKs (Novak et al., 2022). While it can be discouraging to have a Google project as a "competitor", it is however a good sign that NNGeometry's features are needed by other projects and thus a sensible research avenue.

**Personal Contribution.** This is a single person project, in which I have conducted design, implementation, documentation, algorithmic improvements, experiments and paper writing. A small part of the code is borrowed from other open-source projects, and the API has matured from numerous discussions with other users of the library, either from Mila or through the Github issue tracker.

## 3.1. Introduction

Practical and theoretical advances in deep learning have been accelerated by the development of an ecosystem of libraries allowing practitioners to focus on developing new techniques instead of spending weeks or months re-implementing the wheel. In particular, automatic differentiation frameworks such as Theano (Bergstra et al., 2011), Tensorflow (Abadi et al., 2016) or PyTorch (Paszke et al., 2019b) have been the backbone for the leap in performance

of last decade's increasingly deeper neural networks as they allow to compute average gradients efficiently, used in the stochastic gradient algorithm or variants thereof. While being versatile in neural networks that can be designed by varying the type and number of their layers, they are however specialized to the very task of computing these average gradients, so more advanced techniques can be burdensome to implement.

Since the growing popularity of neural networks thanks to their always improving performance, other techniques have emerged. Amongst them, we highlight some involving Fisher Information Matrices (FIM) and Neural Tangent Kernels (NTK). Approximate 2nd order (Schraudolph, 2001) or natural gradient techniques (Amari, 1998) aim at accelerating training, elastic weight consolidation (Kirkpatrick et al., 2017) proposes to fight catastrophic forgetting in continual learning and WoodFisher (Singh & Alistarh, 2020) tackles the problem of network pruning so as to minimize its computational footprint while retaining prediction capability. These 3 methods all use the Fisher Information Matrix, but resort to using different approximations when turning to implementation. In parallel, following the work of (Jacot et al., 2018), a line of work study the NTK in either its limiting infinite-width regime, or during training of actual finite-size networks.

All of these papers follow a start by formalizing the problem in a concise math formula, then face the experimental challenge that computing the FIM or NTK involves performing operations for which off-the-shelf automatic differentiation libraries are not well adapted. An even greater turnoff comes from the fact that these matrices scale with the number of parameters (for the FIM) or the number of examples in the training set (for the empirical NTK). This is prohibitively large for modern neural networks involving millions of parameters or large datasets, a problem circumvented by a series of techniques to approximate the FIM (Ollivier, 2015; Martens & Grosse, 2015; George et al., 2018). NNGeometry aims at making use of these approximations effortless, in order to accelerate development or analysis of new techniques, allowing to spend more time on the theory and less time in fighting implementation bugs. NNGeometry's interface is designed to be as close as possible to maths formulas. In summary, this paper and library contribute:

- We introduce NNGeometry by describing and motivating design choices.
  - A unified interface for all FIM and NTK operations, regardless of how these are approximated.
  - Implicit operations for ability to scale to large networks.
- Using NNGeometry, we get new empirical insights on FIMs and NTKs:
  - We compare different approximations in different scenarios.
  - We scale some NTK experiments to TinyImagenet.

**using a KFAC Fisher**

```
F_kfac = FIM(model=model,
             loader=loader,
             representation=PMatKFAC,
             n_output=10)

v = PVector.from_model(model)

vTMv = F_kfac.vTMv(v)
```

**using implicit computation**

```
F_full = FIM(model=model,
             loader=loader,
             representation=PMatDense,
             n_output=10)

v = PVector.from_model(model)

vTMv = F_full.vTMv(v)
```

**Figure 1.** Computing a vector-Fisher-vector product $\mathbf{v}^\top F\mathbf{v}$, for a 10-fold classification model defined by `model`, can be implemented with the same piece of code for 2 representations of the FIM using NNGeometry, even if they involve very different computations under the hood.

## 3.2. Preliminaries

### 3.2.1. Network linearization

Neural networks are parametric functions $f(x,\mathbf{w}) : \mathcal{X} \times \mathbb{R}^d \to \mathbb{R}^c$ where $x \in \mathcal{X}$ are covariates from an input space, and $\mathbf{w} \in \mathbb{R}^d$ are the network's parameters, arranged in layers composed of weight matrices and biases. The function returns a value in $\mathbb{R}^c$, such as the $c$ scores in softmax classification, or $c$ real values in $c$-dimensional regression. Neural networks are trained by iteratively adjusting their parameters $\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} + \delta\mathbf{w}^{(t)}$ using steps $\delta\mathbf{w}^{(t)}$ typically computed using the stochastic gradient algorithm or variants thereof, in order to minimize the empirical risk of a loss function.

In machine learning, understanding and being able to control the properties of the solution obtained by an algorithm is of crucial interest, as it can provide generalization guarantees, or help design more efficient or accurate algorithms. Contrary to (kernelized) linear models, where closed-form expressions of the empirical risk minimizer exist, deep networks are non-linear functions, whose generalization properties and learning dynamics is not yet fully understood. Amongst the recent advances toward improving theory, is the study of the linearization (in $\mathbf{w}$) of the deep network function $f(x,\mathbf{w})$:

$$f(x,\mathbf{w}+\delta\mathbf{w}) = f(x,\mathbf{w}) + J(x,\mathbf{w})\,\delta\mathbf{w} + o(\|\delta\mathbf{w}\|) \tag{3.2.1}$$

where $J(x,\mathbf{w}) = \frac{\partial f(x,\mathbf{w})}{\partial \mathbf{w}}$ is the Jacobian with respect to parameters $\mathbf{w}$, computed in $(\mathbf{w}, x)$, mapping changes in parameter space $\delta\mathbf{w}$ to corresponding changes in output space using the identity $\delta f(x,\mathbf{w},\delta\mathbf{w}) = J(x,\mathbf{w})\,\delta\mathbf{w}$. The Landau notation $o$ (pronounced "little-o") means a function whose exact value is irrelevant, with the property that $\lim_{x\to 0} \frac{o(x)}{x} = 0$, or in other words that is negligible compared to $x$ for small $x$. For tiny steps $\delta\mathbf{w}$, we neglect the term $o(\|\delta\mathbf{w}\|)$ thus $f$ is close to its linearization. It happens for instance at small step sizes, or for networks trained in the large-width limit with the specific parameter initialization scheme studied in Jacot et al. (2018).

### 3.2.2. Parameter space metrics and Fisher Information Matrix

While neural networks are trained by tuning their parameters $\mathbf{w}$, the end goal of machine learning is not to find the best parameter values, but rather to find *good* functions, in a sense that depends on the task at hand. For instance, different parameter values can represent the same function (Dinh et al., 2017). On the contrary 2 parameter space steps $\delta\mathbf{w}_1$ and $\delta\mathbf{w}_2$ with same euclidean norm can provide very different changes in a function ($\delta f\left(x,\mathbf{w},\delta\mathbf{w}_1\right) \neq \delta f\left(x,\mathbf{w},\delta\mathbf{w}_2\right)$). In order to quantify changes of a function, one generally defines a distance[1] on the function space. Examples of such distances are the $L_k$-norms, Wasserstein distances, or the KL divergence used in information geometry.

To each of these function space distances correspond a parameter space metric. We continue our exposition by focusing on the KL divergence, which is closely related to the Fisher Information Matrix, but our library can be used for other function space distances.

Suppose $f$ is interpreted as the log-probability of a density $p$: $\log p\left(x,\mathbf{w}\right) = f\left(x,\mathbf{w}\right)$, the KL divergence gives a sense of how much the probability distribution changes when adding a small increment $\delta\mathbf{w}$ to the parameters of $f\left(x,\mathbf{w}\right)$. We can approximate it as:

$$\mathrm{KL}\left(p\left(x,\mathbf{w}\right)\|p\left(x,\mathbf{w}+\delta\mathbf{w}\right)\right) = \int_{x\in\mathcal{X}} \log\left(\frac{p\left(x,\mathbf{w}\right)}{p\left(x,\mathbf{w}+\delta\mathbf{w}\right)}\right) dp\left(x,\mathbf{w}\right) \tag{3.2.2}$$

$$= \frac{1}{2}\int_{x\in\mathcal{X}} \left(\frac{1}{p\left(x,\mathbf{w}\right)} J\left(x,\mathbf{w}\right)\delta\mathbf{w}\right)^2 dp\left(x,\mathbf{w}\right) + o\left(\|\delta\mathbf{w}\|^2\right) \tag{3.2.3}$$

where we used this form in order to emphasize how steps in parameter space $\delta\mathbf{w}$ affect distances measured on the function space: equation 3.2.3 is the result of (-) taking a step $\delta\mathbf{w}$ in parameter space; (-) multiplying with $J\left(x,\mathbf{w}\right)$ to push the change to the function space; (-) weight this function space change using $p\left(x,\mathbf{w}\right)^{-1}$; (-) square and sum. In particular, because of the properties of the KL divergence, there is no second derivative of $f$, even if equation 3.2.3 is equivalent to taking the 2nd order Taylor series expansion of the KL divergence. We write in a more concise way:

$$\mathrm{KL}\left(f\left(x,\mathbf{w}\right)\|f\left(x,\mathbf{w}+\delta\mathbf{w}\right)\right) = \delta\mathbf{w}^\top F_{\mathbf{w}}\delta\mathbf{w} + o\left(\|\delta\mathbf{w}\|^2\right) \tag{3.2.4}$$

which uses the $d\times d$ FIM $F_{\mathbf{w}} = \int_{x\in\mathcal{X}} \frac{1}{p(x,\mathbf{w})^2} J\left(x,\mathbf{w}\right)^\top J\left(x,\mathbf{w}\right) dp\left(x,\mathbf{w}\right)$. We can now define the norm $\|\delta\mathbf{w}\|_{F_{\mathbf{w}}} = \delta\mathbf{w}^\top F_{\mathbf{w}}\delta\mathbf{w}$ used in the natural gradient algorithm (Amari (1998), also see Martens (2020) for a more thorough discussion of the FIM), in elastic weight consolidation (Kirkpatrick et al., 2017), or in pruning (Singh & Alistarh, 2020). Other quantities also share the same structure of a covariance of parameter space vectors, such as the covariance of loss gradients in TONGA (Le Roux et al., 2008), the second moment of loss gradients[2] (Kunstner

[1] We here use the notion of distance informally.

[2] The second moment of loss gradients is sometimes called *empirical Fisher*.

et al., 2019; Thomas et al., 2020), or posterior covariances in bayesian deep learning (e.g. in Maddox et al. (2019)). In more generality, any metric defined by a positive define matrix on the parameter space can be used with NNGeometry. These metrics define their corresponding geometry – hence the name NN (for neural networks) geometry.

### 3.2.3. Neural Tangent Kernel

Another very active line of research around the linearization of equation 3.2.1 is to take inspiration from the rich literature on kernel methods by defining the neural tangent kernel (NTK):

$$k_{\mathbf{w}}(x,y) = J(x,\mathbf{w})\, J(y,\mathbf{w})^{\top} \tag{3.2.5}$$

In the limit of networks with infinite width, Jacot et al. (2018) have shown that the tangent kernel remains constant through training using gradient descent when the parameters are initialized using a specific scheme, which allows to directly import learning theory ideas from linear models to deep learning. While this regime is of theoretical interest, it arguably does not explain what happens at finite width, where the NTK evolves during training.

Kernels are functions of the whole input space $\mathcal{X} \times \mathcal{X}$, but we often only have access to a limited number of samples in a datasets. It is often convenient to evaluate the kernel at points $x_i$ of a training or a test set, called the Gram Matrix $(K_{\mathbf{w}})_{ij} = k_{\mathbf{w}}(x_i,x_j)$. Note that in the case where the output space is multidimensional with dimension $c$, then $K_{\mathbf{w}}$ is in fact a 4d tensor.

## 3.3. Design and implementation

### 3.3.1. Challenges

Current deep learning frameworks such as PyTorch and Tensorflow are well adapted to neural network training, which generally requires computing average gradients over parameters, used in optimizers such as Adam and others. However, when going to more advanced algorithms or analysis techniques involving FIMs and NTKs, practitioners typically have to hack the framework's internal mechanisms, which is time-consuming, error-prone, and results in each project having their own slightly different implementation of the very same technique. We here list the difficulties in computing FIMs and NTKs using current frameworks:

**Per-example gradients.** FIMs and NTKs require per-example Jacobians $J(x_i,\mathbf{w})$ of a dataset $(x_i)_i$. This can be obtained by looping through examples $x$, but at the cost of not using mini-batched operations, thus missing the benefit of using GPUs. Instead, NNGeometry's Jacobian generator extensively use efficient techniques (Goodfellow, 2015; Rochette et al., 2019).

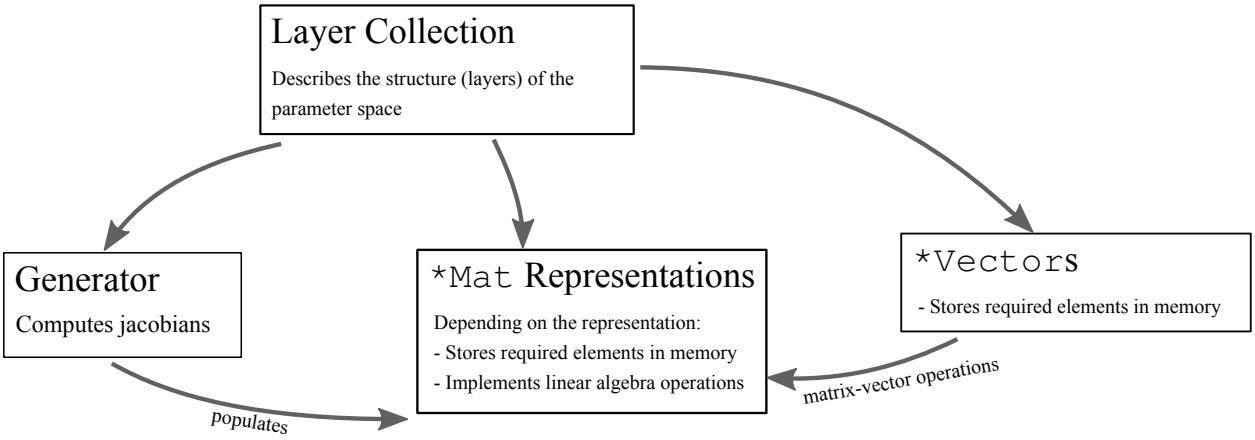



**Figure 2.** Schematic description of NNGeometry's main components

**Memory usage and computational cost.** A FIM matrix is $d \times d$ where $d$ is the total number of parameters. With a memory cost in $\mathcal{O}(d^2)$, this is prohibitively costly even for moderate size networks. Typical linear algebra operations have a computational cost in either $\mathcal{O}(d^2)$ (e.g. matrix-vector product) or even $\mathcal{O}(d^3)$ (e.g. matrix inverse). NNGeometry instead comes with recent, lower memory intensive approximations (Martens & Grosse, 2015; Ollivier, 2015; George et al., 2018).

## 3.3.2. NNGeometry's design

The main components of NNGeometry are sketched in figure 2. Abstract mathematical objects (such as the FIM, parameter space vectors, ...) are represented by concrete representations that define how these objects are stored in memory, and how operations are internally performed. Generators populate these representations using a dataset of examples.

**3.3.2.1. Abstract objects.** In section 3.2, we have worked with abstract mathematical objects $\delta\mathbf{w}$, $\delta f(x, \mathbf{w}, \delta\mathbf{w})$, $J(x,\mathbf{w})$, $F_{\mathbf{w}}$ and $K_{\mathbf{w}}$. These mathematical objects are identified to Python classes in NNGeometry.

**Parameter space.** We start with the parameter space. Previously, we identified the parameter space as $\mathbb{R}^d$, but it is in fact often implemented as a set of weight matrices and bias vectors $\mathbf{w} = \{W_1,b_1,\ldots,W_l,b_l\}$ in deep learning frameworks. Parameter space vectors are represented by the class `PVector` in NNGeometry, which is essentially a dictionary of PyTorch `Parameter`s, with basic algebra logic: `PVector`s can be readily added, subtracted, and scaled by a scalar with standard python operators. As an illustration `wsum = w1 + w2` internally loops through all parameter tensors of `w1` and `w2` and returns a new `PVector` `w_sum`.

Similarly, and more interestingly, parameter space metrics such as the FIM are represented by classes prefixed with `PMat`. For instance, the natural gradient $\delta_{\text{nat}} = -\eta F^{-1}\nabla_{\mathbf{w}}\mathcal{L}$

applies the linear operator $\mathbf{w} \mapsto F^{-1}\mathbf{w}$ to the parameter space vector $\nabla_{\mathbf{w}}\mathcal{L}$, and can be implemented cleanly and concisely using `delta_nat = - eta * F.solve(nabla_L)`, even if the "solve" operation internally involves different computations for different layer types and choice of approximation.

The structure of the parameter space is internally abstracted using the `LayerCollection` class. This gives the flexibility of defining our parameter space as parameters of a subset of layers, in order to treat different layers in different ways. An example use case is to manipulate a mix of KFAC for linear layers parameters, and block-diagonal for GroupNorm layers, as KFAC is not defined for the latter.

**Function space.** Function space vectors `FVector` define objects associated to vectors of the output space, evaluated on a dataset of $n$ examples $X$. As an example, getting back to the linearization $\delta f\left(x,\mathbf{w},\delta\mathbf{w}\right) = J\left(x,\mathbf{w}\right)\delta\mathbf{w}$, we define $\delta\mathbf{f}\left(X\right) = \left(\delta f\left(x_1,\mathbf{w},\delta\mathbf{w}\right),\ldots,\delta f\left(x_n,\mathbf{w},\delta\mathbf{w}\right)\right)$ as the $\mathbb{R}^{c\times n}$ function space vector of output changes evaluated at examples $X$. Gram matrices of the NTK are linear operators on this space, represented by objects prefixed with `FMat`.

**Mixed objects.** Borrowing from the vocabulary of differential geometry, we also define `PushForward` objects that are linear operators from parameter space to function space, and `PullBack` objects that are linear operators from function space to parameter space.

**3.3.2.2. Concrete representations.** These abstract objects are implemented in memory using concrete representations. NNGeometry comes with a number of representations. Amongst them, most notably, are parameter space approximations proposed in recent literature (Ollivier, 2015; Martens & Grosse, 2015; Grosse & Martens, 2016; George et al., 2018), and an implicit representation for each abstract linear operator, that allows to compute linear algebra operations without ever computing or storing the matrix in memory.

`PMatDense` (resp `FMatDense`) and `PMatDiag` represent the full dense matrix and the diagonal matrix. `PMatLowRank` only computes and stores the $c \times n \times d$ Jacobian $\mathbf{J}\left(X,\mathbf{w}\right)$ for all examples of the given dataset, which is often much smaller than the dense $d \times d$ FIM.

Next come representations that do not consider neural networks as black-box functions, but instead take advantage of the layered structure of the networks: `PMatBlockDiag` uses dense blocks of the FIM for parameters of the same layer, and puts zeros elsewhere, ignoring cross-layer covariance. `PMatQuasiDiag` (Ollivier, 2015) uses the full diagonal and adds to each bias parameter the interaction with the corresponding row of the weight matrix. `PMatKFAC` uses KFAC (Martens & Grosse, 2015) and its extension to convolution layers KFC (Grosse & Martens, 2016) to approximate each layer blocks with the kronecker product of 2 much smaller matrices, thus saving memory and compute compared to `PMatBlockDiag`. `PMatEKFAC` uses the EKFAC (George et al., 2018) extension of KFAC.

The last representation that comes with this first release of NNGeometry, `PMatImplicit`, allows computing certain linear algebra operations using the full dense matrix implicitly, without the need to ever store it in memory, which permits scaling to large networks (see experiments in section 3.4). As an illustration, the vector-matrix-vector product $\mathbf{v}^\top F \mathbf{v}$ can be computed using equation 3.2.3, without ever needing to compute and store the $d \times d$ FIM.

Each representation comes with its advantages and drawbacks, allowing to trade off between memory and approximation accuracy. For a new project, we recommend starting with a small network using the `PMatDense` representation, then gradually switching to representations with a lower memory footprint while experimenting with actual modern networks.

While linear algebra operations associated to each representation internally involve very different mechanisms, NNGeometry's core contribution is to give easy access to these operations by using the same simple methods as illustrated in the code snippet 1.

**3.3.2.3. Generators.** Now that we have defined concrete classes, we need a mechanism to populate them using examples $x$ coming from a dataset. This is the role of NNGeometry's generators. It computes actual entries of the matrices, depending on the representation. A naive implementation would be to loop through examples $\{x_i\}_i$, compute $f(x_i, \mathbf{w})$ and compute gradients with respect to parameters using PyTorch's automatic differentiation. It would be rather inefficient as it would not make usage of parallelism in GPUs. Instead, NNGeometry's generator allows using minibatches of examples by intercepting PyTorch's gradients and using techniques similar to Goodfellow (2015) and Rochette et al. (2019):

Let us consider $f(x,\mathbf{w}) : \mathcal{X} \times \mathbb{R}^d \to \mathbb{R}^c$. In order to simplify our exposition, we focus on fully connected layers and suppose that $f$ can be written $f(x,\mathbf{w}) = \sigma_l \circ g_l(\cdot,\mathbf{w}) \circ \sigma_{l-1} \circ g_{l-1}(\cdot,\mathbf{w}) \circ \ldots \circ \sigma_1 \circ g_1(x,\mathbf{w})$ where $\sigma_k$ are activation functions and $g_k$ are parametric affine transformations that compute pre-activations $s_k$ of a layer using a weight matrix $W_l$ and a bias vector $b_k$ with the following expression: $s_k = g_k(a_{k-1},\mathbf{w}) = W_k a_{k-1} + b_k$. For each example $x_i$ in a minibatch, we denote these intermediate quantities by superscripting $s_k^{(i)}$ and $a_k^{(i)}$. The backpropagation algorithm applied to computing gradients of a sum $S = \sum_i f(x_i,\mathbf{w})$ works by sequentially computing intermediate gradients $\frac{\partial f(x_i,\mathbf{w})}{\partial s_k^{(i)}}$ from top layers to bottom layers. Denote by $D\mathbf{s}_k = \left(\frac{\partial f(x_i,\mathbf{w})}{\partial s_k^{(1)}}^\top, \ldots, \frac{\partial f(x_i,\mathbf{w})}{\partial s_k^{(m)}}^\top\right)^\top$ the matrix obtained by stacking these gradients for a minibatch of size $m$, and $\mathbf{a}_k = \left(a_k^{(1)}, \ldots, a_k^{(m)}\right)$ the corresponding matrix of activations of the same layer. These are already computed when performing the backpropagation algorithm, then used to obtain the average gradient w.r.t the weight matrix by means of the matrix/matrix product $\frac{\partial}{\partial W_l}\left\{\sum_i f(x_i,\mathbf{w})\right\} = D\mathbf{s}_k^\top \mathbf{a}_k$. The observation of Goodfellow (2015) is that we can in addition obtain individual gradients $\frac{\partial f(x_i,\mathbf{w})}{\partial s_k^{(1)}}^\top a_k^{(i)\top}$, an operation that can be parallelized efficiently for all examples of the minibatch using the `bmm` PyTorch function.

Instead of reimplementing backpropagation as is for example done in BackPACK (Dangel et al., 2019), we chose to use PyTorch's internal automatic differentiation mechanism, as it already handles most corner cases encountered by deep learning practitioners: we do not have to reimplement backward computations for every new layer, but instead we just have to compute individual gradients by intercepting gradients with respect to pre-activations $D\mathbf{s}_k$.

Other generators are to be added to NNGeometry in the future, either by using different ways of computing the Jacobians, or by populating representations using other matrices such as the Hessian matrix, or the KFRA approximation of the FIM (Botev et al., 2017).

## 3.4. Experimental showcase

Equipped with NNGeometry, we experiment with a large network: We train a 24M parameters Resnet50 network on TinyImagenet. We emphasize that given the size of the network, we would not have been able to compute operations involving the true $F$ without NNGeometry's `PMatImplicit` representation, since $F$ would require 2.3 petabytes of memory ($24M \times 24M \times 4$ bytes for float32).

### 3.4.1. Quality of FIM approximations

We start by comparing the accuracy of several `PMat` representations at computing various linear algebra operations. We use a Monte-Carlo estimate of the FIM, where we use 5 samples from $p(y|x)$ for each example $x$. Here, since this TinyImagenet is a classification task, $p(y|x)$ is a multinoulli distribution with the event probabilities given by the softmax layer. We compare the approximate value obtained for each representation, to a "true" value, obtained using the full matrix with the `PMatImplicit` representation. For trace and $\mathbf{v}^\top F\mathbf{v}$, we compare these quantities using the relative difference $\left|\frac{\text{approx}-\text{true}}{\text{true}}\right|$. For $F\mathbf{v}$, we report the cos-angle $\frac{1}{\|F\mathbf{v}\|_2\|F_{\text{approx}}\mathbf{v}\|_2}\langle F\mathbf{v}, F_{\text{approx}}\mathbf{v}\rangle$, and for the solve operation, we report the cos-angle between $\mathbf{v}$ and $(F_{\text{approx}} + \lambda I)^{-1}(F + \lambda I)\mathbf{v}$. Since the latter is highly dependent on the Tikhonov regularization parameter $\lambda$, we plot the effect on the cos-angle of varying the value of $\lambda$. The results can be observed in figures 3, 4, 5, 6.

From this experiment, there is no best representation for all linear algebra operations. Instead, this analysis suggest to use `PMatKFAC` when possible for operations involving the inverse FIM, and `PMatEKFAC` for operations involving the (forward) FIM. Other representations are less accurate, but should not be discarded as they can offer other advantages, such as lower memory footprint, and faster operations.

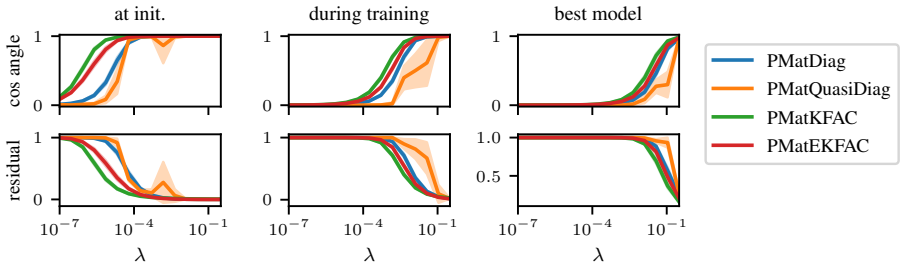



**Figure 3.** Residual $\frac{\|\mathbf{v}-\mathbf{v}'\|_2}{\|\mathbf{v}'\|_2}$ and cos angle between $\mathbf{v}$ and $\mathbf{v}' = (F_{\text{approx}} + \lambda I)^{-1}(F + \lambda I)\mathbf{v}$ for a 24M parameters Resnet50 at different points during training on TinyImagenet, using different approximations $F_{\text{approx}}$ of $F$, for $\mathbf{v}$ uniformly sampled on the unit sphere (higher is better).

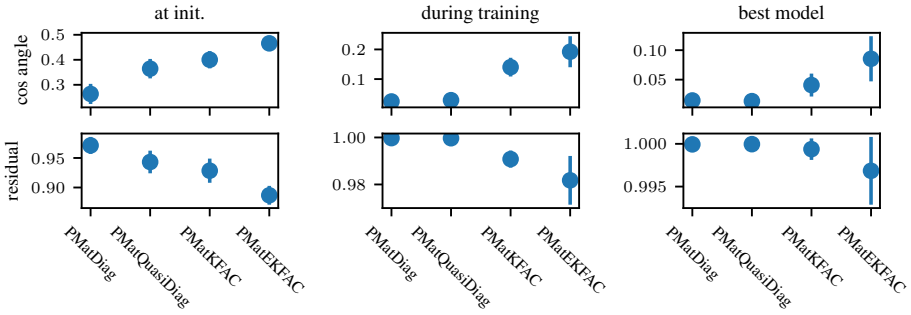



**Figure 4.** Residual and cos angle between $F\mathbf{v}$ and $F_{\text{approx}}\mathbf{v}$ for a 24M parameters Resnet50 at different points during training on TinyImagenet, using different approximations $F_{\text{approx}}$ of $F$, for $\mathbf{v}$ uniformly sampled on the unit sphere (higher is better).

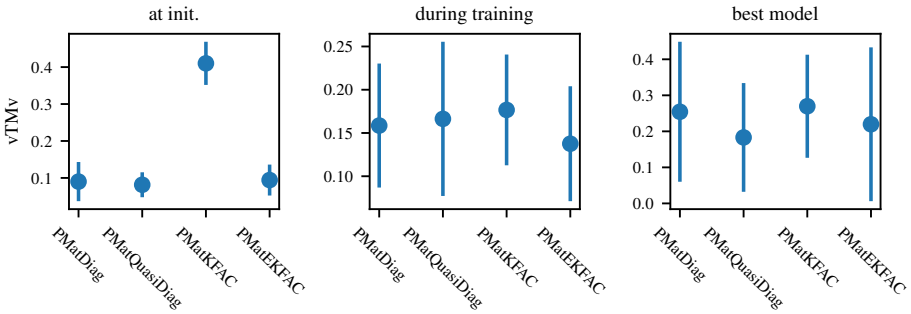



**Figure 5.** Relative difference between $\mathbf{v}^\top F\mathbf{v}$ and $\mathbf{v}^\top F_{\text{approx}}\mathbf{v}$ for a 24M parameters Resnet50 at different points during training on TinyImagenet, using different approximations $F_{\text{approx}}$ of $F$, for $\mathbf{v}$ uniformly sampled on the unit sphere (higher is better).

### 3.4.2. Neural Tangent Kernel eigendecomposition

In the line of recent work that study the finite-width NTK (Baratin et al., 2021; Paccolat et al., 2021), we observe the evolution of the NTK during training. We use a Resnet50 architecture trained on the 200 classes of TinyImagenet, but in order to be able to plot a 2d matrix for analysis, we extract the univariate function $f_{c_1,c_2}(x,\mathbf{w}) = (f(x,\mathbf{w}))_{c_2} - (f(x,\mathbf{w}))_{c_1}$,

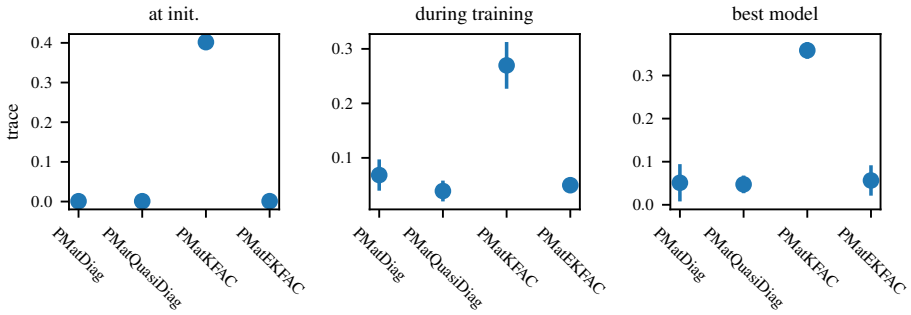



**Figure 6.** Relative difference of trace computed using $F_{\text{approx}}$ and $F$ (lower is better). As we observe, all 3 representations `PMatDiag`, `PMatQuasiDiag` and `PMatEKFAC` estimate the trace very accurately, since the only remaining fluctuation comes from Monte-Carlo sampling of the FIM. On the other hand, the estimation provided by `PMatKFAC` is less accurate.

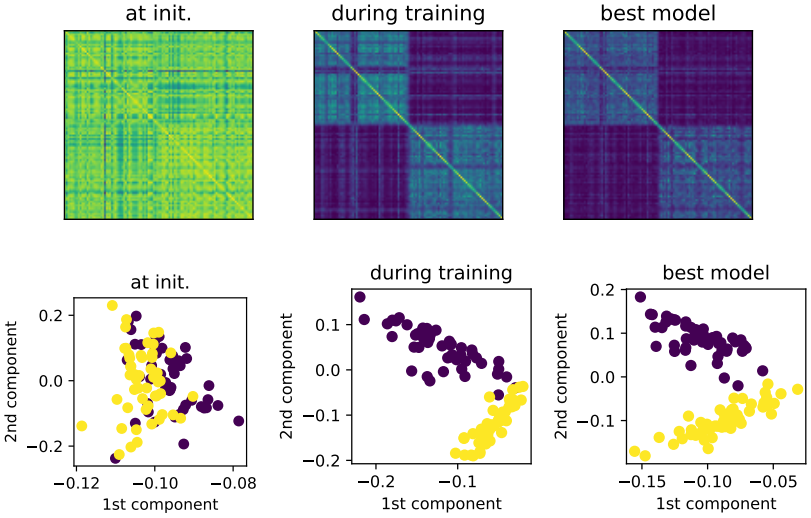



**Figure 7.** NTK analysis for 50 examples of class $c_1$ and 50 examples of class $c_2$ at various points during training. **(top row)** Gram matrix of the NTK. Each row and column is normalized by $\frac{1}{\sqrt{diag\,(G)}}$ for better visualization. We observe that the NTK encodes some information about the task later in training, since it highlights intra-class examples. **(bottom row)** Examples are projected on the 1st 2 principal components of the Gram Matrix at various points during training. While points are merely mixed at initialization, the NTK adapts to the task and becomes a good candidate for kernel PCA since examples become linearly separable as training progresses.

that defines a binary classifier of class $c_2$ vs class $c_1$. We plot at different points during training i) the Gram matrix of examples from the 2 classes $c_1$ and $c_2$ (figure 7, top row) and ii) a kernel PCA of points from classes $c_1$ and $c_2$ projected on the 2 first principal components (figure 7, bottom row). The Gram matrix is computed for valid set examples of classes $c_1$ and $c_2$.

On this larger network, we reproduce the conclusion of Baratin et al. (2021) and Paccolat et al. (2021) that the NTK evolution is not purely random during training, but instead adapts to the task in a very specific way.

## 3.5. Conclusion

We introduced NNGeometry, a PyTorch library that allows to compute various linear algebra operations involving Fisher Information Matrices and Neural Tangent Kernels, using an efficient implementation that is versatile enough given current usages of these matrices, while being easy enough to save time for the user. We also introduced implicit operations, as well as some example usage involving large networks for which we would not otherwise be able to construct FIMs and NTKs as it would be prohibitively costly.

NNGeometry can be readily used in existing projects, and we hope that it will help make progress across deep learning subfields as FIMs and NTKs are used in a range of applications.

Equipped with NNGeometry, we are now able to implement an optimization technique in the next chapter, or analyze the training dynamics in Chapters 5 and 6.

# Chapter 4

# Fast approximate natural gradient in a Kronecker factored Eigenbasis

## Prologue

In this chapter, we present an approximation of the Fisher Information Matrix (FIM) of neural networks, that can be 1/ efficiently computed and stored in memory, even for million-parameters networks and 2/ efficiently handled (in our case we need to compute the inverse of the FIM). We design an approximate natural gradient procedure named EKFAC, that we use to accelerate neural network training, both in number of iterations and in wall-clock time.

***Article Details.*** *Fast approximate natural gradient in a Kronecker factored Eigenbasis.* Thomas George*, César Laurent*, Xavier Bouthillier, Nicolas Ballas, Pascal Vincent. The paper has been published at *NeurIPS 2018* (George et al., 2018). Compared to the paper, we have here added a section introducing more background on the natural gradient algorithm.

* Co-first authors.

**Context.** After the seminal success of AlexNet (Krizhevsky et al., 2017) at beating other machine learning vision systems in the ImageNet classification challenge, a series of architectures have been proposed (GoogleNet, VGG, ResNet) that increased the number of trainable parameters to as many as $10^7$ (this number has since gained 4 orders of magnitude). A natural direction of research was to focus on optimization algorithms capable of handling a very large number of variables, while providing more efficient steps than SGD. Natural gradient (Amari, 1998) was an interesting avenue, and had recently been applied to deep networks, using the KFAC approximate factorization of the FIM. We explored ways to improve over KFAC both in terms of quality of the approximation, and in terms of efficiency. We were also inspired by diagonal optimization methods such as RMSProp, Adam, and the quasi-diagonal method of Ollivier (2015).

This project was also a motivation for starting the development of NNGeometry, since we had to reimplement our competitor KFAC from scratch (no publicly available implementation was available at the time).

**Developments.** Since publication of the paper, EKFAC as an optimization method has not been used much in practice, since practitioners are more used to choosing SGD with momentum or Adam as default, and these often provide reasonable choices that are robust to choice of hyperparameters. Concurrently, GPUs have become much more powerful and cheaper, default neural network architectures (skip connections, ReLU non-linearities, parameter initialization schemes) are much easier to train and normalization schemes (such as batch normalization) are now ubiquitous. Altogether, this renders the entry cost of using natural gradient optimizers too high. The main follow-ups of EKFAC are thus not optimization algorithms, but use the EKFAC parameterization for other purposes.

- Wang et al. (2019) prune a trained network using the KFE, in which candidate directions for pruning are already at hand as the smaller diagonal coefficient of the FIM.
- Bae et al. (2018) perform variational inference using a posterior with a Gaussian distribution whose covariance is captured by projection in the KFE.
- Liu et al. (2018) concurrently explored the idea of rotating the FIM using the SVD of kronecker factors in order to approximate the FIM in the context of continual learning.

We provide an open-source PyTorch implementation of the EKFAC natural gradient optimizer at: `https://github.com/Thrandis/EKFAC-pytorch`, and an EKFAC representation is available in NNGeometry.

**Personal Contribution.** With César, we had been implementing modifications of optimizers inspired by natural neural networks (Desjardins et al., 2015), KFAC (Martens & Grosse, 2015; Grosse & Martens, 2016), and TProp (George, 2018, beginning of chapter 6), and I came up with the original idea of using the EKFAC factorization. I also contributed experiments on MLPs and framing our hyperparameter search method, as well as paper writing and theorem proofs. César implemented and ran innumerable experiments on convolutional networks, and was later supported by Nicolas in scaling experiments to larger networks. Xavier had been working on similar ideas, coming from a different perspective. Pascal helped formalize the method, and contributed theorems and writing.

## 4.1. Introduction

Deep networks have exhibited state-of-the-art performance in many application areas, including image recognition (He et al., 2016a) and natural language processing (Gehring

et al., 2017). However top-performing systems often require days of training time and a large amount of computational power, so there is a need for efficient training methods.

Stochastic Gradient Descent (SGD) and its variants are the current workhorse for training neural networks. Training consists in optimizing the network parameters $\mathbf{w}$ (of size $n_{\mathbf{w}}$) to minimize a cost function $C(\mathbf{w})$, through gradient descent. The negative loss gradient is approximated based on a small subset of training examples (a mini-batch). The loss functions of neural networks are highly non-convex functions of the parameters, and the loss surface is known to have highly imbalanced curvature which limits the efficiency of $1^{\text{st}}$ order optimization methods such as SGD.

Methods that correct for imbalaced curvature have the potential to speed up gradient descent. The parameters are then updated as: $\mathbf{w} \leftarrow \mathbf{w} - \eta G^{-1} \nabla C(\mathbf{w})$, where $\eta$ is a positive learning-rate and $G$ is a preconditioning matrix capturing the local curvature or related information such as the Hessian matrix in Newton's method or the Fisher Information Matrix in Natural Gradient (Amari, 1998, also see section 4.2). Matrix $G$ has a gigantic size $d \times d$ which makes it too large to compute and invert in the context of modern deep neural networks with millions of parameters. For practical applications, it is necessary to trade-off quality of curvature information for efficiency.

A long family of algorithms used for optimizing neural networks can be viewed as approximating the diagonal of a large preconditioning matrix. Diagonal approximations of the Hessian (Becker et al., 1988) have been proven to be efficient, and algorithms that use the diagonal of the covariance matrix of the gradients are widely used among neural networks practitioners, such as Adagrad (Duchi et al., 2011), Adadelta (Zeiler, 2012), RMSProp (Tieleman & Hinton, 2012), Adam (Kingma & Ba, 2015). We refer the reader to Bottou et al. (2016) for an informative review of optimization methods for deep networks, including diagonal rescalings, and connections with the Batch Normalization (BN) technique (Ioffe & Szegedy, 2015).

More elaborate algorithms do not restrict to diagonal approximations, but instead aim at accounting for some correlations between different parameters (as encoded by non-diagonal elements of the preconditioning matrix). These methods range from Ollivier (2015) who introduces a rank 1 update that accounts for the cross correlations between the biases and the weight matrices, to quasi Newton methods (Liu & Nocedal, 1989) that build a running estimate of the exact non-diagonal preconditioning matrix, and also include block diagonals approaches with blocks corresponding to entire layers (Heskes, 2000). Factored approximations such as KFAC (Martens & Grosse, 2015) approximate each block as a Kronecker product of two much smaller matrices, both of which can be estimated and inverted more efficiently than the full block matrix, since the inverse of a Kronecker product of two matrices is the Kronecker product of their inverses.

In the present work, we draw inspiration from both diagonal and factored approximations. We introduce an Eigenvalue-corrected Kronecker Factorization (EKFAC) that consists in tracking a diagonal variance, not in parameter coordinates, but in a Kronecker-factored eigenbasis. We show that EKFAC is a provably better approximation of the Fisher Information Matrix than KFAC. In addition, while computing the Kronecker-factored eigenbasis is a computationally expensive operation that needs to be amortized, tracking of the diagonal variance is a cheap operation. EKFAC therefore allows to perform partial updates of our curvature estimate $G$ at the iteration level. We conduct an empirical evaluation of EKFAC on the deep auto-encoder optimization task using fully-connected networks and CIFAR-10 relying on deep convolutional neural networks where EKFAC shows improvements over KFAC in optimization.

**Organisation of the chapter.**

- in section 4.2, we present the natural gradient algorithm and show that it can be used for optimization but at the cost of inverting a large matrix. We show some early proposed approximations ;
- in section 4.3, we introduce our new method named EKFAC ;
- in section 4.4, we benchmark EKFAC against other popular algorithms for optimizing deep networs.

## 4.2. Natural gradient

We start by discussing the natural gradient algorithm for learning probability distributions, introduced in Amari (1998). We then motivate its use in the context of neural networks and show that computing it is too costly too compute it exactly, encouraging the design of more efficient approximate methods.

### 4.2.1. Steepest descent in the natural metric

**Steepest descent for the euclidean metric.** In the deep learning community, the gradient descent algorithm is often motivated as following the *steepest descent* direction $\delta\mathbf{w}$, where *steepness* refers to the improvement of the cost function $C(\mathbf{w}+\delta\mathbf{w})$ with respect to a given budget $c$ of displacement measured in parameter space using the canonical euclidean metric $\|\delta\mathbf{w}\|_2$. Each iterate of gradient descent incurs an update of the parameter values $\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} + \delta\mathbf{w}_{\text{GD}}$, where $\delta\mathbf{w}_{\text{GD}} = -\eta\nabla C(\mathbf{w})$ is the solution to the following problem (up to first order):

$$\delta\mathbf{w}_{\text{GD}} \quad = \quad \arg\min_{\delta\mathbf{w}} C(\mathbf{w}+\delta\mathbf{w}) \ \text{ s.t. } \ \|\delta\mathbf{w}\|_2 \leq c \tag{4.2.1}$$

We replace $C(\mathbf{w}+\delta\mathbf{w})$ by its linearization $C(\mathbf{w}+\delta\mathbf{w}) = C(\mathbf{w}) + \nabla C(\mathbf{w})^\top \delta\mathbf{w} + o(\|\delta\mathbf{w}\|)$ and neglect the higher order terms $o(\|\delta\mathbf{w}\|)$ for small values of $c$[1]. We also introduce a Lagrange multiplier $\lambda$ to enforce the constraint $\|\delta\mathbf{w}\|_2 \leq c$, which gives:

$$\delta\mathbf{w}_{\mathrm{GD}} = \arg\min_{\delta\mathbf{w}} C(\mathbf{w}) + \nabla C(\mathbf{w})^\top \delta\mathbf{w} + \lambda\left(\|\delta\mathbf{w}\|_2^2 - c^2\right) \tag{4.2.2}$$

The arg min can be solved in closed form and we obtain:

$$\delta\mathbf{w}_{\mathrm{GD}} = -\frac{1}{2\lambda}\nabla C(\mathbf{w}) \tag{4.2.3}$$

By writing the learning rate $\eta = \frac{1}{2\lambda}$ we recover the update of the gradient descent algorithm as expected[2].

**Steepest descent for the natural metric on the space of probability distributions.** We now present the natural gradient algorithm. To his end, let us restrict our hypothesis space $\mathcal{H}$ to be a set of parametric functions that represent probability density functions $p(z|\mathbf{w}) = f_{\mathbf{w}}(z)$.

Recall from section 2.4 in Chapter 2 that the Fisher Information Matrix (FIM) $F_{\mathbf{w}} = \mathbb{E}\left[D\mathbf{w}D\mathbf{w}^\top\right]$ can be used as a local metric on the space of probability density functions, where we denoted $D\mathbf{w} = \frac{\partial \log p(z|\mathbf{w})}{\partial \mathbf{w}}$ the derivative of the log likelihood. For a small increment $\delta\mathbf{w}$ the following equality holds:

$$\mathrm{KL}\left(p(z|\mathbf{w}) \,\|\, p(z|\mathbf{w}+\delta\mathbf{w})\right) = \frac{1}{2}\langle\delta\mathbf{w}, F_{\mathbf{w}}\delta\mathbf{w}\rangle + o\left(\|\delta\mathbf{w}\|^2\right) \tag{4.2.4}$$

We follow the same procedure as for the (euclidean) gradient descent algorithm, but instead of measuring the displacement $f_{\mathbf{w}+\delta\mathbf{w}} - f_{\mathbf{w}}$ using a budget in the parameter space, we will measure this budget in the function space using the FIM metric:

$$\delta\mathbf{w}_{\mathrm{NGD}} = \arg\min_{\delta\mathbf{w}} C(\mathbf{w}+\delta\mathbf{w}) \text{ s.t. } \|\delta\mathbf{w}\|_{F_{\mathbf{w}}} \leq c_1 \text{ and } \|\delta\mathbf{w}\|_2 \leq c_2 \tag{4.2.5}$$

The role of $c_1$ is to constrain the natural norm to be small, and the role of $c_2$ is to account for $2^{nd}$ order effects in both the cost function $C$ and the local approximation of the KL. Similarly as for the gradient descent, we replace $C(\mathbf{w}+\delta\mathbf{w})$ by its linearization and solve the arg min in closed form using Lagrange multipliers to obtain:

$$\delta\mathbf{w}_{\mathrm{NGD}} = -\eta\left(F_{\mathbf{w}} + \epsilon I\right)^{-1}\nabla C(\mathbf{w}) \tag{4.2.6}$$

[1]To be more precise, we need to choose $c$ such that $C(\mathbf{w}+\delta\mathbf{w})$ is well approximated by its linearization, that is we need to ensure that for all $\delta\mathbf{w}$ such that $\|\delta\mathbf{w}\|_2 \leq c$, the second order term in the Taylor expansion $\frac{1}{2}\delta\mathbf{w}^\top\nabla^2 C(\mathbf{w})\,\delta\mathbf{w}$ can be neglected compared to the first order term $\nabla C(\mathbf{w})^\top\delta\mathbf{w}$.

[2]In order to fulfill the constraint we must set $\lambda = \frac{\|\nabla C(\mathbf{w})\|_2}{2c}$.We instead generally use a constant or decaying learning rate $\eta$ so that the actual radius $c$ is adapted depending of the magnitude of the current gradient $\nabla C(\mathbf{w})$. For algorithms of the *trust region* framework, instead of adjusting the learning rate $\eta$ it is instead the radius $c$ that varies depending on the success of a step.

This update rule is called the *natural gradient descent* algorithm (Amari, 1998). It requires computing the $d \times d$ matrix $F_{\mathbf{w}}$, and then inverting it, or at least approximately solve the linear system $\delta \mathbf{w}_{\mathrm{NGD}} = \arg\min_{\delta \mathbf{w}} \|(F_{\mathbf{w}} + \epsilon I)\, \delta \mathbf{w} + \eta \nabla C(\mathbf{w})\|$ up to desired precision. In the following discussion, we will set $\epsilon = 0$ and just write $(F_{\mathbf{w}} + \epsilon I) = F_{\mathbf{w}}$ to simplify notations.

### 4.2.2. Natural gradient for optimization

Besides following the steepest descent direction as measured by the *canonical* metric for probability distributions (Amari, 1985), it is not entirely clear why using the natural gradient would accelerate convergence, over any other metric on function spaces, or even the euclidean metric on the parameter space.

In this section we motivate the use of the natural gradient algorithm for optimization, namely *why would the natural gradient algorithm accelerate convergence to a minimum of* $C(\mathbf{w})$?

**The case of large width 2-layers ReLU networks.** Following the work of Du et al. (2019b,a), a recent preprint (Zhang et al., 2019) shows that in the overparameterized case, and under some assumptions that seem to hold in practice, natural gradient allows for an improved convergence rate over gradient descent.

**Adaptation to the natural geometry of the function space.** Some directions in the parameter space incur very large moves in the function space, whereas some other directions do not affect the resulting function much, an observation that we further discuss in chapter 6. By using the natural gradient algorithm, we normalize all directions, so that updates in directions with large curvature of the FIM will be scaled down and updates in directions with small curvature of the FIM will be amplified. In that sense, it allows taking larger steps in directions of low curvature, without increasing a global learning rate that would be too large for directions of large curvature.

### 4.2.3. Compute/memory cost and approximations

Matrix $F_{\mathbf{w}}$ has a gigantic size $d \times d$, which makes it too big to compute and invert. In order to get a practical algorithm, we must find approximations of $F_{\mathbf{w}}$ that keep some of the relevant curvature information while removing the unnecessary and computationally costly parts.

**Block diagonal approximation.** A first simplification, adopted by nearly all prior approaches, consists in treating each layer of the neural network separately, ignoring cross-layer terms. This amounts to a first block-diagonal approximation of $F_{\mathbf{w}}$: each block $F_{\mathbf{w}}^{(l)}$ caters for the parameters of a single layer $l$.

**KFAC approximation.** Now $F_{\mathbf{w}}^{(l)}$ can typically still be extremely large. A cheap but very crude approximation consists in using a diagonal $F_{\mathbf{w}}^{(l)}$, i.e. taking into account the variance in each parameter dimension, but ignoring all covariance structure. A less stringent approximation (Heskes, 2000; Martens & Grosse, 2015) is to write $F_{\mathbf{w}}^{(l)}$ as a Kronecker product $F_{\mathbf{w}}^{(l)} \approx A \otimes B$ (see appendix A.1 for a definition and properties of the Kronecker product $\otimes$) which involves two smaller matrices, making it much cheaper to store, compute and invert[3].

Specifically for a layer $l$ that receives input $\tilde{a}$ of size $d_{\text{in}}$ and computes linear pre-activations $s = W\tilde{a}$ of size $d_{\text{out}}$ followed by some non-linear activation function, let the backpropagated gradient of the log likelihood on $s$ be $Ds = \frac{\partial \log p(z|\mathbf{w})}{\partial s}$. The gradients on parameters $\mathbf{w}^{(l)} = \text{vec}W$ will be $D\mathbf{w}^{(l)} = \frac{\partial \log p(z|\mathbf{w})}{\partial \mathbf{w}^{(l)}} = \text{vec}(Ds\tilde{a}^\top) = \tilde{a} \otimes Ds$. The Kronecker factored approximation of corresponding block $F_{\mathbf{w}}^{(l)} = \mathbb{E}\left[D\mathbf{w}^{(l)} D\mathbf{w}^{(l)\top}\right] = \mathbb{E}\left[\tilde{a}\tilde{a}^\top \otimes DsDs^\top\right]$ will use $A = \mathbb{E}\left[\tilde{a}\tilde{a}^\top\right]$ and $B = \mathbb{E}\left[DsDs^\top\right]$ i.e. matrices of size $d_{\text{in}} \times d_{\text{in}}$ and $d_{\text{out}} \times d_{\text{out}}$, whereas the full $F_{\mathbf{w}}^{(l)}$ would be of size $d_{\text{in}}d_{\text{out}} \times d_{\text{in}}d_{\text{out}}$. Using this Kronecker approximation (known as KFAC) corresponds to approximating entries of $F_{\mathbf{w}}^{(l)}$ as described in the following proposition:

**Proposition 4.2.1** (KFAC Martens & Grosse (2015))**.** *Assuming that $\tilde{a}\tilde{a}^\top$ and $DsDs^\top$ are uncorrelated (cov$\left(\tilde{a}\tilde{a}^\top, DsDs^\top\right) = 0$) we can break the expectation into two parts:*

$$F_{\mathbf{w}}^{(l)} = \mathbb{E}\left[\tilde{a}\tilde{a}^\top\right] \otimes \mathbb{E}\left[DsDs^\top\right] \coloneqq F_{KFAC} \tag{4.2.7}$$

Proof. Write $\text{cov}\left(\tilde{a}\tilde{a}^\top, DsDs^\top\right) = \mathbb{E}\left[\left(\tilde{a}\tilde{a}^\top - \mathbb{E}\tilde{a}\tilde{a}^\top\right) \otimes \left(DsDs^\top - \mathbb{E}DsDs^\top\right)\right] = \mathbb{E}\left[\tilde{a}\tilde{a}^\top \otimes DsDs^\top\right] - \mathbb{E}\left[DsDs^\top\right] \otimes \mathbb{E}\left[\tilde{a}\tilde{a}^\top\right]$. Assuming that $\text{cov}\left(\tilde{a}\tilde{a}^\top, DsDs^\top\right) = 0$ we obtain $F_{\mathbf{w}}^{(l)} = \mathbb{E}\left[\tilde{a}\tilde{a}^\top \otimes DsDs^\top\right] = \mathbb{E}\left[DsDs^\top\right] \otimes \mathbb{E}\left[\tilde{a}\tilde{a}^\top\right]$. □

In practice the assumption that $\tilde{a}\tilde{a}^\top$ and $DsDs^\top$ are uncorrelated approximately holds for real neural networks as observed in Martens & Grosse (2015).

**Approximate natural gradient algorithm using KFAC.** Wrapping together the block diagonal and KFAC approximations, we obtain the new step by computing for each layer:

$$\begin{aligned}
\delta\mathbf{w}_{\text{KFAC}}^{(l)} &= -\eta \left(F_{\text{KFAC}}^{(l)}\right)^{-1} \nabla C\left(\mathbf{w}^{(l)}\right) && (4.2.8)\\
&= -\eta \left(A^{-1} \otimes B^{-1}\right) \nabla C\left(\mathbf{w}^{(l)}\right) && (4.2.9)\\
&= -\eta B^{-1} \nabla C\left(W^{(l)}\right) A^{-1} && (4.2.10)
\end{aligned}$$

where by a slight abuse of notation we denote by $\nabla C\left(\mathbf{w}^{(l)}\right)$ the gradient of the cost function with respect to the parameters of layer $l$ arranged as a vector, and by $\nabla C\left(W^{(l)}\right)$ the same gradient but arranged as a matrix so that $\nabla C\left(W^{(l)}\right)_{ij} = \frac{\partial C}{\partial W_{ij}^{(l)}}$.

[3]Since $(A \otimes B)^{-1} = A^{-1} \otimes B^{-1}$.

This update in eq 4.2.10 further leverage the property of the Kronecker product since we do not even have to compute the large matrix $A^{-1} \otimes B^{-1}$ but instead we just multiply using the smaller matrices $A^{-1}$ and $B^{-1}$.

### 4.2.4. Structured curvature

Although this chapter focuses on the Fisher Information Matrix and the natural gradient, this technique of compressing a curvature matrix as a much smaller matrix made of Kronecker products is much more general and can be applied to other curvature matrices of neural networks. Examples of such matrices include:

- the Hessian matrix of the cost function with respect to the parameters whose diagonal blocks can be written as $H^{(l)} = \frac{1}{n} \sum_i \tilde{a}_i^{(l)} \tilde{a}_i^{(l)\top} \otimes \nabla^2 C \left( s_i^{(l)} \right)$ where $\nabla^2 C \left( s_i \right)$ is the Hessian of the cost function with respect to the pre-activation of layer $l$, for train set example $i$ ;
- an approximation thereof called the (generalized) Gauss-Newton matrix (Schraudolph, 2001), whose KFAC factorization has already been extensively studied in Botev et al. (2017) ;
- the covariance of the individual gradients of the empirical risk analyzed in TONGA (Le Roux et al., 2008) as a way of tackling the question of the noisy estimate of the gradients in stochastic gradient descent.

As we have briefly shown in section 4.2.3 this structured curvature comes from the arrangement of computation at the backbone of neural networks: the weight matrices represent linear maps from the activation of a layer $\tilde{a}$ to the pre-activation of the next layer $s$: $s = W\tilde{a}$. This linear map leads to a derivative of the form $g = \frac{\partial h}{\partial \mathbf{w}} = \tilde{a} \otimes \frac{\partial h}{\partial s}$ where $h$ is any function of $s$ located upstream in the neural network computation flow, and in particular in our case $h$ is the log likelihood of a given sample.

The approach that we take here is to leverage this structure instead of just using a black box algorithm such as the conjugate gradient (as in Martens et al. (2010)) for solving the linear subproblem $\delta \mathbf{w}_{\text{NGD}} = -\eta F_{\mathbf{w}}^{-1} \nabla C \left( \mathbf{w} \right)$ of the natural gradient algorithm. These 2 approaches have been compared in Martens & Grosse (2015), and KFAC proved faster for optimization on their benchmarks.

## 4.3. Proposed method

### 4.3.1. Motivation: reflexion on diagonal rescaling in different coordinate bases

***Diagonal*** **natural gradient in the canonical euclidean basis.** It is instructive to contrast the type of "exact" natural gradient preconditioning of the gradient that uses the full

Fisher Information Matrix would yield, to what we do when approximating this by using a diagonal matrix only. Using the full matrix $F_{\mathbf{w}} = \mathbb{E}[D\mathbf{w}D\mathbf{w}^\top]$ yields the natural gradient update: $\delta\mathbf{w}_{\text{NGD}} = -\eta F_{\mathbf{w}}^{-1}\nabla C(\mathbf{w})$. When resorting to a diagonal approximation we instead use $F_{\text{diag}} = \text{diag}(\sigma_1^2, \ldots, \sigma_d^2)$ where $\sigma_i^2 = (F_{\mathbf{w}})_{i,i} = \mathbb{E}[D\mathbf{w}_i^2]$. So that update $\delta\mathbf{w}_{\text{diag}} = -\eta F_{\text{diag}}^{-1}\nabla C(\mathbf{w})$ amounts to preconditioning the gradient vector $\nabla C(\mathbf{w})$ by dividing each of its coordinates by an estimated second moment $\sigma_i^2$. This diagonal rescaling happens in the canonical euclidean basis of parameters $\mathbf{w}$.

***True* natural gradient in FIM eigenbasis.** By contrast, a full natural gradient update can be seen to do a similar diagonal rescaling, not along the initial parameter basis axes, but along the axes of the *eigenbasis* of the matrix $F_{\mathbf{w}}$. Let $F_{\mathbf{w}} = USU^\top$ be the eigendecomposition of $F_{\mathbf{w}}$. The operations that yield the full natural gradient update $F_{\mathbf{w}}^{-1}\nabla C(\mathbf{w}) = US^{-1}U^\top\nabla C(\mathbf{w})$ correspond to the sequence of:

(1) multiplying gradient vector $\nabla C(\mathbf{w})$ by $U^\top$ which corresponds to switching to the eigenbasis: $U^\top\nabla C(\mathbf{w})$ yields the coordinates of the gradient vector expressed in that basis

(2) multiplying by a diagonal matrix $S^{-1}$, which rescales each coordinate $i$ (in that eigenbasis) by $S_{ii}^{-1}$

(3) multiplying by $U$, which switches the rescaled vector back to the initial basis of parameters.

It is easy to show that $S_{ii} = \mathbb{E}[(U^\top D\mathbf{w})_i^2]$. So similarly to what we do when using a diagonal approximation, we are rescaling by the second moment of gradient vector components, but rather than doing this in the initial parameter basis, we do this in the eigenbasis of $F_{\mathbf{w}}$. Note that the variance measured along the leading eigenvector can be very different to the variance along the axes of the initial parameter basis, so the effects of the rescaling by using either the full $F_{\mathbf{w}}$ or its diagonal approximation can be very different.

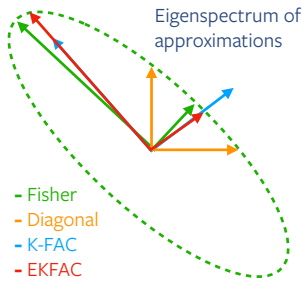
**KFAC: Diagonal factorization in a Kronecker-factored Eigenbasis.** Now what happens when we use the less crude KFAC approximation instead? We approximate $F_{\mathbf{w}}^{(l)} \approx A \otimes B$ yielding the update $\delta\mathbf{w}_{\text{KFAC}}^{(l)} = -\eta\,(A^{-1}\otimes B^{-1})\,\nabla C\left(\mathbf{w}^{(l)}\right)$. Let us similarly look at it through its eigendecomposition. The eigendecomposition of the Kronecker product $A \otimes B$ of two real symmetric positive semi-definite matrices can be expressed using their own eigendecomposition $A = U_A S_A U_A^\top$ and $B = U_B S_B U_B^\top$, yielding $A \otimes B = (U_A S_A U_A^\top) \otimes (U_B S_B U_B^\top) = (U_A \otimes U_B)(S_A \otimes S_B)(U_A \otimes U_B)^\top$. $U_A \otimes U_B$ gives the orthogonal eigenbasis of the Kronecker product, we call it the *Kronecker-Factored Eigenbasis* (KFE). $S_A \otimes S_B$ is the diagonal matrix containing the associated eigenvalues. Note that each such eigenvalue will be a product of an eigenvalue of $A$ stored in $S_A$ and an eigenvalue of $B$ stored in $S_B$. We can view the action of the resulting Kronecker-factored preconditioning in the same way as we viewed the preconditioning by the full matrix: it consists in

(1) expressing gradient vector $\nabla C\left(\mathbf{w}^{(l)}\right)$ in a different basis $U_A \otimes U_B$ which can be thought of as approximating the directions of $U$,
(2) doing a diagonal rescaling by $S_A \otimes S_B$ *in that basis*
(3) switching back to the initial parameter space.

Here however the rescaling factor $(S_A \otimes S_B)_{ii}$ is *not* guaranteed to match the second moment along the associated eigenvector $\mathbb{E}[((U_A \otimes U_B)^\top D\mathbf{w})_i^2]$.

In summary (see Figure 1):

- Full matrix $F_\mathbf{w}$ preconditioning will scale by variances estimated along the eigenbasis of $F_\mathbf{w}$.
- Diagonal preconditioning will scale by variances properly estimated, but along the initial parameter basis, which can be very far from the eigenbasis of $F_\mathbf{w}$.
- KFAC preconditioning will scale the gradient along the KFE basis that will likely be closer to the eigenbasis of $F_\mathbf{w}$, but doesn't use properly estimated variances along these axes for this scaling (the scales being themselves constrained to being a Kronecker product $S_A \otimes S_B$).




Rescaling of the gradient is done along a specific basis; length of vectors indicate (square root of) amount of downscaling. Exact Fisher Information Matrix rescales according to eigenvectors/values of exact covariance structure (green ellipse). Diagonal approximation uses parameter coordinate basis, scaling by actual variance measured along these axes (indicated by horizontal and vertical orange arrows touching exactly the ellipse), KFAC uses directions that approximate Fisher Information Matrix eigenvectors, but uses approximate scaling (blue arrows *not* touching the ellipse). EKFAC corrects this.

**Figure 1.** Cartoon illustration of rescaling achieved by different preconditioning strategies

### 4.3.2. Eigenvalue-corrected Kronecker Factorization (EKFAC)

To correct for the potentially inexact rescaling of KFAC, and obtain a better but still computationally efficient approximation, instead of $F_{\text{KFAC}} = A \otimes B = (U_A \otimes U_B)(S_A \otimes S_B)(U_A \otimes U_B)^\top$ we propose to use an *Eigenvalue-corrected Kronecker Factorization*: $F_{\text{EKFAC}} = (U_A \otimes U_B)S^*(U_A \otimes U_B)^\top$ where $S^*$ is the diagonal matrix defined by $S^*_{ii} = s^*_i = \mathbb{E}[((U_A \otimes U_B)^\top D\mathbf{w})_i^2]$. Vector $s^*$ is the vector of second moments of the gradient vector coordinates expressed in the Kronecker-factored Eigenbasis (KFE) $U_A \otimes U_B$ and can be efficiently estimated and stored.

In section A.2.1 in appendix, we prove that this $S^*$ is the optimal diagonal rescaling in that basis, in the sense that $S^* = \arg\min_S \|F_\mathbf{w} - (U_A \otimes U_B)S(U_A \otimes U_B)^\top\|_2{}^4$ s.t. $S$ is diagonal: it minimizes the approximation error to $F_\mathbf{w}$ as measured by Frobenius norm, which KFAC's

[4] here $\|\cdot\|_2$ denotes the Frobenius norm

corresponding $S = S_A \otimes S_B$ cannot generally achieve. A corollary of this is that we will always have $\|F_{\mathbf{w}} - F_{\text{EKFAC}}\|_2 \leq \|F_{\mathbf{w}} - F_{\text{KFAC}}\|_2$ i.e. EKFAC yields a better approximation of $F_{\mathbf{w}}$ than KFAC (Theorem 1 proven in Appendix). Figure 1 illustrates the different rescaling strategies, including EKFAC.

**Potential benefits:** Since EKFAC is a better approximation of $F_{\mathbf{w}}$ than KFAC (in the limited sense of Frobenius norm of the residual) it may yield a better preconditioning of the gradient for optimizing neural networks[5]. Another benefit is linked to computational efficiency: even if KFAC yields a reasonably good approximation in practice, it is costly to re-estimate and invert matrices $A$ and $B$, so this has to be amortized over many updates: re-estimation of the preconditioning is thus typically done at a much lower frequency than the parameter updates, and may lag behind, no longer accurately reflecting the local $2^{\text{nd}}$ order information. Re-estimating the Kronecker-factored Eigenbasis (KFE) for EKFAC is similarly costly and must be similarly amortized. But re-estimating the diagonal scaling $s^*$ in that basis is cheap, doable with every mini-batch, so we can hope to reactively track and leverage the changes in $2^{\text{nd}}$ order information along these directions. Figure 2 (right) provides an empirical confirmation that tracking $s^*$ indeed allows to keep the approximation error of $F_{\text{EKFAC}}$ small, compared to $F_{\text{KFAC}}$, between recomputations of basis or inverse.

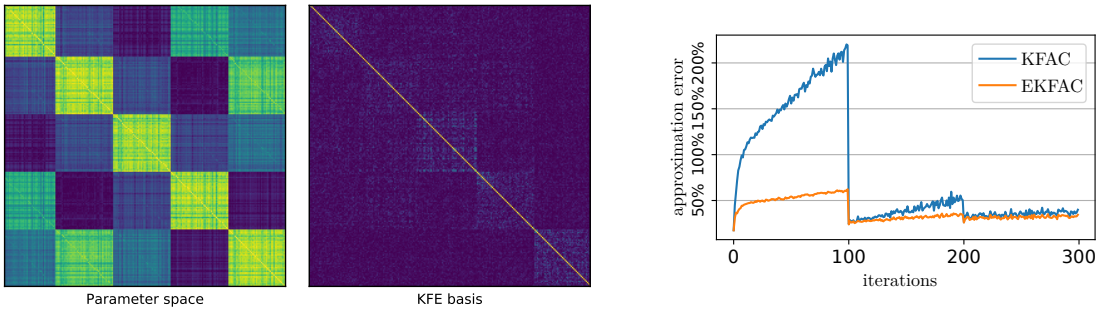



**Figure 2.** **Left:** Gradient *correlation* matrices measured in the initial parameter basis and in the Kronecker-factored Eigenbasis (KFE), computed from a small 4 sigmoid layer MLP classifier trained on MNIST. Block corresponds to 250 parameters in the 2nd layer. Components are largely decorrelated in the KFE, justifying the use of a diagonal rescaling method *in that basis*.
**Right:** Approximation error $\frac{\|F_{\mathbf{w}} - \hat{F}\|_2}{\|F_{\mathbf{w}}\|_2}$ where $\hat{F}$ is either $F_{\text{KFAC}}$ or $F_{\text{EKFAC}}$, for the small MNIST classifier. KFE basis and KFAC inverse are recomputed every 100 iterations. EKFAC's cheap tracking of $s^*$ allows it to drift far less quickly than amortized KFAC from the exact empirical Fisher.

**Dual view by working in the KFE:.** Instead of thinking of this new method as an improved factorized approximation of $G$ that we use as a preconditioning matrix, we can alternatively view it as applying a *diagonal method, but in a different basis where the diagonal*

[5]Although there is no guarantee. In particular $F_{\text{EKFAC}}$ being a better approximation of $F$ does not guarantee that $F^{-1}_{\text{EKFAC}} \nabla C\left(\mathbf{w}^{(l)}\right)$ will be closer to the natural gradient update direction $F^{-1} \nabla C\left(\mathbf{w}^{(l)}\right)$ .

*approximation is more accurate* (an assumption we empirically confirm in Figure 2 left). This can be seen by interpreting the update given by EKFAC as a 3 step process: project the gradient in the KFE (—), apply *diagonal* natural gradient descent in this basis (—), then project back to the parameter space (—):

$$F_{\text{EKFAC}}^{-1} \nabla C\left(\mathbf{w}^{(l)}\right) = \left(U_A \otimes U_B\right) S^{*-1} \left(U_A \otimes U_B\right)^{\top} \nabla C\left(\mathbf{w}^{(l)}\right) \tag{4.3.1}$$

Note that by writing $\tilde{D}\mathbf{w} = \left(U_A \otimes U_B\right)^{\top} D\mathbf{w}$ the projected gradient in the KFE, the computation of the coefficients $s_i^*$ simplifies in $s_i^* = \mathbb{E}[(\tilde{D}\mathbf{w})_i^2]$. Figure 2 shows gradient correlation matrices in both the initial parameter basis and in the KFE. Gradient components appear far less correlated when expressed in the KFE, which justifies using a diagonal rescaling method *in that basis*.

This viewpoint brings us close to network reparameterization approaches such as Fujimoto & Ohira (2018), whose proposal – that was already hinted towards by Desjardins et al. (2015) – amounts to a reparameterization equivalent of KFAC. More precisely, while Desjardins et al. (2015) empirically explored a reparameterization that uses only input covariance $A$ (and thus corresponds only to "half of" KFAC), Fujimoto & Ohira (2018) extend this to use also backpropagated gradient covariance $B$, making it essentially equivalent to KFAC (with a few extra twists). Our approach differs in that moving to the KFE corresponds to a change of *orthonormal basis*, and more importantly that we *cheaply track* and perform a more optimal *full diagonal* rescaling in that basis, rather than the constrained factored $S_A \otimes S_B$ diagonal that these other approaches are implicitly using.

**Algorithm:** Using Eigenvalue-corrected Kronecker factorization (EKFAC) for neural network optimization involves: 1/ periodically (every $n$ mini-batches) computing the Kronecker-factored Eigenbasis by doing an eigendecomposition of the same $A$ and $B$ matrices as KFAC; 2/ estimating scaling vector $s^*$ as second moments of gradient coordinates in that implied basis; 3/ preconditioning gradients accordingly prior to updating model parameters. Algorithm 1 provides a high level pseudo-code of EKFAC, and when using it to approximate the empirical Fisher. In this version, we re-estimate $s^*$ from scratch on each mini-batch. An alternative (EKFAC-ra) is to update $s^*$ as a *running average* of component-wise second moment of mini-batch averaged gradients.

## 4.4. Experiments

This section presents an empirical evaluation of our proposed Eigenvalue Corrected KFAC (EKFAC) algorithm in two variants: EKFAC estimates scalings $s^*$ as second moment of intrabatch gradients (in KFE coordinates) as in Algorithm 1, whereas EKFAC-ra estimates $s^*$ as a running average of squared minibatch gradient (in KFE coordinates). We compare

**Algorithm 1** EKFAC for fully connected networks

**Require:** $n$: recompute eigenbasis every $n$ minibatches
**Require:** $\eta$: learning rate
**Require:** $\epsilon$: damping parameter

**procedure** EKFAC($\mathcal{D}_{\text{train}}$)
  **while** convergence is not reached, iteration $i$ **do**
    sample a minibatch $\mathcal{D}$ from $\mathcal{D}_{\text{train}}$
    Do forward and backprop pass as needed to obtain $h$ and $\delta$
    **for all** layer $l$ **do**
      **if** $i \,\%\, n = 0$ **then** ▷ Amortize eigendecomposition
        ComputeEigenBasis($\mathcal{D}$, $l$)
      **end if**
      ComputeScalings($\mathcal{D}$, $l$)
      $\nabla^{\text{mini}} \leftarrow \mathbb{E}_{(x,y)\in\mathcal{D}}\left[\nabla_\theta^{(l)}(x,y)\right]$
      UpdateParameters($\nabla^{\text{mini}}$, $l$)
    **end for**
  **end while**
**end procedure**

**procedure** ComputeEigenBasis($\mathcal{D}$, $l$)
  $U_A^{(l)}, S_A^{(l)} \leftarrow$ eigendecomposition $\left(\mathbb{E}_{\mathcal{D}}\left[h^{(l)}h^{(l)\top}\right]\right)$
  $U_B^{(l)}, S_B^{(l)} \leftarrow$ eigendecomposition $\left(\mathbb{E}_{\mathcal{D}}\left[\delta^{(l)}\delta^{(l)\top}\right]\right)$
**end procedure**

**procedure** ComputeScalings($\mathcal{D}$, $l$)
  $s^{*(l)} \leftarrow \mathbb{E}_{\mathcal{D}}\left[\left(\left(U_A^{(l)} \otimes U_B^{(l)}\right)^\top \nabla_\theta^{(l)}\right)^2\right]$ ▷ Project gradient in eigenbasis[1]
**end procedure**

**procedure** UpdateParameters($\nabla^{\text{mini}}$, $l$)
  $\tilde{\nabla} \leftarrow \left(U_A^{(l)} \otimes U_B^{(l)}\right)^\top \nabla^{\text{mini}}$ ▷ Project gradients in eigenbasis[1]
  $\tilde{\nabla} \leftarrow \tilde{\nabla} / \left(s^{*(l)} + \epsilon\right)$ ▷ Element-wise scaling
  $\nabla^{\text{precond}} \leftarrow \left(U_A^{(l)} \otimes U_B^{(l)}\right) \tilde{\nabla}$ ▷ Project back in parameter basis[1]
  $\theta^{(l)} \leftarrow \theta^{(l)} - \eta \nabla^{\text{precond}}$ ▷ Update parameters
**end procedure**

them with KFAC and other baselines, primarily SGD with momentum, with and without batch-normalization (BN). For all our experiments KFAC and EKFAC approximate the *empirical $L^2$ metric* $\tilde{G}_{\mathbf{w}}$ (section 2.4). This research focuses on improving optimization techniques, so except when specified otherwise, we performed model and hyperparameter

selection based on the performance of the optimization objective, i.e. on the empirical loss computed on the training set.

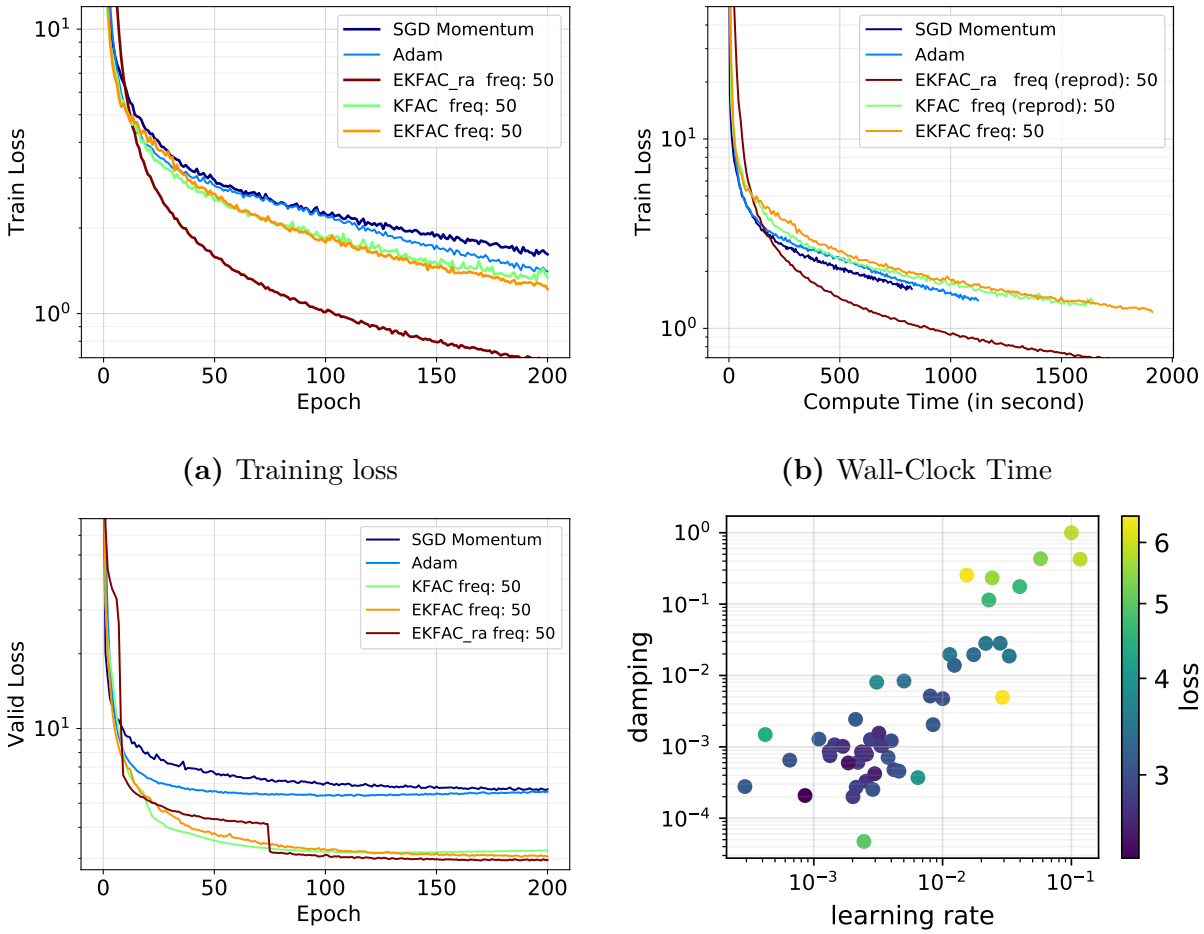
### 4.4.1. Deep auto-encoder

We consider the task of minimizing the reconstruction error of an 8-layer auto-encoder on the MNIST dataset, a standard task used to benchmark optimization algorithms applied to neural networks (Hinton & Salakhutdinov, 2006; Martens & Grosse, 2015; Desjardins et al., 2015). The model consists of an encoder composed of 4 fully-connected sigmoid layers, with a number of hidden units per layer of respectively 1000, 500, 250, 30, and a symmetric decoder (with untied weights).

We compare EKFAC, computing the second moment statistics through its mini-batch, and EKFAC-ra, its running average variant, with different baselines (KFAC, SGD with momentum and BN, ADAM with BN). For each algorithm, best hyperparameters were selected using a mix of grid and random search based on training error. Grid values for hyperparameters are: learning rate $\eta$ and damping $\epsilon$ in $\{10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}\}$, mini-batch size in $\{200, 500\}$. In addition we explored 20 values for ($\eta$,$\epsilon$) by random search around each grid points. We found that extra care must be taken when choosing the values of the learning rate and damping parameter $\epsilon$ in order to get good performances, as often observed when working with algorithms that leverage curvature information (see Figure 3 (d)). The learning rate and the damping parameters are kept constant during training.

Figure 3 (a) reports the training loss throughout training and shows that EKFAC and EKFAC-ra both minimize faster the training loss per epoch than KFAC and the other baselines. In addition, Figure 3 (b) shows that the efficient tracking of diagonal scaling vector $s^*$ in EKFAC-ra, despite its slightly increased computational burden per update, allows to achieve faster training in wall-clock time. Finally, on this task, EKFAC and EKFAC-ra achieve better optimization while also maintaining a very good generalization performance (Figure 3 (c)).

**Frequency of amortization.** Next we investigate how the frequency of the inverse/eigendecomposition recomputation affects optimization. In Figure 4, we compare KFAC/EKFAC with different reparametrization frequencies to a strong KFAC baseline where we reestimate and compute the inverse at each update. This baseline outperforms the amortized version (as a function of number of epochs), and is likely to leverage a better approximation of $G$ as it recomputes the approximated eigenbasis *at each update*. However it comes at a high computational cost, as seen in Figure 4 (b). Amortizing the eigendecomposition allows to strongly decrease the computational cost while slightly degrading the optimization performances. As can be seen in Figure 4 (a), amortized EKFAC preserves better the optimization performance than amortized KFAC. EKFAC re-estimates at each update, the diagonal second moments

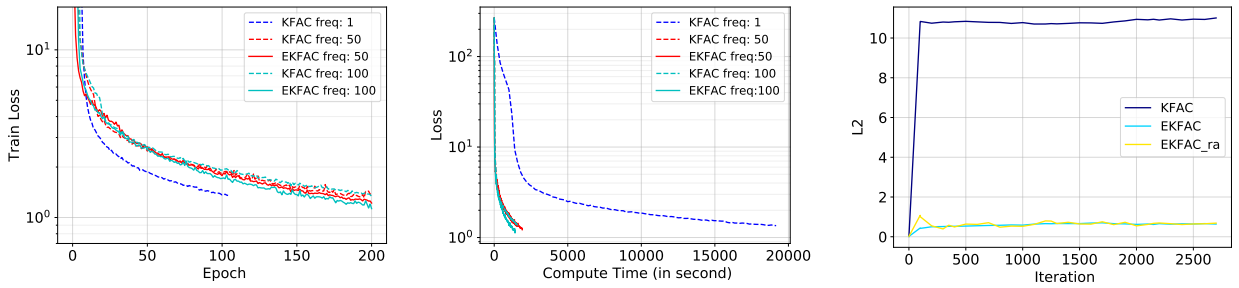



**(a)** Training loss **(b)** Wall-Clock Time

**(c)** Validation Loss **(d)** Hyperparameters

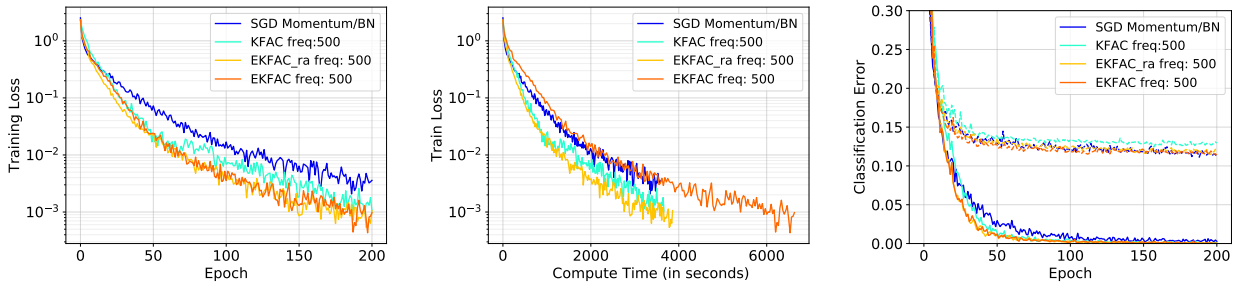
**Figure 3.** MNIST Deep Auto-Encoder task. Models are selected based on the best loss achieved during training. SGD and Adam are with batch-norm. A "freq" of 50 means eigendecomposition or inverse is recomputed every 50 updates. **(a)** Training loss vs epochs. Both EKFAC and EKFAC-ra show an optimization benefit compared to amortized-KFAC and the other baselines. **(b)** Training loss vs wall-clock time. Optimization benefits transfer to faster training for EKFAC-ra. **(c)** Validation performance. KFAC and EKFAC achieve a similar validation performance. **(d)** Sensitivity to hyperparameters values. Color corresponds to final loss reached after 20 epochs for batch size 200.

$s^*$ in the KFE basis, which correspond to the eigenvalues of the EKFAC approximation of $G$. Thus it should better track the true curvature of the function space. To verify this, we investigate how the eigenspectrum of the true empirical Fisher $G$ changes compared to the eigenspectrum of its approximations as $G_{\text{KFAC}}$ (or $G_{\text{EKFAC}}$). In Figure 4 (c), we track their eigenspectra and report the $\ell_2$ distance between them during training. We compute the KFE once at the beginning and then keep it fixed during training. We focus on the $4^{\text{th}}$ layer of the auto-encoder: its small size allows to estimate the corresponding $G$ and compute its eigenspectrum at a reasonable cost. We see that the spectrum of $G_{\text{KFAC}}$ quickly diverges




**(a)** Training loss **(b)** Wall-clock time **(c)** $\ell_2$ distance to spectrum of $G$

**Figure 4.** Impact of frequency of inverse/eigendecomposition recomputation for KFAC/EKFAC. A "freq" of 50 indicates a recomputation every 50 updates. **(a)(b)** Training loss v.s. epochs and wall-clock time. We see that EKFAC preserves better its optimization performances when the eigendecomposition is performed less frequently. **(c)**. Evolution of $\ell_2$ distance between the eigenspectrum of empirical Fisher $G$ and eigenspectra of approximations $G_{\text{KFAC}}$ and $G_{\text{EKFAC}}$. We see that the spectrum of $G_{\text{KFAC}}$ quickly diverges from the spectrum of $G$, whereas the EKFAC variants, thanks to their frequent diagonal reestimation, manage to much better track $G$.




**(a)** Training loss **(b)** Wall-clock time **(c)** Accuracy (solid is train, dash is validation)

**Figure 5.** VGG11 on CIFAR-10. "freq" corresponds to the eigendecomposition (inverse) frequency. In **(a)** and **(b)**, we report the performance of the hyper-parameters reaching the lowest training loss for each epoch (to highlight which optimizers perform best given a fixed epoch budget). In **(c)** models are selected according to the best overall validation error. When the inverse/eigendecomposition is amortized on 500 iterations, EKFAC-ra shows an optimization benefit while maintaining its generalization capability.

from the spectrum of $G$, whereas the cheap frequent reestimation of the diagonal scaling for $G_{\text{EKFAC}}$ and $G_{\text{EKFAC}-\text{ra}}$ allows their spectrum to stay much closer to that of $G$. This is consistent with the evolution of approximation error shown earlier in Figure 2 on the small MNIST classifier.

### 4.4.2. CIFAR-10

In this section, we evaluate our proposed algorithm on the CIFAR-10 dataset using a VGG11 convolutional neural network (Simonyan & Zisserman, 2015) and a Resnet34 (He

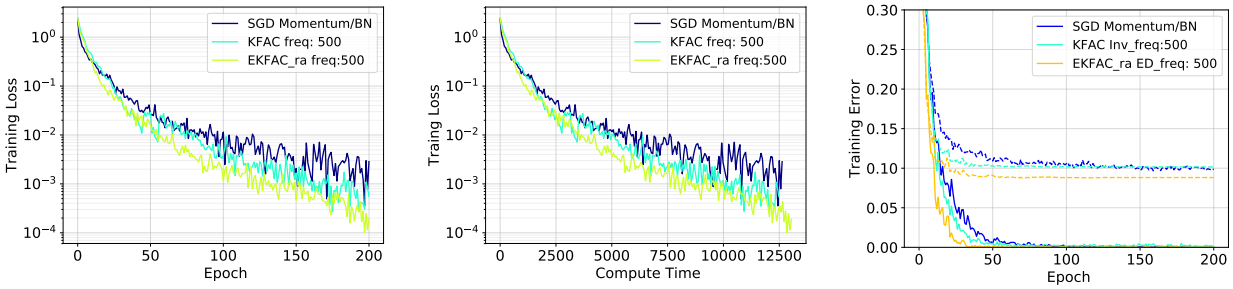
et al., 2016a). To implement KFAC/EKFAC in a convolutional neural network, we rely on the SUA approximation (Grosse & Martens, 2016) which has been shown to be competitive in practice (Laurent et al., 2018). We highlight that we do not use BN in our model when they are trained using KFAC/EKFAC, whereas training with Adam or SGD with momentum requires including BN in order to be competitive. As in the previous experiments, a grid search is performed to select the hyperparameters. Around each grid point, learning rate and damping values are further explored through random search.

In Figure 5, we compare EKFAC/EKFAC-ra to KFAC and SGD Momentum with or without BN when training a VGG-11 network. We use a batch size of 500 for the KFAC based approaches and 200 for the SGD baselines. Figure 5 (a) show that EKFAC yields better optimization than the SGD baselines and KFAC in training loss per epoch when the computation of the KFE is amortized. Figure 5 (c) also shows that models trained with EKFAC maintain good generalization. EKFAC-ra shows some wall-clock time improvements over the baselines in that setting ( Figure 5 (b)). However, we observe that using KFAC with a batch size of 200 can catch-up with EKFAC (but not EKFAC-ra) in wall-clock time despite being outperformed in term of optimization per iteration. VGG11 is a relatively small network by modern standard and the KFAC (with SUA approximation) remains computationally bearable in this model. We hypothesize that using smaller batches, KFAC can be updated often enough per epoch to have a reasonable estimation error while not paying too high a computational price.

In Figure 6, we report similar results when training a Resnet34 He et al. (2016b). We compare EKFAC-ra with KFAC, and SGD with momentum and BN. To be able to train the Resnet34 without BN, we need to rely on a careful initialization scheme (detailed in Appendix A.3) in order to ensure good signal propagation during the forward and backward passes. EKFAC-ra outperforms both KFAC (when amortized) and SGD with momentum and BN in term of optimization per epochs, and compute time. This gain appears robust across different batch sizes.

## 4.5. Discussion

In this work, we introduced the Eigenvalue-corrected Kronecker factorization (EKFAC), an approximate factorization of the Fisher Information Matrix that is computationally manageable while still being accurate. We formally proved that EKFAC yields a more accurate approximation than its closest parent and competitor KFAC, in the sense of the Frobenius norm. Of more practical importance, we showed that our algorithm allows to cheaply perform partial updates of the curvature estimate, by maintaining an up-to-date estimate of its eigenvalues while keeping the estimate of its eigenbasis fixed. This partial updating proves




**(a)** Training loss **(b)** Wall-clock time **(c)** Accuracy (solid is train, dash is validation)

**Figure 6.** Training a Resnet Network with 34 layers on CIFAR-10. "freq" corresponds to eigendecomposition (inverse) frequency. In **(a)** and **(b)**, we report the performance of the hyper-parameters reaching the lowest training loss for each epoch (to highlight which optimizers perform best given a fixed epoch budget). In **(c)** we select model according to the best overall validation error. When the inverse/eigen decomposition is amortized on 500 iterations, EKFAC-ra shows optimization and computational time benefits while maintaining a good generalization capability.

competitive when applied to optimizing deep networks, both with respect to the number of iterations and wall-clock time.

This concludes our chapter on EKFAC and the part of this thesis dedicated to optimization. In the next chapters, we will be interested in getting theoretical insights into the generalization mechanisms that make deep networks generalize well.

# Chapter 5

# Lazy vs hasty: linearization in deep networks impacts learning schedule based on example difficulty

## Prologue

In this chapter, we compare the training dynamics of deep networks, to those of linear models using the initial neural tangent kernel (NTK). Instead of using signal analysis tools to characterize the differences between learned predictors, we use groups of examples sorted by 3 different notions of learning difficulty and show that the difference between linear and non-linear regimes can be appreciated in how they prioritize easy and difficult examples.

***Article Details.*** *Lazy vs hasty: linearization in deep networks impacts learning schedule based on example difficulty.* Thomas George, Guillaume Lajoie, Aristide Baratin. The paper has been published in *TMLR 2022* (George et al., 2022).

**Context.** A main challenge in understanding the generalization properties of deep networks is the difficulty in characterizing the solution at the end of training. Indeed, the optimization problem is non-convex with possibly infinitely many local optima, unlike e.g. ridge regression where there is an analytical solution. Our approach is instead to try and model the training dynamics, as accurately as possible, with tools that are amenable to analytical treatment.

The Neural Tangent Kernel (NTK) paper (Jacot et al., 2018) has popularized the idea of importing theoretical concepts from the richer literature on kernelized linear models to the study of deep networks, for which there is currently no comprehensive theory of generalization. In particular, it has inspired a series of works on overparameterized (with possibly infinitely many parameters) linear models (Bartlett et al., 2021; Mei & Montanari, 2022; Loureiro et al., 2021) . While elegant, the infinite width limiting regime arguably does not capture the phenomena at play in the standard finite regime. We were interested in studying how the true learning regime departs from its idealized linearly trained counterpart.

A second challenge for an analytical model is to take into account the structure of the data. Idealized theoretical settings often resort to crude simplifications (e.g. data is sampled from a Gaussian distribution, or it lies on a sphere). By contrast, we directly work with examples coming from the true dataset. We group examples by 'difficulty' using C-scores (Jiang et al., 2021), in the context of spurious examples (Sagawa et al., 2020b), or in the presence of noisy labels. Our work takes part in a series of recent works that analyze machine learning models by studying which examples are easy or difficult to learn (Koh & Liang, 2017; Toneva et al., 2019; Pruthi et al., 2020; Baldock et al., 2021).

This paper has been published after our last paper in Chapter 6. We chose to include it first since it provides a motivation for the next chapter, and conversely the next chapter offers a mechanistic explanation of how learning in the true regime magnifies the sequentialization between easy and difficult examples.

**Developments.** As of this writing, this is a very recent publication (accepted for publication in December 2022). It has not yet inspired any follow-up work.

**Personal Contribution.** I had been experimenting with different frameworks to model the training dynamics of deep networks using tractable approximations. Aristide proposed to directly look at examples. Guillaume came up with the ideas of normalizing progress adapted to our experiments. I implemented all experiments and generated all plots. I and Aristide did most of the writing. Aristide contributed to the analytical model. Guillaume and Aristide supervised the project, and significantly improved the manuscript.

## 5.1. Introduction

Understanding the performance of deep learning algorithms has been the subject of intense research efforts in the past few years, driven in part by observed phenomena that seem to defy conventional statistical wisdom (Neyshabur et al., 2014; Zhang et al., 2017a; Belkin et al., 2019). Notably, many such phenomena have been analyzed rigorously in simpler contexts of high dimensional linear or random feature models (e.g., Hastie et al., 2022; Bartlett et al., 2021), which shed a new light on the crucial role of overparametrization in the performance of such systems. These results also apply to the so-called *lazy training* regime (Chizat et al., 2019), in which a deep network can be well approximated by its linearization at initialization, characterized by the neural tangent kernel (NTK) (Jacot et al., 2018; Du et al., 2019b; Allen-Zhu et al., 2019). In this regime, deep networks inherit the inductive bias and generalization properties of kernelized linear models.

It is also clear, however, that deep models cannot be understood solely through their kernel approximation. Trained outside the lazy regime (e.g., Woodworth et al., 2020), they are able to learn adaptive representations, with a time-varying tangent kernel (Fort et al., 2020) that specializes to the task during training (Kopitkov & Indelman, 2020; Baratin et al.,

2021; Paccolat et al., 2021; Ortiz-Jiménez et al., 2021). Several results showed examples where their inductive bias cannot be characterized in terms of a kernel norm (Savarese et al., 2019; Williams et al., 2019), or where they provably outperform any linear method (Malach et al., 2021). Yet, the specific mechanisms by which the two regimes differ, which could explain the performance gaps often observed in practice (Chizat et al., 2019; Geiger et al., 2020), are only partially understood.

Our work contributes new qualitative insights into this problem, by **investigating the comparative effect of the lazy and feature learning regimes on the training dynamics of various groups of examples of increasing difficulty**. We do so by means of a control parameter that modulates linearity of the parametrization and smoothly interpolates the two training regimes (Chizat et al., 2019; Woodworth et al., 2020). We provide empirical evidence and theoretical insights suggesting that **the feature learning regime puts higher weight on easy examples** at the beginning of training, which results in an increased learning speed compared to more difficult examples. This can be understood as an instance of the **simplicity bias** brought forward in recent work (Arpit et al., 2017; Rahaman et al., 2019; Kalimeris et al., 2019), which we show here to be much more pronounced in the feature learning regime. It also resonates with the old idea that generalization can benefit from some curriculum learning strategy (Elman, 1993; Bengio et al., 2009).

**Contributions.**

- We introduce and test the hypothesis of qualitatively different example importance between linear and non-linear regimes;
- Using adequately normalized plots, we present a unified picture using 4 different ways to quantify example difficulty, where easy examples are prioritized in the non-linear regime. We illustrate empirically this phenomenon across different ways to quantify example difficulty, including c-score (Jiang et al., 2021), label noise, and spurious correlation to some easy-to-learn set of features.
- We illustrate some of our insights in a simple quadratic model amenable to analytical treatment.

The general setup of our experiments is introduced in Section 5.2. Section 5.3 is our empirical study, which begins with an illustrative example on a toy dataset (Section 5.3.1), followed by experiments on CIFAR 10 in two setups where example difficulty is quantified using respectively c-scores and label noise (Section 5.3.2). We also examine standard setups where easy examples are those with strong correlations between their labels and some spurious features (Section 5.3.3). Section 5.4 illustrates our findings with a theoretical analysis of a specific class of quadratic models, whose training dynamics is solvable in both regimes. We conclude in Section 5.5.

**Related work.** The neural tangent kernel was initially introduced in the context of infinitely wide networks, for a specific parametrization that provably leads to the lazy regime (Jacot et al., 2018). Such a regime allows to cast deep learning as a linear model using a fixed kernel, enabling import of well-known results from linear models, such as guarantees of convergence to a global optimum (**?**Du et al., 2019b; Allen-Zhu et al., 2019). On the other hand, it is also clear that the kernel regime does not fully capture the behavior of deep models – including, in fact, infinitely wide networks (Yang & Hu, 2021). For example, in the so-called mean field limit, training two-layer networks by gradient descent learns adaptive representations (Chizat & Bach, 2018; Mei et al., 2018) and it can be shown that the inductive bias cannot be characterized in terms of a RKHS norm (Savarese et al., 2019; Williams et al., 2019). Performance gaps between the two regimes are also often observed in practice (Chizat et al., 2019; Arora et al., 2019; Geiger et al., 2020).

Subsequent work showed and analyzed how, for a fixed (finite-width) network, the scaling of the model at initialization also controls the transition between the lazy regime, governed by the empirical neural tangent kernel, and the standard feature learning regime (Chizat et al., 2019; Woodworth et al., 2020; Agarwala et al., 2020). In line with this prior work, in our experiments below we use a scaling parameter $\alpha > 0$ that modulates linearity of the parametrization and allows us to smoothly interpolates between the vanilla training ($\alpha = 1$) where features are learned and the lazy ($\alpha \to \infty$) regime where they are not. We compare training runs with various values of $\alpha$ and empirically assess linearity with several metrics described in Section 5.2 below.

Our results are in line with a group of work showing how deep networks learn patterns and functions of increasing complexity during training (Arpit et al., 2017; Kalimeris et al., 2019). An instance of this is the so-called spectral bias empirically observed in Rahaman et al. (2019) where a sum of sinusoidal signals is incrementally learned from low frequencies to high frequencies, or in Zhang et al. (2021) that use Fourier decomposition to analyze the function learned by a deep convolutional network on a vision task. The spectral bias is well understood in linear regression, where the gradient dynamics favours the large singular directions of the feature matrix. For neural networks in the lazy regime, spectral analysis of the neural tangent kernel have been investigated for architectures and data (e.g, uniform data on the sphere) allowing for explicit computations (e.g., decomposition of the NTK in terms of spherical harmonics) (Bietti & Mairal, 2019; Basri et al., 2019; Yang & Salman, 2019). This is a setup where the spectral bias can be rigorously analyzed, i.e., we can get explicit information about the type of functions that are learned quickly and generalize well. In this context, our work specifically focuses on comparing the lazy and the standard feature learning regimes.

Our theoretical model in Section 5.4 reproduces some of the key technical ingredients of known analytical results (Saxe et al., 2014; Gidel et al., 2019) on deep linear networks.

These results show how, in the context of multiclass classification or matrix factorization, the principal components of the input-output correlation matrix are learned sequentially from the highest to the lowest mode. Note however that in the framework of these prior works, the number of modes is bounded by the output dimension - which thus reduces to one in the context of regression or binary classification. By contrast, our theoretical analysis applies to the components of the vector of labels $\boldsymbol{Y} \in \mathbb{R}^n$ in the eigenbasis of the kernel $\boldsymbol{X}\boldsymbol{X}^\top$ defined by the input matrix $\boldsymbol{X} \in \mathbb{R}^{n \times d}$. Thus, despite the technical similarities, our framework approaches the old problem of the relative learning speed of different modes in factorized models from a novel angle. In particular, it allows us to frame the notion of example difficulty in this context.

Example difficulty is a loosely defined concept that has been the subject of intense research recently. Ways to quantify example difficulty for a model/algorithm to learn individual examples include e.g. self-influence (Koh & Liang, 2017), example forgetting (Toneva et al., 2019), TracIn (Pruthi et al., 2020), C-scores (Jiang et al., 2021), or prediction depth (Baldock et al., 2021). We believe that a comprehensive theory of generalization in deep learning will require to understand how neural networks articulate learning and memorization to both fit the head (easy examples) and the tail (difficult examples) of the data distribution (Hooker et al., 2020; Feldman & Zhang, 2020; Sagawa et al., 2020a,b).

## 5.2. Setup

We consider neural networks $f_{\boldsymbol{\theta}}$ parametrized by $\boldsymbol{\theta} \in \mathbb{R}^p$ (i.e. weights and biases for all layers), and trained by minimizing some task-dependent loss function $\ell(\boldsymbol{\theta}) := \sum_{i=1}^n \ell_i(f_{\boldsymbol{\theta}}(\mathbf{x}_i))$ computed on a training dataset $\{\mathbf{x}_1, \cdots, \mathbf{x}_n\}$, using variants of gradient descent,

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta \nabla_{\boldsymbol{\theta}} \ell(\boldsymbol{\theta}^{(t)}), \tag{5.2.1}$$

with some random initialization $\boldsymbol{\theta}^{(0)}$ and a chosen learning rate $\eta > 0$.

**Linearization.** A Taylor expansion and the chain rule give the corresponding updates $f^{(t)} := f_{\boldsymbol{\theta}^{(t)}}$ for any network output, at first order in the learning rate,

$$f^{(t+1)}(\mathbf{x}) \simeq f^{(t)}(\mathbf{x}) - \eta \sum_{i=1}^n K^{(t)}(\mathbf{x}, \mathbf{x}_i) \nabla \ell_i, \tag{5.2.2}$$

which depend on the time-varying **tangent kernel** $K^{(t)}(\mathbf{x}, \mathbf{x}') := \nabla_{\boldsymbol{\theta}} f^{(t)}(\mathbf{x})^\top \nabla_{\boldsymbol{\theta}} f^{(t)}(\mathbf{x}')$. The *lazy regime* is one where this kernel remains nearly constant throughout training. Training the network in this regime thus corresponds to training the linear predictor defined by

$$\bar{f}_{\boldsymbol{\theta}}(\mathbf{x}) := f_{\boldsymbol{\theta}^{(0)}}(\mathbf{x}) + (\boldsymbol{\theta} - \boldsymbol{\theta}^{(0)})^\top \nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}^{(0)}}(\mathbf{x}). \tag{5.2.3}$$

**Modulating linearity.** In our experiments, following Chizat et al. (2019), we modulate the level of "non-linearity" during training of a deep network with a scalar parameter $\alpha \geq 1$, by

replacing our prediction $f_{\boldsymbol{\theta}}$ by

$$f_{\boldsymbol{\theta}}^{\alpha}(\mathbf{x}) := f_{\boldsymbol{\theta}^{(0)}}(\mathbf{x}) + \alpha\left(f_{\boldsymbol{\theta}}(\mathbf{x}) - f_{\boldsymbol{\theta}^{(0)}}(\mathbf{x})\right) \tag{5.2.4}$$

and by rescaling the learning rate as $\eta_\alpha = \eta/\alpha^2$. In this setup, gradient descent steps in parameter space are rescaled by $1/\alpha$ while steps in function space (up to first order) are in $O(1)$ in $\alpha$.[1] $\alpha$ can also be viewed as controlling the *level of feature adaptivity*, where large values of $\alpha$ result in linear training where features are not learned. We will also experiment with $\alpha < 1$ below, which enhances adaptivity compared the standard regime ($\alpha = 1$). The goal of this procedure is two-fold: $(i)$ to be able to smoothly interpolate between the standard regime at $\alpha = 1$ and the linearized one, $(ii)$ to work with models that are nearly linearized yet practically trainable with gradient descent.

**Linearity measures.** In order to assess linearity of training runs for empirically chosen values of the re-scaling factor $\alpha$, we track three different metrics during training (fig 2 right):

- **Sign similarity** counts the proportion of ReLUs in all layers that have kept the same activation status (0 or $> 0$) from initialization.
- **Tangent kernel alignment** measures the similarity between the Gram matrix $\boldsymbol{K}_{ij}^{(t)} = K^{(t)}(\mathbf{x}_i, \mathbf{x}_j)$ of the tangent kernel with its initial value $\boldsymbol{K}^{(0)}$, using kernel alignment (Cristianini et al., 2001)

$$\mathrm{KA}(\boldsymbol{K}^{(t)}, \boldsymbol{K}^{(0)}) = \frac{\mathrm{Tr}[\boldsymbol{K}^{(t)}\boldsymbol{K}^{(0)}]}{\|\boldsymbol{K}^{(t)}\|_F \|\boldsymbol{K}^{(0)}\|_F} \quad (\|\cdot\|_F \text{ is the Froebenius norm}) \tag{5.2.5}$$

- **Representation alignment** measures the similarity of the last non-softmax layer representation $\phi_R^{(t)}(x)$ with its initial value $\phi_R^{(0)}$, in terms of the kernel alignment (eq. 5.2.5) of the corresponding Gram matrices $(\boldsymbol{K}_R^{(t)})_{ij} = \phi_R^{(t)}(\mathbf{x}_i)^\top \phi_R^{(t)}(\mathbf{x}_j)$.

## 5.3. Empirical Study

### 5.3.1. A motivating example on a toy dataset

We first explore the effect of modulating the training regime for a binary classification task on a toy dataset with 2d inputs, for which we can get a visual intuition. We use a fully-connected network with 4 layers and ReLU activations. For 100 independent initial parameter values, we generate 100 training examples uniformly on the square $[-1,1]^2$ from the yin-yang dataset (fig. 1.a). We perform 2 training runs for $\alpha = 1$ and $\alpha = 100$ using (full-batch) gradient descent with learning rate 0.01.

**Global training speed-up and normalization.** After the very first few iterations, we observe a speed-up in training progress of the non-linear regime (fig. 1.b). This is consistent

[1] Under some assumptions such as strong convexity of the loss, it was shown (Chizat et al., 2019, Thm 2.4) that as $\alpha \to \infty$, the gradient descent trajectory of $f_{\boldsymbol{\theta}^{(t)}}^{\alpha}$ gets uniformly close to that of the linearization $\bar{f}_{\boldsymbol{\theta}^{(t)}}$.

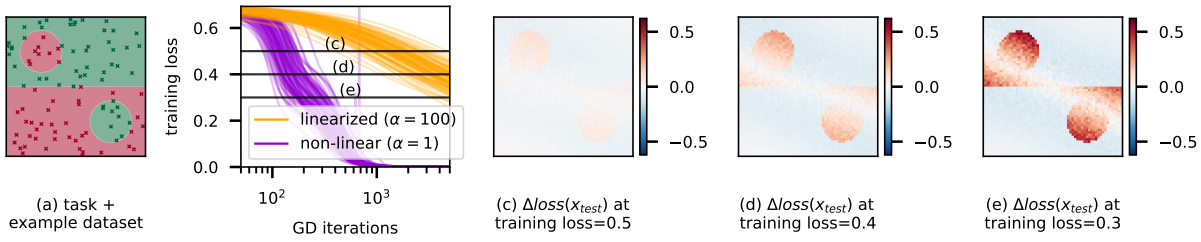



**Figure 1.** 100 randomly initialized runs of a 4 layers MLP trained on the yin-yang dataset **(a)** using gradient descent in both the non-linear ($\alpha = 1$) and linearized ($\alpha = 100$) setting. The training losses **(b)** show a speed-up in the non-linear regime: in order to compare both regimes at equal progress, we normalize by comparing models extracted at equal training loss thresholds **(c)**, **(d)** and **(e)**. We visualize the differences $\Delta\text{loss}\,(x_{\text{test}}) = \text{loss} f_{\text{non-linear}}\,(x_{\text{test}}) - \text{loss} f_{\text{linear}}\,(x_{\text{test}})$ for test points paving the 2d square $[-1,1]^2$ using a color scale. We observe that these differences are not uniformly spread across examples: instead they suggest a comparative bias of the non-linear regime towards correctly classifying easy examples (large areas of the same class), whereas difficult examples (e.g. the small disks) are boosted in the linear regime.

with previously reported numerical experiments on the lazy training regime (Chizat et al., 2019, section 3). This raises the question of whether this acceleration comes from a global scaling in all directions, or if it prioritizes certain particular groups of examples. We address this question by comparing the training dynamics at equal progress: we counteract the difference in training speed by normalizing by the mean training loss, and we compare the linear and non-linear regimes at common thresholds ((c), (d) and (e) horizontal lines in fig. 1.b).

**Comparing linear and non-linear regimes.** At every threshold value, we compute the predictions on test examples uniformly paving the 2d square. We compare both regimes on individual test examples by plotting (fig. 1.c, 1.d, 1.e) the differences in loss values,

$$\Delta\text{loss}\,(x_{\text{test}}) = \text{loss} f_{\text{non-linear}}\,(x_{\text{test}}) - \text{loss} f_{\text{linear}}\,(x_{\text{test}}) \tag{5.3.1}$$

Red (resp. blue) areas indicate a lower test loss for the linearized (resp. non-linear) model. Remarkably, the resulting picture is not uniform: these plots suggest that compared to the linear regime, the non-linear training dynamics speeds up for specific groups of examples (the large top-right and bottom-left areas) at the expense of examples in more intricate areas (both the disks and the areas between the disks and the horizontal boundary).

## 5.3.2. Hastening easy examples

We now experiment with deeper convolutional networks on CIFAR10, in two setups where the training examples are split into groups of varying difficulty. Additional experiments with various other choices of hyperparameters and initialization seed are reported in Appendix C.6.

**5.3.2.1. Example difficulty using C-scores.** In this section we quantify example difficulty using consistency scores (C-scores, Jiang et al., 2021). Informally, C-scores measure how likely an example is to be well-classified by models trained on subsets of the dataset that do not contain it. Intuitively, examples with a high C-score share strong regularities with a large group of examples in the dataset. Formally, given a choice of model $f$, for each (input, label) pair $(x,y)$ in a dataset $\mathcal{D}$, Jiang et al. (2021) defines its empirical consistency profile as:

$$\hat{C}_{\mathcal{D},n}(x,y) = \hat{\mathbb{E}}^{r}_{D \overset{n}{\sim} \mathcal{D}\setminus\{(x,y)\}} \left[ \mathbb{P}\left( f\left(x; D\right) = y \right) \right], \tag{5.3.2}$$

where $\hat{\mathbb{E}}^{r}$ is the empirical average over $r$ subsets $D$ of size $n$ uniformly sampled from $\mathcal{D}$ excluding $(x, y)$, and $f(\cdot, D)$ is the model trained on $D$. A scalar C-score is obtained by averaging the consistency profile over various values of $n = 1, \cdots, |\mathcal{D}| - 1$. For CIFAR10 we use pre-computed scores available as `https://github.com/pluskid/structural-regularity`.

While training, we compute the loss separately on 10 subsets of the training set ranked by increasing C-scores deciles (e.g. examples in the last subset are top-10% C-scores), for both the training set and test set. We also train a linearized copy of this network (so as to share the same initial conditions) with $\alpha = 100$. The $\alpha = 1$ run is trained for 200 epochs (64 000 SGD iterations) whereas in order to converge the $\alpha = 100$ run is trained for 1 000 epochs (320 000 SGD iterations). Similarly to fig. 1, we normalize training progress using the mean training loss in order to compare regimes at equal progress. We check (fig. 2 top right) that the model with $\alpha = 100$ indeed stays in the linear regime during the whole training run, since all 3 linearity metrics that we report essentially remain equal to 1. By contrast, in the non-linear regime ($\alpha = 1$), a steady decrease of linearity metrics as training progresses indicates a rotation of the NTK and the representation kernel, as well as lower sign similarity of the ReLUs.

The results are shown in fig. 2 (top left). As one might expect, examples with high C-scores are learned faster during training than examples with low C-scores in both regimes. Remarkably, this effect is amplified in the non-linear regime compared to the linear one, as we can observe by comparing e.g. the top (resp. bottom) decile in light green (resp dark blue). This illustrates a relative acceleration of the non-linear regime in the direction of easy examples.

**5.3.2.2. Example difficulty using label noise.** In this section we use label noise to define difficult examples. We train a ResNet18 on CIFAR10 where 15% of the training examples are assigned a wrong (random) label. We compute the loss and accuracy independently on the regular examples with their true label, and the noisy examples whose label is flipped. In parallel, we train a copy of the initial model in the linearized regime with $\alpha = 100$. Fig. 2 bottom right shows the 3 linearity metrics during training in both regimes.

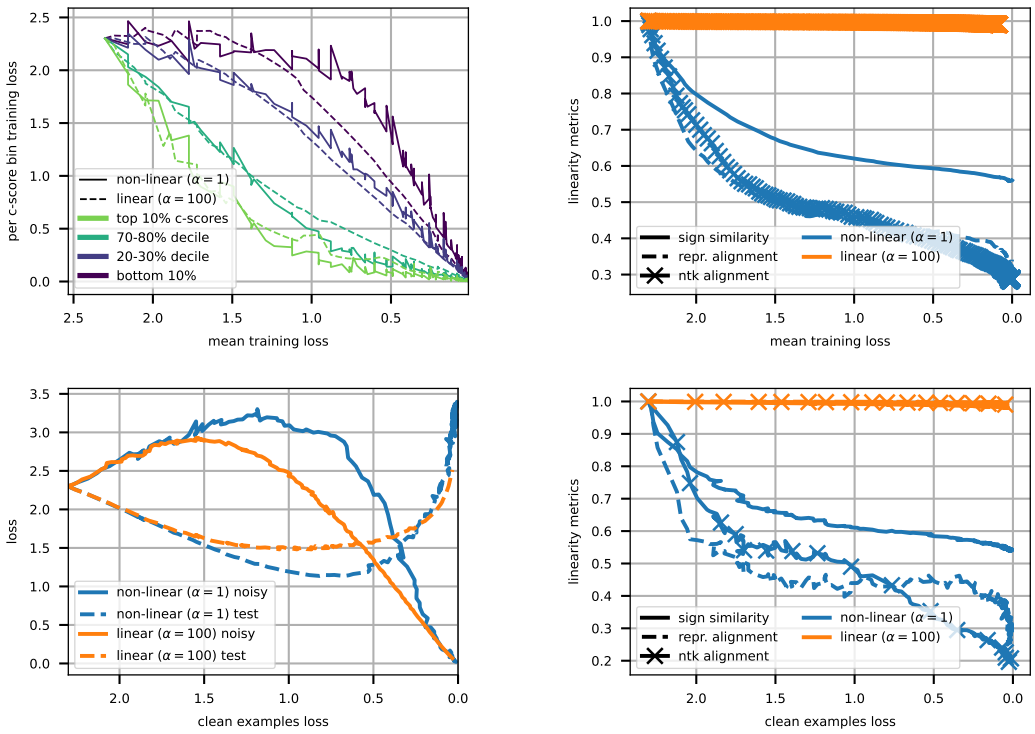



**Figure 2.** Starting from the same initial parameters, we train 2 ResNet18 models with $\alpha = 1$ (standard training) and $\alpha = 100$ (linearized training) on CIFAR10 using SGD with momentum. **(Top left)** We compute the training loss separately on 10 subgroups of examples ranked by their C-scores. Training progress is normalized by the mean training loss on the $x$-axis. Unsurprisingly, in both regimes examples with high C-scores are learned faster. Remarkably, this ranking is more pronounced in the non-linear regime as can be observed by comparing dashed and solid lines of the same color. **(Bottom left)** We randomly flip the class of 15% of the training examples. At equal progress (measured by equal clean examples loss), the non-linear regime prioritizes learning clean examples and nearly ignores noisy examples compared to the linear regime since the solid curve remains higher for the non-linear regime. Concomitantly, the non-linear test loss reaches a lower value. **(Right)** On the same training run, as a sanity check we observe that the $\alpha = 100$ training run remains in the linear regime throughout since all metrics stay close to 1, whereas in the $\alpha = 1$ run, the NTK and representation kernel rotate, and a large part of ReLU signs are flipped. These experiments are completed in Appendix C.6 with accuracy plots for the same experiments, and with other experiments with varying initial model parameters and mini-batch order.

The results are shown in fig. 2 (bottom left). In both regimes, the training process begins with a phase where only examples with true labels are learned, causing the loss on examples with wrong labels to increase. A second phase starts (in this run) at 1.25 clean examples loss for the non-linear regime and 1.5 clean loss for the linear regime, where random labels are getting memorized. We see that the first phase takes a larger part of the training run in the non-linear regime than in the linear regime. We interpret this as the fact that the non-linear regime prioritizes learning easy examples. As a consequence, the majority of the training process is dedicated to learning the clean examples in the non-linear regime, whereas in the

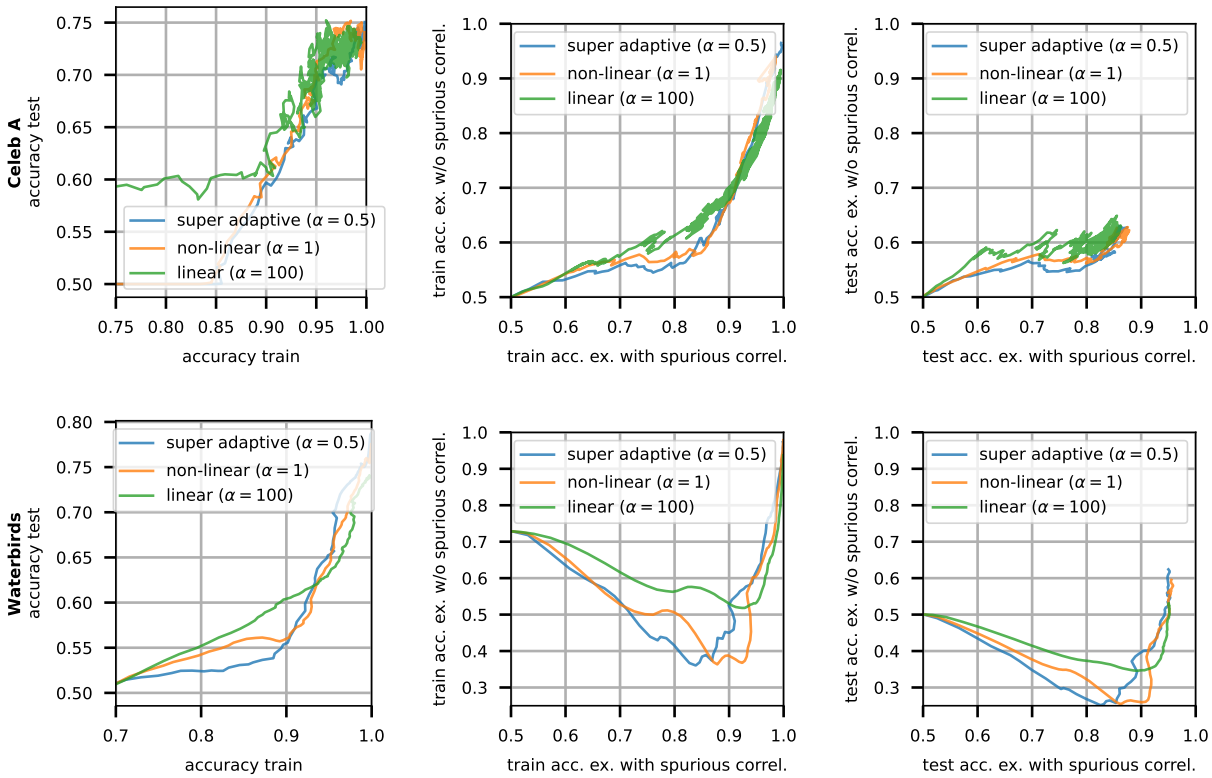



**Figure 3.** We visualize the trajectories of training runs on 2 spurious correlations setups, by computing the accuracy on 2 separate subsets: one with examples that contain the spurious feature (`with spurious`), the other one without spurious correlations (`w/o spurious`). On Celeb A **(top row)**, the attribute 'blond' is spuriously correlated with the gender 'woman'. In the first phase of training we observe that **(left)** the test accuracy is essentially higher for the linear run, which can be further explained by observing that **(middle)** the training accuracy for `w/o spurious` examples increases faster in the linear regime than in non-linear regimes at equal `with spurious` training accuracy. **(right)** A similar trend holds for test examples. In this first part the linear regime is less sensitive to the spurious correlation (easy examples) thus gets better robustness. **(bottom row)** On Waterbirds, the background (e.g. a lake) is spuriously correlated with the label (e.g. a water bird). **(left)** We observe the same hierarchy between the linear run and other runs. In the first training phase, the linear regime is less prone to learning the spurious correlation: the `w/o spurious` accuracy stays higher while the `with spurious` examples are learned (**(middle)** and **(right)**). These experiments are completed in fig. 7 in Appendix C.6 with varying initial model parameters and mini-batch order.

linear regime both the clean and the noisy labels are learned simultaneously passed the 1.5 clean loss mark. Concomitantly, comparing the sweet spot (best test loss) of the two regimes indicates a higher robustness to label noise thus a clear advantage for generalization in the non-linear regime.

### 5.3.3. Spurious correlations

We now examine setups where easy examples are those with strong correlations between their labels and some spurious feature (Sagawa et al., 2020b). We experiment with CelebA (Liu et al., 2015) and Waterbirds (Wah et al., 2011) datasets.

**CelebA.** (Liu et al., 2015) is a collection of photographs of celebrities' faces, each annotated with its attributes, such as hair color or gender. Similarly to Sagawa et al. (2020b), our task is to classify pictures based on whether the person is blond or not. In this dataset, the attribute "the person is a woman" is spuriously correlated with the attribute "the person is blond", since the attribute "blond" is over-represented among women (24% are blond) compared to men (2% are blond).

We use 20 000 examples of CelebA to train a ResNet18 classifier on the task of predicting whether a person is blond or not, using SGD with learning rate 0.01, momentum 0.9 and batch size 100. We also extract a balanced dataset with 180 (i.e. the total number of blond men in the test set) examples in each of 4 categories: blond man, blond woman, non-blond man and non-blond woman. While training progresses, we measure the loss and accuracy on the subgroups man and woman. Starting from the same initial conditions, we train 3 different classifiers with $\alpha \in \{.5, 1, 100\}$.

**Waterbirds.** (Wah et al., 2011) is a smaller dataset of pictures of birds. We reproduce the experiments of Sagawa et al. (2020a) where the task is to distinguish land birds and water birds. In this dataset, the background is spuriously correlated with the type of bird: water birds are typically photographed on a background such as a lake, and similarly there is generally no water in the background of land birds. There are exceptions: a small part of the dataset consists of land birds on a water background and vice versa (e.g. a duck walking in the grass).

We use 4 795 training examples of the Waterbirds dataset. Since the dataset is smaller, we start from a pre-trained ResNet18 classifier from default PyTorch models (pre-trained on ImageNet). We replace the last layer with a freshly initialized binary classification layer, and we set batch norm layers to evaluation mode[2] We train using SGD with learning rate 0.001, momentum 0.9 and minibatch size 100.

From the training set, we extract a balanced dataset with 180 examples in each of 4 groups: land birds on land background, land birds on water background, water bird on land background, and water bird on land background, which we group in two sets: `with spurious` when the type of bird and the background agree, and `w/o spurious` otherwise.

[2]In order not to interfere with $\alpha$ scaling (see section 5.2), and since we consider that batch norm is a whole different theoretical challenge by itself, we chose to turn it off, and instead keep the mean and variance buffers to their pre-trained value. See appendix C.4 for further discussion of this point.

While training, we measure the accuracy separately on these 2 sets. We train 3 different classifiers with $\alpha \in \{.5,1,100\}$.

Looking at fig. 3 left, for both the Waterbirds and the CelebA experiments we identify two phases: in the first phase the test accuracy is higher for the linear $\alpha = 100$ run than for other runs. In the second phase all 3 runs seem to converge, with a slight advantage for non-linear runs in Waterbirds. Taking a closer look at the first phase in fig. 3 middle and right, we understand this difference in test accuracy in light of spurious and non-spurious features: in the non-linear regime, the training dynamics learns the majority examples faster, at the cost of being more prone to spurious correlations. This can be seen both on the balanced training set and the balanced test set.

## 5.4. Theoretical Insights

The goal here is to illustrate some of our insights in a simple setup amenable to analytical treatment.

### 5.4.1. A simple quadratic model

We consider a standard linear regression analysis: given $n$ input vectors $\mathbf{x}_i \in \mathbb{R}^d$ with their corresponding labels $y_i \in \mathbb{R}$, $1 \leq i \leq n$, the goal is to fit a linear function $f_{\boldsymbol{\theta}}(\mathbf{x})$ to the data by minimizing the least-squares loss $\ell(\boldsymbol{\theta}) := \frac{1}{2}\sum_i (f_{\boldsymbol{\theta}}(\mathbf{x}_i) - y_i)^2$. We will focus on a specific quadratic parametrization, which can be viewed as a subclass of two-layer networks with linear activations.

**Notation.** We denote by $\boldsymbol{X} \in \mathbb{R}^{n\times d}$ the matrix of inputs and by $\boldsymbol{y} \in \mathbb{R}^n$ the vector of labels. We consider the singular value decomposition (SVD),

$$\boldsymbol{X} = \boldsymbol{U}\boldsymbol{M}\boldsymbol{V}^\top := \sum_{\lambda=1}^{r_X} \sqrt{\mu_\lambda}\boldsymbol{u}_\lambda \boldsymbol{v}_\lambda^\top \tag{5.4.1}$$

where $\boldsymbol{U} \in \mathbb{R}^{n\times n}, \boldsymbol{V} \in \mathbb{R}^{d\times d}$ are orthogonal and $\boldsymbol{M}$ is rectangular diagonal. $r_X$ denotes the rank of $\boldsymbol{X}$, $\mu_1 \geq \cdots \geq \mu_{r_X} > 0$ are the non zero (squared) singular values; we also set $\mu_\lambda = 0$ for $r_X < \lambda \leq \max(n,d)$. Left and right singular vectors extend to orthonormal bases $(\boldsymbol{u}_1, \cdots, \boldsymbol{u}_n)$ and $(\boldsymbol{v}_1, \cdots, \boldsymbol{v}_d)$ of $\mathbb{R}^n$ and $\mathbb{R}^d$, respectively. In what follows we assume, without loss of generality[3], that the vector of labels has positive components in the basis $\boldsymbol{u}_\lambda$, i.e., $y_\lambda := \boldsymbol{u}_\lambda^\top \boldsymbol{y} \geq 0$ for $\lambda = 1, \cdots n$.

**Parametrization.** We consider the following class of functions

$$f_{\boldsymbol{\theta}}(\mathbf{x}) = \boldsymbol{\theta}^\top \mathbf{x}, \quad \boldsymbol{\theta} := \frac{1}{2}\sum_{\lambda=1}^{d} w_\lambda^2 \, \boldsymbol{v}_\lambda \tag{5.4.2}$$

[3]One can use the reflection invariance $\sqrt{\mu_\lambda}\boldsymbol{u}_\lambda\boldsymbol{v}_\lambda^\top = \sqrt{\mu_\lambda}(-\boldsymbol{u}_\lambda)(-\boldsymbol{v}_\lambda)^\top$ of the SVD to flip the sign of $y_\lambda$.

In this setting, the least-squares loss is minimized by gradient descent over the vector parameter $\mathbf{w} = [w_1, \cdots w_d]^\top$. Given an initialization $\mathbf{w}^0$, we want to compare the solution of the vanilla gradient (non linear) dynamics with the solution of the lazy regime, which corresponds to training the linearized function (5.2.3), given in our setting by $\bar{f}_{\bar{\boldsymbol{\theta}}}(\mathbf{x}) = \bar{\boldsymbol{\theta}}^\top \mathbf{x}$ where $\bar{\boldsymbol{\theta}} = \sum_{\lambda=1}^d w_\lambda w_\lambda^0 \boldsymbol{v}_\lambda$.

## 5.4.2. Gradient dynamics

For simplicity, we analyze the continuous-time counterpart of gradient descent,

$$\mathbf{w}(0) = \mathbf{w}^0, \quad \dot{\mathbf{w}}(t) = -\nabla_{\mathbf{w}} \ell(\boldsymbol{\theta}(t)) \tag{5.4.3}$$

where the dot denotes the time-derivative. Making the gradients explicit and differentiating (5.4.2) yield

$$\dot{\boldsymbol{\theta}}(t) = -\boldsymbol{\Sigma}(t)(\boldsymbol{\theta}(t) - \boldsymbol{\theta}^*), \quad \boldsymbol{\Sigma}(t) = \boldsymbol{V}\mathrm{Diag}(\mu_1 w_1^2(t), \cdots, \mu_d w_d^2(t))\boldsymbol{V}^\top \tag{5.4.4}$$

where $\boldsymbol{\theta}^*$ is the solution of the linear dynamics.[4] Note how $\boldsymbol{\Sigma}(t)$ is obtained from the input correlation matrix $\boldsymbol{X}^\top \boldsymbol{X}$ by rescaling each eigenvalue $\mu_\lambda$ by the time-varying factor $w_\lambda^2$. By contrast, the lazy regime is described by the equation obtained from (5.4.4) by replacing $\boldsymbol{\Sigma}(t)$ the constant matrix $\boldsymbol{\Sigma}(0)$.

In the proposition below (proved in Appendix C.1), we consider the system (5.4.3, 5.4.4) initialized as $\boldsymbol{\theta}^0 := \frac{1}{2}\sum_{\lambda=1}^d (w_\lambda^0)^2 \boldsymbol{v}_\lambda$ where we assume that $w_\lambda^0 \neq 0$ for all $\lambda$. We denote by $\tilde{y}_\lambda$ the components of the input-label correlation vector $\boldsymbol{X}^\top \boldsymbol{y} \in \mathbb{R}^d$ in the basis $\boldsymbol{v}_\lambda$: we have $\tilde{y}_\lambda = \sqrt{\mu_\lambda} y_\lambda$ for $1 \leq \lambda \leq r_X$ and 0 when $r_X < \lambda \leq d$.

**Proposition 5.4.1.** *The solution of (5.4.3, 5.4.4) is given by,*

$$\boldsymbol{\theta}(t) = \sum_{\lambda=1}^d \theta_\lambda(t) \boldsymbol{v}_\lambda, \qquad \theta_\lambda(t) = \begin{cases} \dfrac{\theta_\lambda^0 \theta_\lambda^*}{\theta_\lambda^0 - e^{-2\tilde{y}_\lambda t}(\theta_\lambda^0 - \theta_\lambda^*)} & \text{if } \theta_\lambda^* \neq 0 \\ \dfrac{\theta_\lambda^0}{1 + 2\mu_\lambda \theta_\lambda^0 t} & \text{if } \theta_\lambda^* = 0 \end{cases} \tag{5.4.5}$$

*By contrast, the solution in the linearized regime where $\boldsymbol{\Sigma}(t) \approx \boldsymbol{\Sigma}(0)$ is,*

$$\theta_\lambda(t) = \theta_\lambda^* + e^{-\mu_\lambda \theta_\lambda^0 t}(\theta_\lambda^0 - \theta_\lambda^*) \tag{5.4.6}$$

## 5.4.3. Discussion

While the gradient dynamics (5.4.5,5.4.6) converge to the same solution $\boldsymbol{\theta}^*$, we see that the convergence rates of the various modes differ in the two regimes. To quantify this, for each dynamical mode $1 \leq \lambda \leq r_X$ such that $|\theta_\lambda^*| \neq 0$, let $t_\lambda(\epsilon), t_\lambda^{\mathrm{lin}}(\epsilon)$ be the times required

[4] Explicitly, $\boldsymbol{\theta}^* = \boldsymbol{\Sigma}^+ \boldsymbol{X}^\top \boldsymbol{y} + P_\perp(\boldsymbol{\theta}^0)$ where $\boldsymbol{\Sigma}^+$ is the pseudoinverse of the input correlation matrix $\boldsymbol{\Sigma} = \boldsymbol{X}^\top \boldsymbol{X}$ and $P_\perp$ projects onto the null space of $\boldsymbol{X}$. It decomposes as $\boldsymbol{\theta}^* = \sum_{\lambda=1}^d \theta_\lambda^* \boldsymbol{v}_\lambda$ in the basis $\boldsymbol{v}_\lambda$, where $\theta_\lambda^* = y_\lambda / \sqrt{\mu_\lambda}$ for $1 \leq \lambda \leq r_X$ and $\theta_\lambda^* = \theta_\lambda^0$ for $r_X < \lambda \leq d$.

for $\theta_\lambda$ to be $\epsilon$-close to convergence, i.e $|\theta_\lambda - \theta_\lambda^*| = \epsilon$, in the two regimes. Substituting into (5.4.5) and (5.4.6) we find that, close to convergence $\epsilon \ll \theta_\lambda^*$ and for a small initialization, $\theta_\lambda^0 \ll \theta_\lambda^*$,

$$t_\lambda(\epsilon) = \frac{1}{\tilde{y}_\lambda} \log \frac{\theta_\lambda^*}{\epsilon \theta_\lambda^0}, \qquad t^{\rm lin}_\lambda(\epsilon) = \frac{1}{\mu_\lambda \theta_\lambda^0} \log \frac{\theta_\lambda^*}{\epsilon} \tag{5.4.7}$$

up to terms in $O(\epsilon/\theta_\lambda^*, \theta_\lambda^0/\theta_\lambda^*)$. Two remarks are in order:

**Linearization impacts learning schedule.** Specifically, while the learning speed of each mode depends on the components of the label vector in the non linear regime, through $\tilde{y}_\lambda = \sqrt{\mu_\lambda} y_\lambda$, it does not in the linearized one. Thus, the ratio of $\epsilon$-convergence times for two given modes $\lambda, \lambda'$ is $t_\lambda / t_{\lambda'} = \tilde{y}_\lambda / \tilde{y}_{\lambda'}$ in the non-linear case and $t^{\rm lin}_\lambda / t^{\rm lin}_\lambda = \mu_{\lambda'} \theta^0_{\lambda'} / \mu_\lambda \theta^0_\lambda$ in the linearized case, up to logarithmic factors.

**Sequentialization of learning.** Non-linearity, together with a vanishingly small initialization, induces a sequentialization of learning for the various modes (see also Gidel et al., 2019, Thm 2). To see this, pick a mode $\lambda$ and consider, for any other mode $\lambda'$, the value $\theta_{\lambda'}(t_\lambda)$ at the time $t_\lambda := t_\lambda(\epsilon)$ where $\lambda$ reaches $\epsilon$-convergence. Let us also write $\theta_\lambda^0 = \sigma \tilde{\theta}_\lambda^0$ where $\sigma$ is small and $\tilde{\theta}_\lambda^0 = O(1)$ as $\sigma \to 0$. Then elementary manipulations show the following:

$$\theta_{\lambda'}(t_\lambda) = \frac{\theta^*_{\lambda'} \tilde{\theta}^0_{\lambda'}}{\tilde{\theta}^0_{\lambda'} + \left[\epsilon \tilde{\theta}^0_{\lambda'} / \theta^*_{\lambda'}\right]^{2\frac{\tilde{y}_{\lambda'}}{\tilde{y}_\lambda}} \sigma^{2\frac{\tilde{y}_{\lambda'}}{\tilde{y}_\lambda} - 1}} =_{\sigma \to 0} \begin{cases} \theta^*_{\lambda'} & \tilde{y}_{\lambda'} > \tilde{y}_\lambda \\ 0 & \tilde{y}_{\lambda'} < \tilde{y}_\lambda \end{cases} \tag{5.4.8}$$

In words, for fixed $\epsilon > 0$ and in the limit of small initialization, the mode $\lambda$ gets $\epsilon$-close to convergence before any of the subdominant mode deviates from their (vanishing) initial value: the modes are learned sequentially.

## 5.4.4. Mode vs example difficulty

We close this section by illustrating the link between *mode* and *example difficulty* on three concrete examples of structure for the training data $\{\mathbf{x}_i, y_i\}_{i=1}^n$. In what follows, we consider the overparametrized setting where $d \geq n$. We denote by $\boldsymbol{e}_1, \cdots \boldsymbol{e}_d$ the canonical basis of $\mathbb{R}^d$.

For each of the examples below, we consider an initialization $\mathbf{w}^0 = \sqrt{\sigma}[1 \cdots 1]^\top$ with small $\sigma > 0$ and the corresponding solutions in Prop. 5.4.1 for both regimes.

**Example 1.** We begin with a rather trivial setup where each mode corresponds to a training example. It will illustrate how non-linearity can *reverse* the learning order of the examples. Given a sequence of strictly positive numbers $\mu_1 \geq \cdots \mu_n > 0$, we consider the training data,

$$\mathbf{x}_i = \sqrt{\mu_i} \boldsymbol{e}_i, \quad y_i = \mu_{n-i+1} / \sqrt{\mu_i}, \qquad 1 \leq i \leq n \tag{5.4.9}$$

In the linearized regime, $f_{\boldsymbol{\theta}(t)}(\mathbf{x}_i)$ converges to $y_i$ at the linear rate $\sigma \mu_i$; the model learns faster the examples with higher $\mu_i$, hence with *lower* index $i$. In the non linear regime, the examples are learned sequentially according to the value $\tilde{y}_i = \sqrt{\mu_i} y_i = \mu_{n-i+1}$, hence from

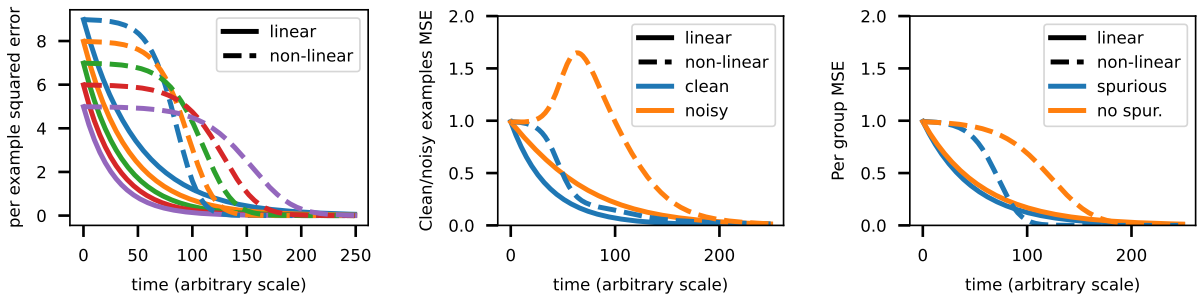



**Figure 4.** **(left)** Different input/label correlation (example 1): examples are learned in a flipped order in the two regimes. **(middle)** Label noise (example 2): the non-linear dynamics prioritizes learning the clean labels **(right)** Spurious correlations (example 3): the non-linear dynamics prioritizes learning the spuriously correlated feature. These analytical curves are completed with numerical experiments on standard (dense) 2-layer MLP in figure 9 in appendix C.7, which shows a similar qualitative behaviour.

*high to low* index $i$. Thus in this setting, linearization flips the learning order of the training examples (see fig. 4 left).

In examples 2 and 3 below we aim at modelling situations where the labels depend on low-dimensional (in this case, one dimensional) projections on the inputs (e.g. an image classification problem where the labels mainly depend on the low frequencies of the image). These two examples mirror the two sets of experiments in Sections 5.3.2.2 and 5.3.3, respectively.

**Example 2.** We consider a simple classification setup on linearly separable data with label noise. Here we assume $d > n$. Conditioned on a set of binary labels $y_i = \pm 1$, the inputs are given by

$$\mathbf{x}_i = \kappa_i y_i \boldsymbol{e}_1 + \eta \boldsymbol{e}_{i+1} \quad 1 \leq i \leq n \tag{5.4.10}$$

where $\kappa_i = \pm 1$ is some 'label flip' variable and $\eta > 0$. We assume we have $q$ 'noisy' examples with flipped labels, i.e. $\kappa_i = -1$, where $1 \leq q < \lceil n/2 \rceil$.

The SVD of the feature matrix $\boldsymbol{X} \in \mathbb{R}^{n \times d}$ defined by (5.4.10) can be made explicit. In particular, the top left singular vector is $\boldsymbol{u}_1 = \bar{\boldsymbol{y}}/\sqrt{n}$, where $\bar{\boldsymbol{y}} \in \mathbb{R}^n$ denotes the vector of noisy labels $\bar{y}_i := \kappa_i y_i$. This singles out a dominant mode $\boldsymbol{y}_1 := (\boldsymbol{u}_1^\top \boldsymbol{y})\boldsymbol{u}_1$ of the label vector $\boldsymbol{y}$ that is *learned first* by the non-linear dynamics. Explicitly,

$$\boldsymbol{y}_1 = (1 - \frac{2q}{n})\bar{\boldsymbol{y}} \tag{5.4.11}$$

For a small noise ratio $q/n \ll 1$, this yields $y_{1i} \approx y_i$ for clean examples and $-y_i$ for noisy ones: fitting the dominant mode amounts to learning the clean examples – while assigning the wrong label to the noisy ones. This is illustrated in fig 4 (middle plot).

**Example 3.** We consider a simple spurious correlation setup (Sagawa et al., 2020b), obtained by adding to (5.4.10) a 'core' feature that separates all training points. Here we assume

$d > n + 1$. Conditioned on a set of binary labels $y_i = \pm 1$, the inputs are given by

$$\mathbf{x}_i = \kappa_i y_i \boldsymbol{e}_1 + \lambda y_i \boldsymbol{e}_2 + \eta \boldsymbol{e}_{i+2}, \quad 1 \leq i \leq n \tag{5.4.12}$$

for some binary variable $\kappa_i = \pm 1$, scaling factor $\lambda \in (0, 1)$, and $\eta > 0$. Given $1 \leq q < \lceil n/2 \rceil$, we assume we have a majority group of $n - q$ training examples with $\kappa_i = 1$, whose label is spuriously correlated with the spurious feature $\boldsymbol{e}_2$, and a minority group of $q$ examples with $\kappa_i = -1$.

The analysis is similar as in Example 2. For small noise ratio and scaling factor $\lambda$, the non-linear dynamics enhances the bias towards fitting first the majority group of ('easy') examples with spuriously correlated labels. This illustrates an increased sensitivity of the non linear regime to the spurious feature – at least in the first part of training. This is shown in fig 4 (right plot).

## 5.5. Conclusion

The recent emphasis on the lazy training regime, where deep networks behave as linear models amenable to analytical treatment, begs the question of the specific mechanisms and implicit biases which differentiates it from the full-fledged feature learning regime of the algorithms used in practice. In this paper, we investigated the comparative effect of the two regimes on subgroups of examples based on their difficulty. We provided experiments in various setups suggesting that easy examples are given more weight in non-linear training mode (deep learning) than in linear training, resulting in a comparatively higher learning speed for these examples. We illustrated this phenomenon across various ways to quantify examples difficulty, through C-scores, label noise, and correlations to some easy-to-learn spurious features. We complemented these empirical observations with a theoretical analysis of a quadratic model whose training dynamics is tractable in both regimes. We believe that our findings makes a step towards a better understanding of the underlying mechanisms that drive the good generalization properties observed in practice.

This concludes our comparison of the lazy and feature learning regimes. In the next chapter, we formalize a dynamical mechanism of rotation and stretching of the NTK during training. This mechanism sheds light on the 'how' does the feature learning regime prioritize certain groups of examples.

# Chapter 6

# Implicit Regularization via Neural Feature Alignment

## Prologue

In this chapter, we study how the finite-width neural tangent kernel (NTK) varies during training. We discover an implicit regularization mechanism given by an increased alignment of the NTK to the target kernel as training progresses. We exploit this alignment by describing a Rademacher complexity bound based on a family of functions stemming from a sequence of increasingly aligned kernels. We empirically show the relevance of this bound at capturing the generalization gap of deep networks while we vary the number of parameters, or the proportion of corrupted labels in the training set.

***Article Details.*** *Implicit Regularization via Neural Feature Alignment.* Aristide Baratin*, Thomas George*, César Laurent, R Devon Hjelm, Guillaume Lajoie, Pascal Vincent, Simon Lacoste-Julien. The paper has been published at *AISTATS 2021* (Baratin et al., 2021).

* Co-first authors.

**Context.** The genesis of the project came from a previous paper about the spectral bias of gradient descent on deep networks (Rahaman et al., 2019), where it is empirically shown on synthetic data that Fourier modes are learned sequentially from low frequencies to high frequencies. At the same time, the NTK paper (Jacot et al., 2018) popularized the idea of importing ideas from kernel methods to deep learning theory. A natural idea was to explore this spectral bias using modes of the NTK instead of Fourier modes, and we indeed observed that the dominant modes are aligned (see Figure 2 in Appendix B). Concurrently, other authors also analyzed linear regimes (Yang & Salman, 2019; Bietti & Mairal, 2019) and identified generalization properties in synthetic data setups.

The actual training regime of deep networks is however not linear. Locally to each iterate, it can nonetheless be linearized using the first order Taylor expansion of the function with respect to the parameters. This defines tangent features that vary throughout the course of training, and a local, time-varying NTK. We empirically analyzed this NTK as

training progresses, and report 2 main insights: i) the effective rank of the NTK diminishes, concentrating on a few modes with large eigenvalues, and ii) the alignment between the NTK and the target kernel increases. i) can be interpreted as a compression mechanism that locally biases the training dynamics towards a few directions in function space. ii) is a popular criterion for kernel learning (Cortes et al., 2012), since aligned kernels tend to have better generalization properties (Cristianini et al., 2001).

We interpret this alignment as an implicit bias (Neyshabur et al., 2014) that effectively reduces the capacity of families of trained neural networks. By considering the whole optimization trajectory as a series of steps made using a sequence of increasingly aligned kernels, we propose a Rademacher complexity measure for such families of functions.

These alignment and complexity measure are empirically evaluated in different settings, including in the presence of label noise, or by training networks with varying number of parameters. This sensitivity analyses show that the alignment mechanism is consistent across setups, and that the complexity measure correlates with the generalization gap.

**Developments.** Concurrently to our work, following publication of the NTK paper of Jacot et al. (2018), other teams studied the properties of the time varying tangent kernel. Fort et al. (2020) empirically showed that the evolution of the NTK is very rapid in a first phase of training, then it is mostly frozen. Learning a kernelized linear model using the NTK extracted late in training (coined the 'after kernel' in Long (2021)) gives better generalization than the initial NTK. Atanasov et al. (2021) and Shan & Bordelon (2021) study the alignment effect in deep linear networks.

Finally, in the paper presented in Chapter 5, we continue our exploration by looking more closely at which examples are given more weight by the deep learning training regime. The mechanism that is at play can be understood in the words of the current paper, by saying that the NTK aligns preferably to the groups of easy examples that we have identified.

**Personal Contribution.** Aristide contributed the core idea of looking at the evolution of the NTK during training, and the Rademacher complexity bound. I contributed to most of the experiments, including some that did not end up appearing in the paper. César helped with experiments. Devon, Guillaume, Pascal and Simon contributed to the project through discussions and supervision.

## 6.1. Introduction

One important property of deep neural networks is their ability to generalize well on real data. Surprisingly, this is even true with very high-capacity networks *without explicit regularization* (Neyshabur et al., 2014; Zhang et al., 2017a; Hoffer et al., 2017). This seems at odds with the usual understanding of the bias-variance trade-off (Geman et al., 1992; Neal et al., 2019; Belkin et al., 2019): highly complex models are expected to overfit the training

data and perform poorly on test data (Hastie et al., 2009). Solving this apparent paradox requires understanding the various learning biases induced by the training procedure, which can act as implicit regularizers (Neyshabur et al., 2014, 2017b).

In this paper, we help clarify one such implicit regularization mechanism, by examining the evolution of the *neural tangent features* (Jacot et al., 2018) learned by the network along the optimization paths. Our results can be understood from two complementary perspectives: a *geometric* perspective – the (uncentered) covariance of the tangent features defines a metric on the function class, akin to the Fisher information metric (e.g., Amari, 2016); and a *functional* perspective – through the tangent kernel and its RKHS. In standard supervised classification settings, our main observation is a dynamical alignment of the tangent features along a small number of task-relevant directions during training. We interpret this phenomenon as a combined mechanism of *feature selection* and *compression*. The intuition motivating this work is that such a mechanism allows large models to adapt their capacity to the task, which in turn underpins their generalization abilities.

Specifically, our main contributions are as follows:

(1) Through experiments with various architectures on MNIST and CIFAR10, we give empirical insights on how the tangent features and their kernel adapt to the task during training (Section 6.3). We observe in particular a sharp increase of the anisotropy of their spectrum early in training, as well as an increasing similarity with the class labels, as measured by *centered kernel alignment* (Cortes et al., 2012).

(2) Drawing upon intuitions from linear models (Section 6.4.1), we argue that such a dynamical alignment acts as *implicit regularizer*. We motivate a new heuristic complexity measure which captures this phenomenon, and empirically show better correlation with generalization compared to various measures proposed in the recent literature (Section 6.4).

## 6.2. Preliminaries

Let $\mathcal{F}$ be a class of functions (e.g a neural network) parametrized by $\mathbf{w} \in \mathbb{R}^P$. We restrict here to *scalar* functions $f_{\mathbf{w}} \colon \mathcal{X} \to \mathbb{R}$ to keep notation light.[1]

**Tangent Features.** We define the **tangent features** as the function gradients w.r.t the parameters,

$$\Phi_{\mathbf{w}}(\mathbf{x}) := \nabla_{\mathbf{w}} f_{\mathbf{w}}(\mathbf{x}) \in \mathbb{R}^P. \tag{6.2.1}$$

The corresponding kernel $k_{\mathbf{w}}(\mathbf{x}, \tilde{\mathbf{x}}) = \langle \Phi_{\mathbf{w}}(\mathbf{x}), \Phi_{\mathbf{w}}(\tilde{\mathbf{x}}) \rangle$ is the **tangent kernel** (Jacot et al., 2018). Intuitively, the tangent features govern how small changes in parameter affect the

[1]The extension to vector-valued functions, relevant for the multiclass classification setting, is presented in Appendix B.1, along with more mathematical details.

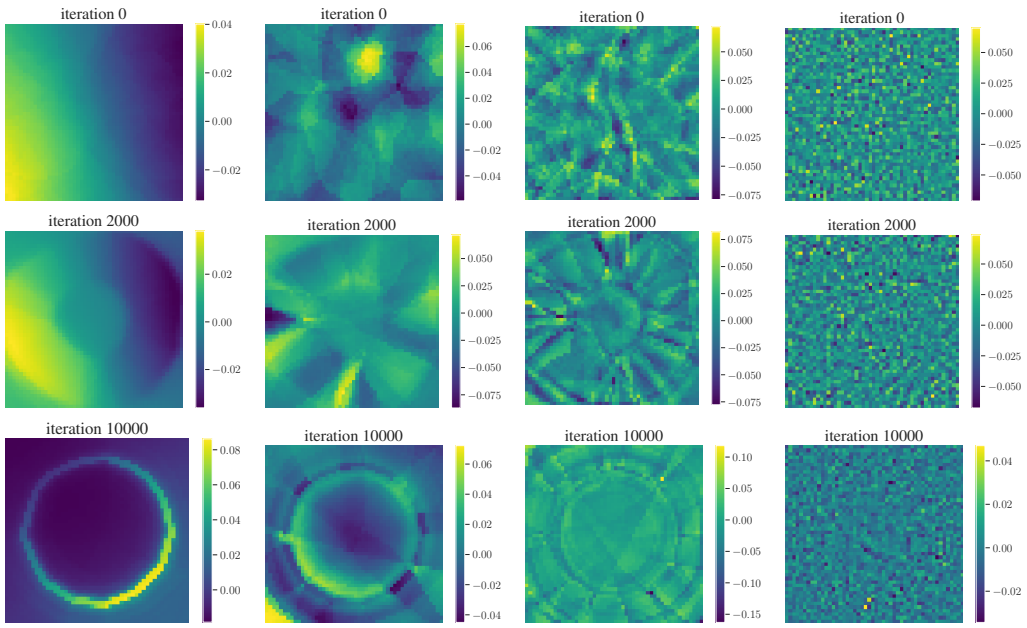



**Figure 1.** Evolution of eigenfunctions of the tangent kernel, ranked in nonincreasing order of the eigenvalues (**in columns**), at various iterations during training (**in rows**), for the $2d$ Disk dataset. After a number of iterations, we observe modes corresponding to the class structure (e.g. boundary circle) in the top eigenfunctions. Combined with an increasing anistropy of the spectrum (e.g $\lambda_{20}/\lambda_1 = 1.5\%$ at iteration 0, 0.2% at iteration 2000), this illustrates a stretch of the tangent kernel, hence a (soft) compression of the model, along a small number of features that are highly correlated with the classes.

function's outputs,

$$\delta f_{\mathbf{w}}(\mathbf{x}) = \langle \delta \mathbf{w}, \Phi_{\mathbf{w}}(\mathbf{x}) \rangle + O(\|\delta \mathbf{w}\|^2). \tag{6.2.2}$$

More formally, the (uncentered) covariance matrix $g_{\mathbf{w}} = \mathbb{E}_{\mathbf{x}\sim\rho}\left[\Phi_{\mathbf{w}}(\mathbf{x})\Phi_{\mathbf{w}}(\mathbf{x})^\top\right]$ w.r.t the input distribution $\rho$ acts as a **metric tensor** on $\mathcal{F}$: assuming $\mathcal{F} \subset L^2(\rho)$, this is the metric induced on $\mathcal{F}$ by pullback of the $L^2$ scalar product. It characterizes the geometry of the function class $\mathcal{F}$. Metric (as symmetric $P \times P$ matrices) and tangent kernels (as rank $P$ integral operators) share the same spectrum (see Prop B.1.1 in Appendix B.1.3).

**Spectral Bias.** The structure of the tangent features impacts the evolution of the function during training. To formalize this, we introduce the covariance eigenvalue decomposition $g_{\mathbf{w}} = \sum_{j=1}^P \lambda_{\mathbf{w}j} \boldsymbol{v}_{\mathbf{w}j} \boldsymbol{v}_{\mathbf{w}j}^\top$, which summarizes the predominant directions in parameter space. Given $n$ input samples $(\mathbf{x}_i)$ and $\boldsymbol{f}_{\mathbf{w}} \in \mathbb{R}^n$ the vector of outputs $f_{\mathbf{w}}(\mathbf{x}_i)$, consider gradient descent updates $\delta \mathbf{w}_{\text{GD}} = -\eta \nabla_{\mathbf{w}} L$ for some cost function $L := L(\boldsymbol{f}_{\mathbf{w}})$. The following elementary result (see Appendix B.1.5) shows how the corresponding function updates in

the linear approximation (6.2.2), $\delta f_{\text{GD}}(\mathbf{x}) := \langle \delta \mathbf{w}_{\text{GD}}, \Phi_{\mathbf{w}}(\mathbf{x}) \rangle$, decompose in the **eigenbasis**[2] of the tangent kernel:

$$u_{\mathbf{w}j}(\mathbf{x}) = \frac{1}{\sqrt{\lambda_{\mathbf{w}j}}} \langle \boldsymbol{v}_{\mathbf{w}j}, \Phi_{\mathbf{w}}(\mathbf{x}) \rangle \tag{6.2.3}$$

**Lemma 6.2.1** (Local Spectral Bias)**.** *The function updates decompose as* $\delta f_{GD}(\mathbf{x}) = \sum_{j=1}^{P} \delta f_j u_{\mathbf{w}j}(\mathbf{x})$ *with*

$$\delta f_j = -\eta \lambda_{\mathbf{w}j} (\boldsymbol{u}_{\mathbf{w}j}^\top \nabla_{\boldsymbol{f}_{\mathbf{w}}} L), \tag{6.2.4}$$

*where* $\boldsymbol{u}_{\mathbf{w}j} = [u_{\mathbf{w}j}(\mathbf{x}_1), \cdots u_{\mathbf{w}j}(\mathbf{x}_n)]^\top \in \mathbb{R}^n$ *and* $\nabla_{\boldsymbol{f}_{\mathbf{w}}}$ *denotes the gradient w.r.t the sample outputs.*

This illustrates how, from the point of view of function space, the metric/tangent kernel eigenvalues act as a mode-specific rescaling $\eta \lambda_{\mathbf{w}j}$ of the learning rate.[3] This is a local version of a well-known bias for linear models trained by gradient descent (e.g in linear regression, see Appendix B.1.5.2), which prioritizes learning functions within the top eigenspaces of the kernel. Several recent works (Bietti & Mairal, 2019; Basri et al., 2019; Yang & Salman, 2019) investigated such bias for neural networks, in *linearized* regimes where the tangent kernel remains constant during training (Jacot et al., 2018; Du et al., 2019b; Allen-Zhu et al., 2019). As a simple example, for a randomly initialized MLP on 1D uniform data, Fig. 2 in Appendix B.1.5 shows an alignment of the tangent kernel eigenfunctions with Fourier modes of increasing frequency, in line with prior empirical observations (Rahaman et al., 2019; Xu et al., 2019) of a 'spectral bias' towards low-frequency functions.

**Tangent Features Adapt to the Task.** By contrast, our aim in this paper is to highlight and discuss *non-linear* effects, in the (standard) regime where the tangent features and their kernel evolve during training (e.g., Geiger et al., 2020; Woodworth et al., 2020).

As a first illustration of such effects, Fig. 1 shows visualizations of eigenfunctions of the tangent kernel (ranked in nonincreasing order of the eigenvalues), during training of a 6-layer deep 256-unit wide MLP by gradient descent of the binary cross entropy loss, on a simple classification task: $y(\mathbf{x}) = \pm 1$ depending on whether $\mathbf{x} \sim \text{Unif}[-1,1]^2$ is in the centered disk of radius $\sqrt{2/\pi}$ (details in Appendix B.3.1). After a number of iterations, we observe (rotation invariant) modes corresponding to the class structure (e.g. boundary circle) showing up in the *top* eigenfunctions of the learned kernel. We also note an increasing spectrum anisotropy – for example, the ratio $\lambda_{20}/\lambda_1$, which is 1.5% at iteration 0, has dropped to 0.2% at iteration 2000. The interpretation is that the tangent kernel (and the metric) *stretch* along a relatively small number of directions that are highly correlated with the classes during training. We quantify and investigate this effect in more detail below.

[2]The functions $(u_{\mathbf{w}j})_{j=1}^P$ form an orthonormal family in $L^2(\rho)$, i.e. $\mathbb{E}_{\mathbf{x}\sim\rho}[u_{\mathbf{w}j} u_{\mathbf{w}j'}] = \delta_{jj'}$, and yield the spectral decomposition $k_{\mathbf{w}}(\mathbf{x}, \tilde{\mathbf{x}}) = \sum_{j=1}^P \lambda_{\mathbf{w}j} u_{\mathbf{w}j}(\mathbf{x}) u_{\mathbf{w}j}(\tilde{\mathbf{x}})$ of the tangent kernel as an integral operator (see Appendix B.1.3).

[3]Intuitively, the eigenvalue $\lambda_{\mathbf{w}j}$ can be thought of as defining a local 'learning speed' for the mode $j$.

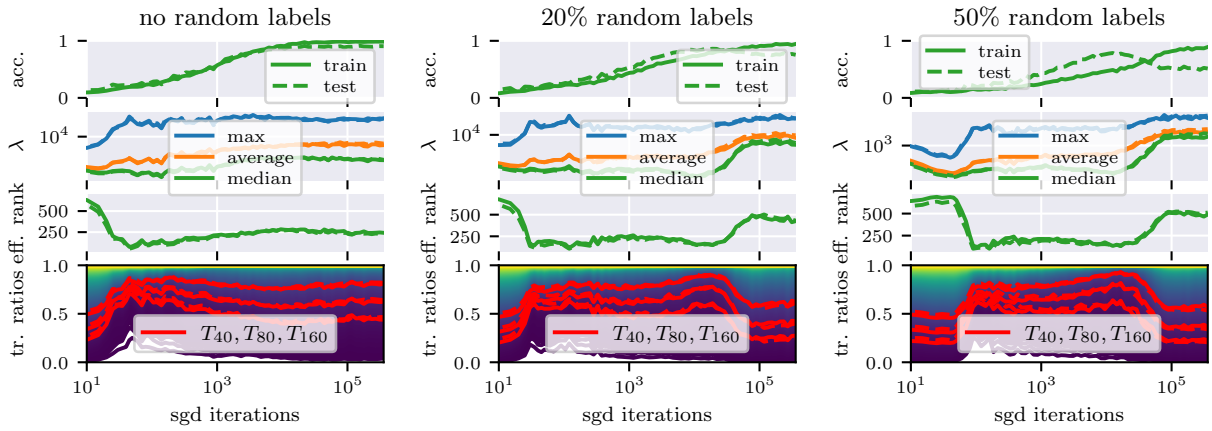



**Figure 2.** Evolution of the tangent kernel **spectrum** (max, average and median eigenvalues), **effective rank** (6.3.1) and **trace ratios** (6.3.2) during training of a VGG19 on CIFAR10 with various ratio of random labels, using cross-entropy and SGD with batch size 100, learning rate 0.01 and momentum 0.9. Tangent kernels are evaluated on batches of size 100 from both the training set (solid lines) and the test set (dashed lines). The plots in the top row show train/test accuracy.

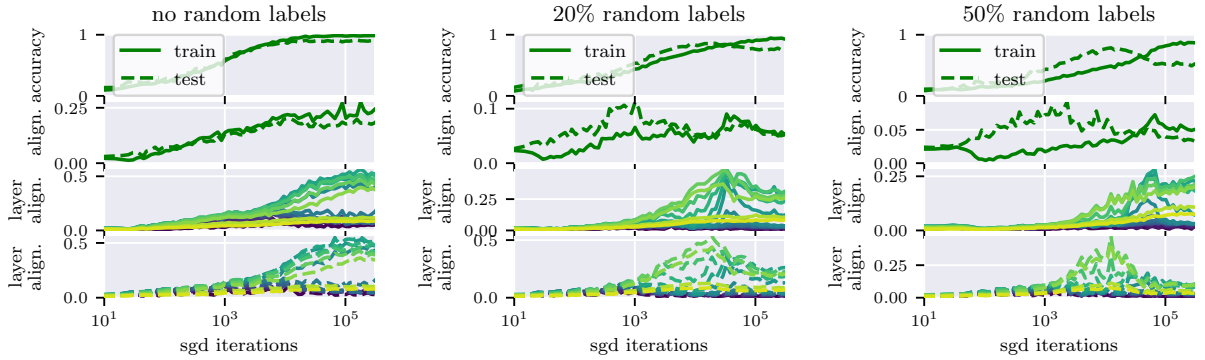



**Figure 3.** Evolution of the (tangent) **feature alignment with class labels** as measured by CKA (6.3.3), during training of a VGG19 on CIFAR10 (same setup as in Fig. 2). Tangent kernels and label vectors are evaluated on batches of size 100 from both the training set (solid lines) and the test set (dashed lines). The plots in the last two rows show the alignment of tangent features associated to *each layer*. Layers are mapped to colors sequentially from input layer (-), through intermediate layers (-), to output layer (-). See Fig. 5 and 7 in Appendix B.3 for additional architectures and datasets.

## 6.3. Neural Feature Alignment

In this section, we study in more detail the evolution of the tangent features during training. Our main results are to highlight (*i*) a sharp increase of the anisotropy of their spectrum early in training; (*ii*) an increasing similarity with the class labels, as measured by **centered kernel alignment** (CKA) (Cristianini et al., 2001; Cortes et al., 2012). tends as a combined mechanism of feature selection and model compression.

### 6.3.1. Setup

We run experiments on MNIST (LeCun et al., 2010) and CIFAR10 (Krizhevsky & Hinton, 2009) with standard MLPs, VGG (Simonyan & Zisserman, 2015) and Resnet (He et al., 2016a) architectures, trained by stochastic gradient descent (SGD) with momentum, using cross-entropy loss. We use PyTorch (Paszke et al., 2019a) and NNGeometry (George, 2021) for efficient evaluation of tangent kernels.

In multiclass settings, tangent kernels evaluated on $n$ samples carry additional class indices $y \in \{1 \cdots c\}$ and thus are $nc \times nc$ matrices, $(\boldsymbol{K}_{\mathbf{w}})_{ij}^{yy'} := k_{\mathbf{w}}(\mathbf{x}_i, y; \mathbf{x}_j, y')$ (details in Appendix B.1.4). In all our experiments, we evaluate tangent kernels on mini-batches of size $n = 100$ from both the training set and the test set; for $c = 10$ classes, this yields kernel matrices of size $1000 \times 1000$. We report results obtained from *centered* tangent features $\Phi_{\mathbf{w}}(\mathbf{x}) \to \Phi_{\mathbf{w}}(\mathbf{x}) - \mathbb{E}_{\mathbf{x}}\Phi_{\mathbf{w}}(\mathbf{x})$, though we obtain qualitatively similar results for uncentered features (see plots in Appendix B.3.2).

### 6.3.2. Spectrum Evolution

We first investigate the evolution of the tangent kernel *spectrum* for a VGG19 on CIFAR 10, trained with and without label noise (Fig. 2). The take away is an anisotropic increase of the spectrum during training. We report results for kernels evaluated on training examples (solid line) and test examples (dashed line).[4]

The first observation is a significant *increase* of the spectrum, early in training (note the log scale for the $x$-axis). By the time the model reaches 100% training accuracy, the maximum and average eigenvalues (Fig. 2, 2nd row) have gained more than 2 orders of magnitude.

The second observation is that this evolution is highly *anisotropic*, i.e larger eigenvalues increase faster than lower ones. This results in a (sharp) increase of spectrum anisotropy, early in training. We quantify this using a notion of **effective rank** based on spectral entropy (Roy & Vetterli, 2007). Given a kernel matrix $\boldsymbol{K}$ in $\mathbb{R}^{r \times r}$ with (strictly) positive eigenvalues $\lambda_1, \cdots, \lambda_r$, let $\mu_j = \lambda_j / \sum_{i=1}^r \lambda_j$ be the trace-normalized eigenvalues. The effective rank is defined as $\text{erank} = \exp(H(\boldsymbol{\mu}))$ where $H(\boldsymbol{\mu})$ is the Shannon entropy,

$$\text{erank} = \exp(H(\boldsymbol{\mu})), \;\; H(\boldsymbol{\mu}) = -\sum_{j=1}^{r} \mu_j \log(\mu_j). \tag{6.3.1}$$

This effective rank is a real number between 1 and $r$, upper bounded by $\text{rank}(\boldsymbol{K})$, which measures the 'uniformity' of the spectrum through the entropy. We also track the various

[4]The striking similarity of the plots for train and test kernels suggests that the spectrum of empirical tangent kernels is robust to sampling variations in our setting.

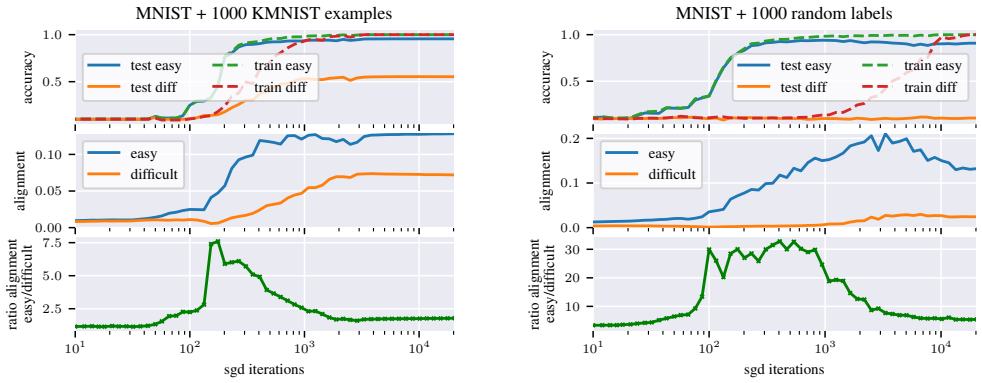



**Figure 4.** Alignment *easy* versus *difficult*: We augment a dataset composed of 10.000 *easy* MNIST examples with 1000 *difficult* examples from 2 different setups: **(left)** 1000 MNIST examples with random label **(right)** 1000 KMNIST examples. We train a MLP with 6 layers of 80 hidden units using SGD with learning rate=0.02, momentum=0.9 and batch size=100. We observe that the alignment to (train) labels increases faster and to a higher value for the easy examples.

**trace ratios**

$$T_k = \sum_{j<k} \lambda_j / \sum_j \lambda_j, \tag{6.3.2}$$

which quantify the relative importance of the top $k$ eigenvalues.

We note (Fig. 2, third row) a drop of the effective rank early in training (e.g. to less than 10% of its initial value in our experiments with no random labels; less than 20% when half of the labels are randomized). This can also be observed from the highlighted (in red) trace ratios $T_{40}$, $T_{80}$ and $T_{160}$ (Fig. 2, fourth row), e.g. the first top 40 eigenvalues ($T_{40}$), over 1000 in total, accounting for more than 70% of the total trace.

Remarkably, in the presence of high label noise, the effective rank of the tangent kernel (and hence that of the metric) evaluated on *training* examples (anti)-correlates nicely with the *test* accuracy: while decreasing and remaining relatively low during the learning phase (increase of test accuracy), it begins to rise again when overfitting starts (decrease of test accuracy). This suggests that this effective rank already provides a good proxy for the effective capacity of the network.

## 6.3.3. Alignment to class labels

We now include the evolution of the eigenvectors in our study. We investigate the similarity of the learned tangent features with the class label through centered kernel alignment. Given two kernel matrices $\boldsymbol{K}$ and $\boldsymbol{K}'$ in $\mathbb{R}^{r\times r}$, it is defined as (Cortes et al., 2012)

$$\mathrm{CKA}(\boldsymbol{K}, \boldsymbol{K}') = \frac{\mathrm{Tr}[\boldsymbol{K}_c \boldsymbol{K}'_c]}{\|\boldsymbol{K}_c\|_F \|\boldsymbol{K}'_c\|_F} \in [0, 1] \tag{6.3.3}$$

where the subscript $c$ denotes the feature centering operation, i.e. $\boldsymbol{K}_c = C\boldsymbol{K}C$ where $C = I_r - \frac{1}{r}\mathbf{1}\mathbf{1}^T$ is the centering matrix, and $\|\cdot\|_F$ is the Froebenius norm. CKA is a

normalized version of the Hilbert-Schmidt Independence Criterion (Gretton et al., 2005) designed as a dependence measure for two sets of features. The normalization makes CKA invariant under isotropic rescaling.

Let $\boldsymbol{Y} \in \mathbb{R}^{nc}$ be the vector resulting from the concatenation of the one-hot label representations $\boldsymbol{Y}_i \in \mathbb{R}^c$ of the $n$ samples. Similarity with the labels is measured through CKA with the rank-one kernel $\boldsymbol{K}_{\boldsymbol{Y}} := \boldsymbol{Y}\boldsymbol{Y}^\top$. Intuitively, $\text{CKA}(\boldsymbol{K}, \boldsymbol{K}_{\boldsymbol{Y}})$ is high when $\boldsymbol{K}$ has low (effective) rank and such that the angle between $\boldsymbol{Y}$ and its top eigenspaces is small.[5] Maximizing such an index has been used as a criterion for kernel selection in the literature on learning kernels (Cortes et al., 2012).

With the same setup as in Section 6.3.2, we observe (Fig. 3, 2nd row) an increasingly high CKA between the tangent kernel and the labels as training progresses. The trend is similar for other architectures and datasets (e.g., Fig. 5 in Appendix B.3 shows CKA plots for MLP on MNIST and Resnets 18 on CIFAR10).

Interestingly, in the presence of high level noise, the CKA reaches a much higher value during the learning phase (increase of test accuracy) for tangent kernels and labels evaluated for *test* than for *train* inputs (note test labels are not randomized). Together with Equ. 6.2.4, this suggests a stronger learning bias towards features predictive of the *clean* labels. This is line with empirical observations that, in the presence of noise, deep networks 'learn patterns faster than noise' (Arpit et al., 2017) (see Section 6.3.4 below for additional insights).

We also report the alignments of the *layer-wise* tangent kernels. By construction, the tangent kernel, obtained by pairing features $\Phi_{w_p}(\mathbf{x})\Phi_{w_p}(\tilde{\mathbf{x}})$ and summing over all parameters $w_p$ of the network, can also be expressed as the sum of layer-wise tangent kernels, $\boldsymbol{K}_{\mathbf{w}} = \sum_{\ell=1}^{L} \boldsymbol{K}_{\mathbf{w}}^{\ell}$, where $\boldsymbol{K}_{\mathbf{w}}^{\ell}$ results from summing only over parameters of the layer $\ell$. We observe a high CKA, reaching more than 0.5 for a number of intermediate layers.[6] In the presence of high label noise, we note that CKAs tend to peak when the test accuracy does.

## 6.3.4. Hierarchical Alignment

A key aspect of the generalization question concerns the articulation between learning and memorization, in the presence of noise (Zhang et al., 2017a) or difficult examples (e.g., Sagawa et al., 2020b). Motivated by this, we would like to probe the evolution of the tangent features *separately* in the directions of both types of examples in such settings. To do so, our strategy is to measure CKA for tangent kernels and label vectors evaluated on examples from two subsets of the same size in the training dataset – one with 'easy' examples, the other with 'difficult' ones. Our setup is to augment 10.000 MNIST training examples with 1000 difficult examples of 2 types: $(i)$ examples with random labels and $(ii)$ examples from

[5] In the limiting case $\text{CKA}(\boldsymbol{K}, \boldsymbol{K}_{\boldsymbol{Y}}) = 1$, the features are all aligned with each other and parallel to $\boldsymbol{Y}$.

[6] We were expecting to see a gradually increasing CKA with $\ell$; we do not have any intuitive explanation for the relatively low alignment observed for the very top layers.

the dataset KMNIST Clanuwat et al. (2018). KMNIST images present features similar to MNIST digits (grayscale handwritten characters) but represent Japanese characters.

The results are shown in Fig. 4. As training progresses, we observe that the CKA on the easy examples increases faster (and to a higher value) than that on the difficult ones; in the case of the (structured) difficult examples from KMNIST, we also note an increase of the CKA later in training. This demonstrates a hierarchy in the adaptation of the kernel, measured by the ratio between both alignments. From the intuition developed in the paper (see spectral bias in Equ.(6.2.4)), we interpret this aspect of the non-linear dynamics as favoring a sequentialization of learning across patterns of different complexity ('easy patterns first'), a phenomenon analogous to one pointed out in the context of deep linear networks (Saxe et al., 2014; Lampinen et al., 2018; Gidel et al., 2019).

### 6.3.5. Sensitivity analysis

**Effect of depth.** In order to study the influence of depth on alignment and test the robustness to the choice of seeds, we reproduce the experiment of the previous section for MLP with different depths, while varying parameter initialization and minibatch sampling. Our results, shown in Fig 7 (Appendix B.3), suggest that the alignment effect is magnified as depth increases. We also observe that the ratio of the maximum alignment between easy and difficult examples is increased with depth, but stays high for a smaller number of iterations.

**Effect of the learning rate.** We observed in our experiments that increasing the learning rate tend to enhance alignment effects.[7]. As an illustration, we reproduce in Fig. 8 the same plots as in Fig. 2, for a learning rate reduced to 0.003. We observe a similar drop of the effective rank as in Fig. 2 at the beginning of training, but to a much (about 3 times) higher value.

## 6.4. Measuring Complexity

In this section, drawing upon intuitions from linear models, we illustrate in a simple setting how the alignment of tangent features can act as implicit regularization. By extrapolating Rademacher complexity bounds for linear models, we also motivate a new complexity measure for neural networks and compare its correlation to generalization against various measures proposed in the literature. We refer to Appendix B.2 for a review of classical results, further technical details, and proofs.

[7]Note that for wide enough networks and small enough learning rate, we expect to recover the linear regime where the tangent features are constant during training (Jacot et al., 2018; Du et al., 2019b; Allen-Zhu et al., 2019).

## 6.4.1. Insights from Linear Models

**6.4.1.1. Setup.** We restrict here to scalar functions $f_{\mathbf{w}}(\mathbf{x}) = \langle \mathbf{w}, \Phi(\mathbf{x}) \rangle$ linearly parametrized by $\mathbf{w} \in \mathbb{R}^P$. Such a function class defines a *constant* (tangent) kernel and geometry, as defined in Section 6.2. Given $n$ input samples, the $n$ features $\Phi(\mathbf{x}_i) \in \mathbb{R}^P$ yield an $n \times P$ feature matrix $\mathbf{\Phi}$.

Our discussion will be based on the (empirical) **Rademacher complexity**, which shows up in generalization bounds Bartlett & Mendelson (2002); see Appendix B.2.2 for a review. It measures how well $\mathcal{F}$ correlates with random noise on the sample set $\mathcal{S}$:

$$\widehat{\mathcal{R}}_{\mathcal{S}}(\mathcal{F}) = \mathbb{E}_{\boldsymbol{\sigma} \in \{\pm 1\}^n} \left[ \sup_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^{n} \sigma_i f(\mathbf{x}_i) \right]. \tag{6.4.1}$$

The Rademacher complexity depends on the size (or **capacity**) of the class $\mathcal{F}$. Constraints on the capacity, such as those induced by the implicit bias of the training algorithm, can reduce the Rademacher complexity and lead to sharper generalization bounds.

A standard approach for controlling capacity is in terms of the *norm* of the weight vector – usually the $\ell_2$-norm. In general, given any invertible matrix $A \in \mathbb{R}^{P \times P}$, we may consider the norm $\|\mathbf{w}\|_A := \sqrt{\mathbf{w}^\top g_A \mathbf{w}}$ induced by the metric $g_A = AA^\top$. Consider the (sub)classes of functions induced by balls of given radius:

$$\mathcal{F}^A_{M_A} = \{ f_{\mathbf{w}} \colon \mathbf{x} \mapsto \langle \mathbf{w}, \Phi(\mathbf{x}) \rangle \mid \|\mathbf{w}\|_A \leq M_A \}. \tag{6.4.2}$$

A direct extension of standard bounds for the Rademacher complexity (see Appendix B.2.3) yields,

$$\widehat{\mathcal{R}}_{\mathcal{S}}(\mathcal{F}^A_{M_A}) \leq (M_A/n) \|A^{-1}\mathbf{\Phi}^\top\|_{\mathrm{F}} \tag{6.4.3}$$

where $\|A^{-1}\mathbf{\Phi}^\top\|_{\mathrm{F}}$ is the Froebenius norm of the *rescaled* feature matrix.[8]

This freedom in the choice of rescaling matrix $A$ raises the question of which of the norms $\|\cdot\|_A$ provide meaningful measures of the model's capacity. Recent works (Belkin et al., 2018; Muthukumar et al., 2020) pointed out that using $\ell_2$ norm is not coherently linked with generalization in practice. We discuss this issue in Appendix B.2.5, illustrating how meaningful norms critically depend on the geometry defined by the features.

**6.4.1.2. Feature Alignment as Implicit Regularization.** Here we describe a simple procedure making the geometry *adaptive* along optimization paths. The goal is to illustrate in a simple setting how feature alignment can impact complexity and generalization, in a way that mimics the behaviour of a non-linear dynamics. The idea is to *learn* a rescaling metric at each iteration of our algorithm, using a local version of the bounds (6.4.3).

**Complexity of Learning Flows.** Since we are interested in functions $f_{\mathbf{w}}$ that result from an iterative algorithm, we consider functions $f_{\mathbf{w}} = \sum_t \delta f_{\mathbf{w}_t}$ written in terms of a sequence

[8] We also have $\|A^{-1}\mathbf{\Phi}^\top\|_{\mathrm{F}} = \sqrt{\mathrm{Tr}\boldsymbol{K}_A}$ in terms of the (rescaled) kernel matrix $\boldsymbol{K}_A = \mathbf{\Phi} g_A^{-1} \mathbf{\Phi}^\top$.

**SuperNat** update ($\tilde{A}_0 = \boldsymbol{I}$, $\Phi_0 = \Phi$, $\boldsymbol{K}_0 = \boldsymbol{K}$):

(1) Perform gradient step $\tilde{\mathbf{w}}_{t+1} \leftarrow \mathbf{w}_t + \delta\mathbf{w}_{\mathrm{GD}}$

(2) Find minimizer $\tilde{A}_{t+1}$ of $\|\delta\mathbf{w}_{\mathrm{GD}}\|_{\tilde{A}}\|\tilde{A}^{-1}\boldsymbol{\Phi}_t^\top$

(3) Reparametrize:

$$\mathbf{w}_{t+1} \leftarrow \tilde{A}_{t+1}^\top \tilde{\mathbf{w}}_{t+1}, \Phi_{t+1} \leftarrow \tilde{A}_{t+1}^{-1}\Phi_t$$

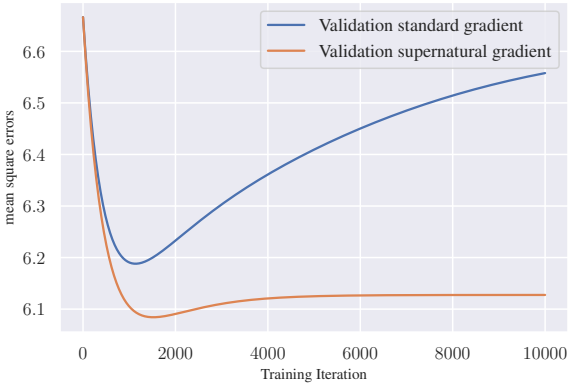



**Figure 5. (left) SuperNat** algorithm and **(right)** validation curves obtained with standard and **SuperNat** gradient descent, on the noisy linear regression problem. At each iteration, **SuperNat** identifies dominant features and stretches the kernel along them, thereby slowing down and eventually freezing the learning dynamics in the noise direction. This naturally yields better generalization than standard gradient descent on this problem.

of updates[9] $\delta f_{\mathbf{w}_t}(\mathbf{x}) = \langle \delta\mathbf{w}_t, \Phi(\mathbf{x})\rangle$ (we set $f_0 = 0$ to keep the notation simple), with *local* constraints on the parameter updates:

$$\mathcal{F}_{\boldsymbol{m}}^{\boldsymbol{A}} = \{f_{\mathbf{w}} \colon \mathbf{x} \mapsto \textstyle\sum_t \langle \delta\mathbf{w}_t, \Phi(\mathbf{x})\rangle \mid \|\delta\mathbf{w}_t\|_{A_t} \le m_t\} \tag{6.4.4}$$

The result (6.4.3) extends as follows.

**Theorem 6.4.1** (Complexity of Learning Flows)**.** *Given any sequences* $\boldsymbol{A}$ *and* $\boldsymbol{m}$ *of invertible matrices* $A_t \in \mathbb{R}^{P\times P}$ *and positive numbers* $m_t > 0$*, we have the bound*

$$\widehat{\mathcal{R}}_{\mathcal{S}}(\mathcal{F}_{\boldsymbol{m}}^{\boldsymbol{A}}) \le \textstyle\sum_t (m_t/n)\|A_t^{-1}\boldsymbol{\Phi}^\top\|_{\mathrm{F}}. \tag{6.4.5}$$

Note that, by linear reparametrization invariance $\mathbf{w} \mapsto A^\top\mathbf{w}$, $\Phi \mapsto A^{-1}\Phi$, the *same* result can be formulated in terms of the sequence $\boldsymbol{\Phi} = \{\Phi_t\}_t$ of feature maps $\Phi_t = A_t^{-1}\Phi$. The function class (6.4.4) can equivalently be written as

$$\mathcal{F}_{\boldsymbol{m}}^{\boldsymbol{\Phi}} = \{f_{\mathbf{w}} \colon \mathbf{x} \mapsto \textstyle\sum_t \langle \tilde{\delta}\mathbf{w}_t, \Phi_t(\mathbf{x})\rangle \mid \|\tilde{\delta}\mathbf{w}_t\|_2 \le m_t\} \tag{6.4.6}$$

In this formulation, the result (6.4.5) reads:

$$\widehat{\mathcal{R}}_{\mathcal{S}}(\mathcal{F}_{\boldsymbol{m}}^{\boldsymbol{\Phi}}) \le \textstyle\sum_t (m_t/n)\|\boldsymbol{\Phi}_t\|_{\mathrm{F}}. \tag{6.4.7}$$

**Optimizing the Feature Scaling.** To obtain learning flows with lower complexity, Thm. 6.4.1 suggests modification of the algorithm to include, at each iteration $t$, a reparametrization step with a suitable matrix $\tilde{A}_t$ giving a low contribution to the bound (6.4.5). Applied to gradient descent (GD), this leads to a new update rule sketched in

[9] In order to not assume a specific upper bound on the number of iterations, we can think of the updates from an iterative algorithm as an infinite sequence $\{\delta\mathbf{w}_0, \cdots \delta\mathbf{w}_t, \cdots\}$ such that for some $T$, $\delta\mathbf{w}_t = 0$ for all $t > T$.

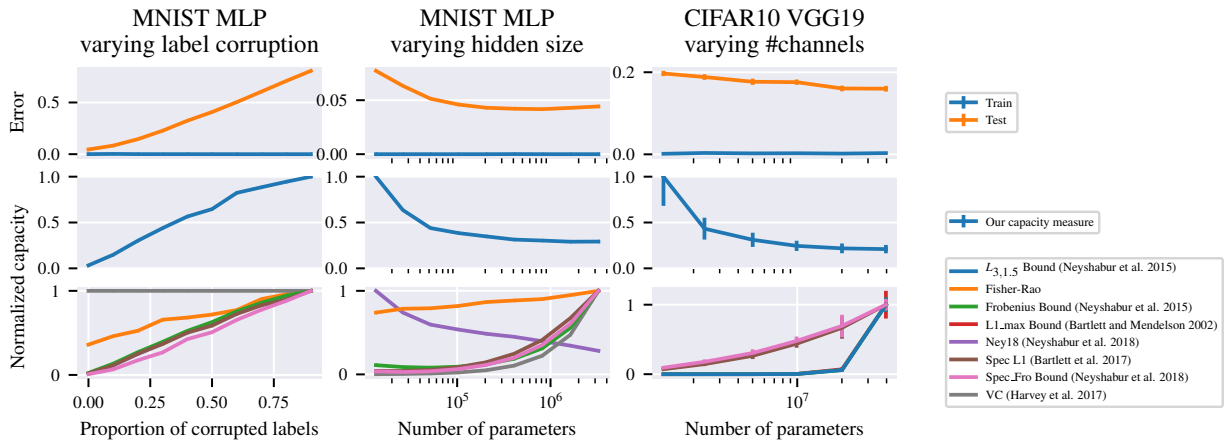



**Figure 6.** Complexity measures on MNIST with a 1 hidden layer MLP **(left)** as we increase the hidden layer size, **(center)** for a fixed hidden layer of 256 units as we increase label corruption and **(right)** for a VGG19 on CIFAR10 as we vary the number of channels. All networks are trained until cross-entropy reaches 0.01. Our proposed complexity measure and the one by Neyshabur et al. 2018 are the only ones to correctly reflect the shape of the generalization gap in these settings.

Fig. 5 (left), where the optimization in Step 2 is over a given class of reparametrization matrices. The successive reparametrizations yield a varying feature map $\Phi_t = A_t^{-1}\Phi$ where $A_t = \tilde{A}_0 \cdots \tilde{A}_t$.[10]

In the original representation $\Phi$, **SuperNat** amounts to natural gradient descent (Amari, 1998) with respect to the local metric $g_{A_t} = A_t A_t^\top$. By construction, we also have $\delta f_{\mathbf{w}_t}(\mathbf{x}) = \langle \delta \mathbf{w}_{\mathrm{GD}}, \Phi_t(\mathbf{x}) \rangle$ where $\delta \mathbf{w}_{\mathrm{GD}}$ are standard gradient descent updates in the linear model with feature map $\Phi_t$.

As an example, let $\mathbf{\Phi} = \sum_{j=1}^n \sqrt{\lambda_j} \boldsymbol{u}_j \boldsymbol{v}_j^\top$ be the SVD of the feature matrix. We restrict to the class of matrices

$$\tilde{A}_{\boldsymbol{\nu}} = \sum_{j=1}^n \sqrt{\nu_j} \boldsymbol{v}_j \boldsymbol{v}_j^\top + \mathrm{Id}_{\mathrm{span}\{\boldsymbol{v}\}^\perp} \tag{6.4.8}$$

labelled by weights $\nu_j > 0, j = 1, \cdots, n$. With such a class, the action $\mathbf{\Phi}_t^\top \to A_{\boldsymbol{\nu}}^{-1} \mathbf{\Phi}_t^\top$ merely rescales the singular values $\lambda_{jt} \to \lambda_{jt}/\nu_j$, leaving the singular vectors unchanged. We work with gradient descent w.r.t a cost function $L$, so that $\delta \mathbf{w}_{\mathrm{GD}} = -\eta \nabla_{\mathbf{w}} L$.

**Proposition 6.4.2.** *Any minimizer in Step 2 of* **SuperNat** *over matrices $\tilde{A}_{\boldsymbol{\nu}}$ in the class (6.4.8), takes the form*

$$\nu_{jt}^* = \kappa \frac{1}{|\boldsymbol{u}_j^\top \nabla_{\boldsymbol{f}_{\mathbf{w}}} L|} \tag{6.4.9}$$

*where $\nabla_{\boldsymbol{f}_{\mathbf{w}}}$ denotes the gradient w.r.t the sample outputs $f_{\mathbf{w}} := [f_{\mathbf{w}}(\mathbf{x}_1), \cdots f_{\mathbf{w}}(\mathbf{x}_n)]^\top$, for some constant $\kappa > 0$.*

[10]Note that upon training a non-linear model, the updates of the tangent features take the same form $\Phi_t = \tilde{A}_t^{-1} \Phi_{t-1}$ as in Step 3 of **SuperNat**, the difference being that $\tilde{A}_t$ is now a differential operator, e.g. at first order $\tilde{A}_t = \mathrm{Id} - \delta \mathbf{w}_t^\top \frac{\partial}{\partial \mathbf{w}_t}$.

In this context, this yields the following update rule, up to isotropic rescaling, for the singular values of $\mathbf{\Phi}_t$:

$$\lambda_{j(t+1)} = |\boldsymbol{u}_j^\top \nabla_{\boldsymbol{f}_\mathbf{w}} L| \lambda_{jt}. \tag{6.4.10}$$

In this illustrative setting, we see how the feature map (or kernel) adapts to the task, by stretching (resp. contracting) its geometry in directions $\boldsymbol{u}_j$ along which the residual $\nabla_{\boldsymbol{f}_\mathbf{w}} L$ has large (resp. small) components. Intuitively, if a large component $|\boldsymbol{u}_j^\top \nabla_{\boldsymbol{f}_\mathbf{w}} L|$ corresponds to signal and a small one $|\boldsymbol{u}_k^\top \nabla_{\boldsymbol{f}_\mathbf{w}} L|$ corresponds to noise, then the ratio $\lambda_{jt}/\lambda_{kt}$ of singular values gets rescaled by the signal-to-noise ratio, thereby increasing the alignment of the learned features to the signal.

As a proof of concept, we consider the following regression setup. We consider a linear model with Gaussian features $\Phi = [\varphi, \varphi_{\text{noise}}] \in \mathbb{R}^{d+1}$ where $\varphi \sim \mathcal{N}(0,1)$ and $\varphi_{\text{noise}} \sim \mathcal{N}(0, \frac{1}{d} I_d)$. Given $n$ input samples, the $n$ features $\Phi(\mathbf{x}_i)$ yield $\boldsymbol{\varphi} \in \mathbb{R}^n$ and $\boldsymbol{\varphi}_{\text{noise}} \in \mathbb{R}^{n \times d}$. We assume the label vector takes the form $\boldsymbol{y} = \boldsymbol{\varphi} + P_{\text{noise}}(\boldsymbol{\epsilon})$, where Gaussian noise $\boldsymbol{\epsilon} \sim \mathcal{N}(0, \sigma^2 I_n)$ is projected onto the noise features through $P_{\text{noise}} = \boldsymbol{\varphi}_{\text{noise}} \boldsymbol{\varphi}_{\text{noise}}^\top$. The model is trained by gradient descent of the mean squared loss and its **SuperNat** variant, where Step 2 uses the analytical solution of Proposition 6.4.2. We set $d = 10, \sigma^2 = 0.1$ and use $n = 50$ training points.

Fig 5 (right) shows test error obtained with standard and **SuperNat** gradient descent on this problem. At each iteration, **SuperNat** identifies dominant features (feature selection, here $\varphi$) and stretches the metric along them, thereby slowing down and eventually freezing the dynamics in the orthogonal (noise) directions (compression). The working hypothesis in this paper, supported by the observations of Section 6.3, is that for neural networks, such a (tangent) feature alignment is dynamically induced as an effect of non-linearity.

## 6.4.2. A New Complexity Measure for Neural Networks

Equ. (6.4.7) provides a bound of the Rademacher complexity for the function classes (6.4.4) specified by a *fixed* sequence of feature maps (see Appendix B.2.4 for a generalization to the multiclass setting). By extrapolation to the case of non-deterministic sequences of feature maps, we propose using

$$\mathcal{C}(f_\mathbf{w}) = \sum_t \|\delta \mathbf{w}_t\|_2 \|\mathbf{\Phi}_t\|_{\text{F}} \tag{6.4.11}$$

as a heuristic measure of complexity for neural networks, where $\mathbf{\Phi}_t$ is the learned tangent feature matrix[11] at training iteration $t$, and $\|\delta \mathbf{w}_t\|_2$ is the norm of the SGD update. Following a standard protocol for studying complexity measures (e.g., Neyshabur et al., 2017a), Fig. 6 shows its behaviour for MLP on MNIST and VGG19 on CIFAR10 trained with cross entropy loss, with **(left)** fixed architecture and varying level of corruption in the labels and **(right)**

[11] In terms of tangent kernels, $\|\mathbf{\Phi}_t\|_{\text{F}} = \sqrt{\text{Tr} \boldsymbol{K}_t}$ where $\boldsymbol{K}_t$ is the tangent kernel (Gram) matrix.

varying hidden layer size/number of channels up to 4 millions parameters, against other capacity measures proposed in the recent literature. We observe that it correctly reflects the shape of the generalization gap.

## 6.5. Related Work

**Role of Feature Geometry in Linear Models.** Analysis of the relation between capacity and feature geometry can be traced back to early work on kernel methods (Schölkopf et al., 1999a), which lead to data-dependent error bounds in terms of the eigenvalues of the kernel Gram matrix (Schölkopf et al., 1999b).

Recently, new analysis of minimum norm interpolators and max margin solutions for overparametrized linear models emphasize the key role of feature geometry, and specifically feature anisotropy, in the generalization performance (Bartlett et al., 2019; Muthukumar et al., 2019, 2020; Xie et al., 2020). Feature anisotropy combined to a high predictive power of the dominant features is the condition for a high centered alignment between kernel and class labels. In the context of neural networks, our results highlight the role of the non linear training dynamics in favouring such conditions.

**Generalization Measures.** There has been a large body of work on complexity/generalization measures for neural networks (see Jiang et al., 2020, and references therein), some of which theoretically motivated by norm or margin based bounds (e.g., Neyshabur et al., 2019; Bartlett et al., 2017). Liang et al. (2019) proposed using the Fisher-Rao norm of the solution as a geometrically invariant complexity measure. By contrast, our approach to measuring complexity takes into account the geometry along the whole optimization trajectories. Since the geometry we consider is defined through the gradient second moments, our perspective is closely related to the notions of stiffness (Fort et al., 2019) and coherent gradients (Chatterjee, 2020).

**Dynamics of Tangent Kernels**. Several recent works investigated the 'feature learning' regime where neural tangent kernels evolve during training (Geiger et al., 2020; Woodworth et al., 2020). Independent concurrent works highlight alignment and compression phenomena similar to the one we study here (Kopitkov & Indelman, 2020; Paccolat et al., 2021). We offer various complementary empirical insights, and frame the alignment mechanism from the point of view of implicit regularization.

## 6.6. Conclusion

Through experiments with modern architectures, we highlighted an effect of dynamical alignment of the neural tangent features and their kernel along a small number of task-dependent directions during training, reflected by an early drop of the effective rank and an increasing similarity with the class labels, as measured by centered kernel alignment. We

interpret this effect as a combined mechanism of feature selection and model compression of around dominant features.

Drawing upon intuitions from linear models, we argued that such a dynamical alignment acts as implicit regularizer. By extrapolating a new analysis of Rademacher complexity bounds for linear models, we also proposed a complexity measure that captures this phenomenon, and showed that it correlates with the generalization gap when varying the number of parameters, and when increasing the proportion of corrupted labels.

The results of this paper open several avenues for further investigation. The type of complexity measure we propose suggests new principled ways to design algorithms that learn the geometry in which to perform gradient descent (Srebro et al., 2011; Neyshabur et al., 2017b). Whether a procedure such as **SuperNat** can produce meaningful practical results for neural networks remains to be seen.

One of the consequences one can expect from the alignment effects highlighted here is to bias learning towards explaining most of the data with a small number of highly predictive features. While this feature selection ability might explain in part the performance of neural networks on a range of supervised tasks, it may also make them brittle under spurious correlation (e.g., Sagawa et al., 2020b) and underpin their notorious weakness to generalize out-of-distribution (e.g., Geirhos et al., 2020). Resolving this tension is an important challenge towards building more robust models.

# Chapter 7

# Conclusion and discussion

## 7.1. Summary

Trained deep learning models are the result of an iterative optimization algorithm (i.e. variants of gradient descent) that travels through the parameter space from a randomly initialized state $\mathbf{w}_0$, to final parameters $\mathbf{w}_T$ after $T$ iterates. The final function used as a predictor is thus an ensemble of the contributions $g_t$ of each of these iterates:

$$f_{\mathbf{w}_T} = f_{\mathbf{w}_0} + \sum_{t=0}^{T-1} \underbrace{\left(f_{\mathbf{w}_{t+1}} - f_{\mathbf{w}_t}\right)}_{:=g_t} \tag{7.1.1}$$

In this thesis, we studied the local linearization of iterates $\{g_t\}_t$:

$$g_t(x) = \langle \delta\mathbf{w}_t, \Phi_{\mathbf{w}_t}(x) \rangle + O(\|\delta\mathbf{w}_t\|^2) \tag{7.1.2}$$

We argued that studying the sequence of Jacobians/tangent features $\{\Phi_{\mathbf{w}_t}\}_t$ is a promising avenue in order to improve our understanding of the properties of trained deep models, along the following axes:

(1) We released a software library that enables to work with Jacobians, Neural Tangent Kernels and Fisher Information Matrices flawlessly through a unified Application Programming Interface. We described new algorithmic enhancements to manipulate these objects even in highly overparameterized settings where they cannot fit in memory (Chapter 3).

(2) We introduced an efficient approximate to the Fisher Information Matrix, proved that it is more accurate than previously published analogous techniques, and that it can be re-estimated at low computational cost when parameters vary. We used it in the natural gradient algorithm to counteract the ill-conditioning of the Jacobians (Chapter 4).

(3) We compared the full training dynamics of deep networks to linearized dynamics at initialization, using groups of examples, and showed that the sequentialization of

learning from easy to difficult examples is magnified in the full training dynamics. Compared to recent results that use the NTK in the infinite-width limit regime, this suggests a more pronounced implicit bias towards a predictor relying on the most representative examples in the training dataset, while essentially discarding less relevant examples in a first phase of the training course (Chapter 5).

(4) We discovered that the time-varying Neural Tangent Kernel increasingly aligns to the target kernel during training, leading to a regularization mechanism that restricts the effective capacity of trained models to a few directions in function space. We derived a complexity measure for families of functions resulting from a sequence of kernels, mimicking our modelling of the full training dynamics. Through a sensitivity analysis, we showed that this measure correlates with the generalization gap as we vary the number of parameters or the noise level in the training dataset (Chapter 6).

These findings are some building blocks in a global effort for a comprehensive theory explaining generalization in deep learning. We are however still far from fully cracking the mystery of their good properties or their failure modes.

## 7.2. Proposed future avenues

This thesis suggests exploring some future avenues:

**Adoption of optimization methods.** There are still some drawbacks when using more sophisticated optimization methods such as the natural gradient in place of first order methods, that hinder wider adoption, and call for more research effort. By allowing to take larger steps, they are often working on the edge of stability, where a slightly larger steps (caused by a too large learning rate, or a too small damping parameter) will cause the loss to blow up. The main issue with current minibatched algorithms is to deal with stochastic estimates of i) the gradient and ii) the FIM as well as the fact that we do not work with the full FIM but rather an estimate of it. Following the work of Mahsereci & Hennig (2015) that propose a technique for an adaptive learning rate in such as setting, it could be interesting to develop a similar approach for the damping parameter. The ultimate goal would be to package a versatile optimizer that can replace more popular methods such as SGD with momentum, that would accelerate training with default parameters in most cases, while not hurting generalization

**Architectures and $2^{\text{nd}}$ order.** We also propose the hypothesis that current architectures are tuned to be trainable by first order methods, since they were developed using frameworks that foster trying them first. An interesting avenue of research would be to explore this hypothesis, keeping in mind that biological neural networks (i.e. brains) certainly do not use first order methods, and that they remain the best general purpose learning systems.

**Salient directions.** The elephant in the room to the attentive reader of this thesis is the apparent contradiction between the last chapters (5 and 6) that suggest that the reason for the good generalization properties of deep learning is that they favor a few salient directions, and Chapter 4 where we study a natural gradient optimizer, that precisely amplifies less representative directions. We conjecture that this is where the damping parameter plays an important role: too small a damping parameter and we race in all directions, including those detrimental to generalization; too large a damping parameter and we successfully filter out small modes, but at the cost of a slower optimization of the training loss. In the context of kernelized linear models, Ma & Belkin (2017) advocate for a hard cut-off that selects a few top modes and zeros out other directions, an idea that could be adapted to neural network training.

**Data-adaptive methods.** The insights from Chapter 5 comparing linear and non-linear training regimes suggest that while in most cases the full non-linear training dynamics is beneficial to generalization, by contrast in presence of spurious correlations, it makes the trained model more sensible to the spuriously correlated features. A follow-up could be to design an optimizer that adaptively tunes the regime during training. More generally, recent works (Sagawa et al., 2020b; Hooker et al., 2020) that study the importance of groups of examples between a head of representative examples and a tail of examples that are merely memorized, may inspire new algorithms that weight examples differently.

## 7.3. Closing words

Over the past decade, there has been a significant shift in the perception of neural networks, from being considered too enigmatic to be practically useful, to their current widespread deployment in various fields due to their practical successes. As intriguing as it is, deep learning is changing many aspects of our societies. It is capable of improving automation of tedious work, freeing humanity to do other more interesting activities, but it is also capable of piloting automated weapons, create nearly indistinguishable fake videos or influencing our shopping behaviors. As researchers, it is important that we develop tools to analyze deep learning models in order to explain to the public what can be done so that applications of Artificial Intelligence are beneficial to humanity, and the dystopias pictured in science-fiction movies never arise.

# References


Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, pp. 265–283, 2016.

Madhu S Advani and Andrew M Saxe. High-dimensional dynamics of generalization error in neural networks. *arXiv preprint arXiv:1710.03667*, 2017.

Atish Agarwala, Jeffrey Pennington, Yann Dauphin, and Sam Schoenholz. Temperature check: theory and practice for training models with softmax-cross-entropy losses, 2020. URL https://arxiv.org/abs/2010.07344.

Zeyuan Allen-Zhu, Yuanzhi Li, and Zhao Song. A convergence theory for deep learning via over-parameterization. volume 97 of *Proceedings of Machine Learning Research*, pp. 242–252, Long Beach, California, USA, 09–15 Jun 2019. PMLR. URL http://proceedings.mlr.press/v97/allen-zhu19a.html.

Shun-ichi Amari. Differential-geometrical methods in statistics. *Lecture Notes on Statistics*, 28:1, 1985.

Shun-Ichi Amari. Natural gradient works efficiently in learning. *Neural computation*, 1998.

Shun-Ichi Amari. *Information Geometry and Its Applications*, volume 194. Springer, 2016.

Sanjeev Arora, Simon S Du, Wei Hu, Zhiyuan Li, Russ R Salakhutdinov, and Ruosong Wang. On exact computation with an infinitely wide neural net. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/file/dbc4d84bfcfe2284ba11beffb853a8c4-Paper.pdf.

Devansh Arpit, Stanislaw Jastrzebski, Nicolas Ballas, David Krueger, Emmanuel Bengio, Maxinder S. Kanwal, Tegan Maharaj, Asja Fischer, Aaron Courville, Yoshua Bengio, and Simon Lacoste-Julien. A closer look at memorization in deep networks. In Doina Precup and Yee Whye Teh (eds.), *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 233–242. PMLR, 06–11 Aug 2017. URL https://proceedings.mlr.press/v70/arpit17a.html.

Alexander Atanasov, Blake Bordelon, and Cengiz Pehlevan. Neural networks as kernel learners: The silent alignment effect. In *International Conference on Learning Representations*, 2021.

Juhan Bae, Guodong Zhang, and Roger Grosse. Eigenvalue corrected noisy natural gradient. *arXiv preprint arXiv:1811.12565*, 2018.

Robert John Nicholas Baldock, Hartmut Maennel, and Behnam Neyshabur. Deep learning through the lens of example difficulty. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=WWRBHhH158K.

Aristide Baratin, Thomas George, César Laurent, R Devon Hjelm, Guillaume Lajoie, Pascal Vincent, and Simon Lacoste-Julien. Implicit regularization via neural feature alignment. In Arindam Banerjee and Kenji Fukumizu (eds.), *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*, volume 130 of *Proceedings of Machine Learning Research*, pp. 2269–2277. PMLR, 13–15 Apr 2021. URL https://proceedings.mlr.press/v130/baratin21a.html.

Peter L Bartlett and Shahar Mendelson. Rademacher and gaussian complexities: Risk bounds and structural results. *JMLR*, 2002.

Peter L. Bartlett, Dylan J. Foster, and Matus Telgarsky. Spectrally-normalized margin bounds for neural networks. In *NIPS*, 2017.

Peter L Bartlett, Philip M Long, Gábor Lugosi, and Alexander Tsigler. Benign overfitting in linear regression. *arXiv preprint arXiv:1906.11300[stat.ML]*, 2019.

Peter L Bartlett, Andrea Montanari, and Alexander Rakhlin. Deep learning: a statistical viewpoint. *Acta numerica*, 30:87–201, 2021.

Ronen Basri, David Jacobs, Yoni Kasten, and Shira Kritchman. The convergence rate of neural networks for learned functions of different frequencies. In *Advances in Neural Information Processing Systems 32*, pp. 4761–4771. 2019. URL http://papers.nips.cc/paper/8723-the-convergence-rate-of-neural-networks-for-learned-functions-of-different-frequ pdf.

Sue Becker, Yann Le Cun, et al. Improving the convergence of back-propagation learning with second order methods. In *Proceedings of the 1988 connectionist models summer school*. San Matteo, CA: Morgan Kaufmann, 1988.

Mikhail Belkin, Siyuan Ma, and Soumik Mandal. To understand deep learning we need to understand kernel learning. In *ICML*, 2018.

Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mandal. Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences*, 116(32):15849–15854, 2019.

Yoshua Bengio, Olivier Delalleau, Nicolas Le Roux, Jean-François Paiement, Pascal Vincent, and Marie Ouimet. Learning eigenfunctions links spectral embedding and kernel

PCA. *Neural Computation*, 16(10):2197–2219, 2004. URL http://www.iro.umontreal.ca/~lisa/pointeurs/bengio_eigenfunctions_nc_2004.pdf.

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pp. 41–48, 2009.

Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35 (8):1798–1828, 2013.

James Bergstra, Frédéric Bastien, Olivier Breuleux, Pascal Lamblin, Razvan Pascanu, Olivier Delalleau, Guillaume Desjardins, David Warde-Farley, Ian Goodfellow, Arnaud Bergeron, et al. Theano: Deep learning on gpus with python. In *NIPS 2011, BigLearning Workshop, Granada, Spain*, volume 3, pp. 1–48. Citeseer, 2011.

Alberto Bietti and Julien Mairal. On the inductive bias of neural tangent kernels. In *Advances in Neural Information Processing Systems 32*, pp. 12893–12904. 2019. URL http://papers.nips.cc/paper/9449-on-the-inductive-bias-of-neural-tangent-kernels.pdf.

Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

Aleksandar Botev, Hippolyt Ritter, and David Barber. Practical gauss-newton optimisation for deep learning. In *International Conference on Machine Learning*, pp. 557–565, 2017.

Léon Bottou, Frank E Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *arXiv preprint*, 2016.

S Boucheron, O Bousquet, and G Lugosi. Advanced lectures in machine learning, chapter introduction to statistical learning theory, 2004.

Mikio L Braun. *Spectral properties of the kernel matrix and their relation to kernel methods in machine learning*. PhD thesis, Universitäts-und Landesbibliothek Bonn, 2005.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Joy Buolamwini and Timnit Gebru. Gender shades: Intersectional accuracy disparities in commercial gender classification. In *Conference on fairness, accountability and transparency*, pp. 77–91. PMLR, 2018.

Satrajit Chatterjee. Coherent gradients: An approach to understanding generalization in gradient descent-based optimization. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=ryeFY0EFwS.

Lénaïc Chizat and Francis Bach. On the global convergence of gradient descent for over-parameterized models using optimal transport. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (eds.), *Advances in*

*Neural Information Processing Systems*, volume 31, pp. 3036–3046. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper/2018/file/a1afc58c6ca9540d057299ec3016d726-Paper.pdf.

Lenaic Chizat, Edouard Oyallon, and Francis Bach. On lazy training in differentiable programming. *Advances in Neural Information Processing Systems*, 32, 2019.

Tarin Clanuwat, Mikel Bober-Irizar, Asanobu Kitamoto, Alex Lamb, Kazuaki Yamamoto, and David Ha. Deep learning for classical japanese literature. 2018.

Corinna Cortes, Mehryar Mohri, and Afshin Rostamizadeh. Algorithms for learning kernels based on centered alignment. *JMLR*, 13(1):795–828, 2012. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=2503308.2188413.

Andrew Cotter, Ohad Shamir, Nati Srebro, and Karthik Sridharan. Better mini-batch algorithms via accelerated gradient methods. In J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K.Q. Weinberger (eds.), *Advances in Neural Information Processing Systems*, volume 24. Curran Associates, Inc., 2011. URL https://proceedings.neurips.cc/paper/2011/file/b55ec28c52d5f6205684a473a2193564-Paper.pdf.

Nello Cristianini, John Shawe-Taylor, André Elisseeff, and Jaz Kandola. On kernel-target alignment. In T. Dietterich, S. Becker, and Z. Ghahramani (eds.), *Advances in Neural Information Processing Systems*, volume 14. MIT Press, 2001. URL https://proceedings.neurips.cc/paper/2001/file/1f71e393b3809197ed66df836fe833e5-Paper.pdf.

Felix Dangel, Frederik Kunstner, and Philipp Hennig. Backpack: Packing more into backprop. In *International Conference on Learning Representations*, 2019.

Aaron Defazio, Francis Bach, and Simon Lacoste-Julien. Saga: A fast incremental gradient method with support for non-strongly convex composite objectives. *Advances in neural information processing systems*, 27, 2014.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255. Ieee, 2009.

Guillaume Desjardins, Karen Simonyan, Razvan Pascanu, et al. Natural neural networks. In *NIPS*, 2015.

Laurent Dinh, Razvan Pascanu, Samy Bengio, and Yoshua Bengio. Sharp minima can generalize for deep nets. *arXiv preprint arXiv:1703.04933*, 2017.

Simon S. Du, Jason D. Lee, Haochuan Li, Liwei Wang, and Xiyu Zhai. Gradient descent finds global minima of deep neural networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov (eds.), *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*, pp. 1675–1685. PMLR, 2019a. URL http://proceedings.mlr.press/v97/du19c.html.

Simon S. Du, Xiyu Zhai, Barnabas Poczos, and Aarti Singh. Gradient descent provably optimizes over-parameterized neural networks. In *International Conference on Learning Representations*, 2019b. URL https://openreview.net/forum?id=S1eK3i09YQ.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul): 2121–2159, 2011.

Alexander D'Amour, Katherine Heller, Dan Moldovan, Ben Adlam, Babak Alipanahi, Alex Beutel, Christina Chen, Jonathan Deaton, Jacob Eisenstein, Matthew D Hoffman, et al. Underspecification presents challenges for credibility in modern machine learning. *Journal of Machine Learning Research*, 2020.

Jeffrey L. Elman. Learning and development in neural networks: The importance of starting small. *Cognition*, 48:71–99, 1993.

Vitaly Feldman and Chiyuan Zhang. What neural networks memorize and why: Discovering the long tail via influence estimation. *Advances in Neural Information Processing Systems*, 33:2881–2891, 2020.

Stanislav Fort, Pawel Krzysztof Nowak, Stanislaw Jastrzebski, and Srini Narayanan. Stiffness: A new perspective on generalization in neural networks. *arXiv preprint arXiv:1901.09491*, 2019.

Stanislav Fort, Gintare Karolina Dziugaite, Mansheej Paul, Sepideh Kharaghani, Daniel M Roy, and Surya Ganguli. Deep learning versus kernel learning: an empirical study of loss landscape geometry and the time evolution of the neural tangent kernel. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 5850–5861. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/405075699f065e43581f27d67bb68478-Paper.pdf.

Yuki Fujimoto and Toru Ohira. A neural network model with bidirectional whitening. In *International Conference on Artificial Intelligence and Soft Computing*, pp. 47–57. Springer, 2018.

Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. Convolutional sequence to sequence learning. In *ICLR*, 2017.

Mario Geiger, Stefano Spigler, Arthur Jacot, and Matthieu Wyart. Disentangling feature and lazy training in deep neural networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2020(11):113301, nov 2020. doi: 10.1088/1742-5468/abc4de. URL https://doi.org/10.1088%2F1742-5468%2Fabc4de.

Robert Geirhos, Jörn-Henrik Jacobsen, Claudio Michaelis, Richard Zemel, Wieland Brendel, Matthias Bethge, and Felix A Wichmann. Shortcut learning in deep neural networks. *arxiv preprint arXiv:2004.07780 [cs.CV]*, 2020.

Stuart Geman, Elie Bienenstock, and René Doursat. Neural networks and the bias/variance dilemma. *Neural Computation*, 4(1):1–58, 1992. doi: 10.1162/neco.1992.4.1.1. URL https://doi.org/10.1162/neco.1992.4.1.1.

Paul-Louis George. Writing a good paper in osm. 2005.

Thomas George. Factorized second order methods in neural networks. 2018.

Thomas George. NNGeometry: Easy and Fast Fisher Information Matrices and Neural Tangent Kernels in PyTorch, February 2021. URL https://doi.org/10.5281/zenodo.4532597.

Thomas George, César Laurent, Xavier Bouthillier, Nicolas Ballas, and Pascal Vincent. Fast approximate natural gradient descent in a kronecker factored eigenbasis. In *Advances in Neural Information Processing Systems*, pp. 9550–9560, 2018.

Thomas George, Guillaume Lajoie, and Aristide Baratin. Lazy vs hasty: linearization in deep networks impacts learning schedule based on example difficulty. *Transactions on Machine Learning Research*, 2022. URL https://openreview.net/forum?id=lukVf4VrfP.

Gauthier Gidel, Francis Bach, and Simon Lacoste-Julien. Implicit regularization of discrete gradient dynamics in linear neural networks. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/file/f39ae9ff3a81f499230c4126e01f421b-Paper.pdf.

Ian Goodfellow. Efficient per-example gradient computations. *arXiv preprint arXiv:1510.01799*, 2015.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. The MIT Press, 2016. ISBN 0262035618, 9780262035613.

Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

Arthur Gretton, Olivier Bousquet, Alexander Smola, and Bernhard Schölkopf. Measuring statistical dependence with hilbert-schmidt norms, 2005.

Roger Grosse and James Martens. A kronecker-factored approximate fisher matrix for convolution layers. In *ICML*, 2016.

Suriya Gunasekar, Jason Lee, Daniel Soudry, and Nathan Srebro. Characterizing implicit bias in terms of optimization geometry. In Jennifer Dy and Andreas Krause (eds.), *ICML*, volume 80 of *Proceedings of Machine Learning Research*, pp. 1832–1841, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR. URL http://proceedings.mlr.press/v80/gunasekar18a.html.

Stephen Hanson and Lorien Pratt. Comparing biases for minimal network construction with back-propagation. *Advances in neural information processing systems*, 1, 1988.

Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning: data mining, inference and prediction*. Springer, 2009.

Trevor Hastie, Andrea Montanari, Saharon Rosset, and Ryan J. Tibshirani. Surprises in high-dimensional ridgeless least squares interpolation. *The Annals of Statistics*, 50(2):949 – 986, 2022. doi: 10.1214/21-AOS2133. URL `https://doi.org/10.1214/21-AOS2133`.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *ICCV*, 2015.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016a.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *ECCV*. Springer, 2016b.

Tom Heskes. On "natural" learning and pruning in multilayered perceptrons. *Neural Computation*, 12(4):881–901, 2000.

Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

Elad Hoffer, Itay Hubara, and Daniel Soudry. Train longer, generalize better: Closing the generalization gap in large batch training of neural networks. In *NIPS*, 2017.

Sara Hooker, Aaron Courville, Gregory Clark, Yann Dauphin, and Andrea Frome. What do compressed deep neural networks forget? *arXiv preprint arXiv:1911.05248 [cs.LG]*, 2020.

Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015.

Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In *Advances in neural information processing systems*, pp. 8571–8580, 2018.

Yiding Jiang, Behnam Neyshabur, Hossein Mobahi, Dilip Krishnan, and Samy Bengio. Fantastic generalization measures and where to find them. In *ICLR*, 2020.

Ziheng Jiang, Chiyuan Zhang, Kunal Talwar, and Michael C. Mozer. Characterizing structural regularities of labeled data in overparameterized models. In Marina Meila and Tong Zhang (eds.), *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pp. 5034–5044. PMLR, 2021. URL `http://proceedings.mlr.press/v139/jiang21k.html`.

Rie Johnson and Tong Zhang. Accelerating stochastic gradient descent using predictive variance reduction. *Advances in neural information processing systems*, 26, 2013.

Dimitris Kalimeris, Gal Kaplun, Preetum Nakkiran, Benjamin Edelman, Tristan Yang, Boaz Barak, and Haofeng Zhang. Sgd on neural networks learns functions of increasing complexity. *Advances in neural information processing systems*, 32, 2019.

Ryo Karakida, Shotaro Akaho, and Shun-ichi Amari. Pathological spectra of the fisher information metric and its variants in deep neural networks. *arXiv:1910.05992 [stat.ML]*, 2019a.

Ryo Karakida, Shotaro Akaho, and Shun-ichi Amari. Universal statistics of fisher information in deep neural networks: Mean field approach. *AISTATS 2019*, 2019b.

Kenji Kawaguchi, Leslie Pack Kaelbling, and Yoshua Bengio. Generalization in deep learning. *arXiv preprint arXiv:1710.05468*, 2017.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions. In *International conference on machine learning*, pp. 1885–1894. PMLR, 2017.

Dmitry Kopitkov and Vadim Indelman. Neural spectrum alignment: Empirical study. In *International Conference on Artificial Neural Networks*, pp. 168–179. Springer, 2020.

Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. 2009.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

Anders Krogh and John A Hertz. A simple weight decay can improve generalization. In *Advances in neural information processing systems*, pp. 950–957, 1992.

Frederik Kunstner, Philipp Hennig, and Lukas Balles. Limitations of the empirical fisher approximation for natural gradient descent. In *Advances in Neural Information Processing Systems*, pp. 4156–4167, 2019.

Andrew K Lampinen, Andrew K Lampinen, and Surya Ganguli. An analytic theory of generalization dynamics and transfer learning in deep linear networks. *arXiv.org*, 2018.

S. Lang. *Fundamentals of Differential Geometry*. Graduate Texts in Mathematics. Springer New York, 2012. ISBN 9781461205418. URL `https://books.google.ca/books?id=Ku3jBwAAQBAJ`.

César Laurent, Thomas George, Xavier Bouthillier, Nicolas Ballas, and Pascal Vincent. An evaluation of fisher approximations beyond kronecker factorization. *ICLR Workshop*, 2018.

César Laurent, Camille Ballas, Thomas George, Nicolas Ballas, and Pascal Vincent. Revisiting loss modelling for unstructured pruning. *CoRR*, abs/2006.12279, 2020. URL

https://arxiv.org/abs/2006.12279.

Nicolas Le Roux, Pierre-Antoine Manzagol, and Yoshua Bengio. Topmoumoute online natural gradient algorithm. In *NIPS*, 2008.

Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010.

M. Ledoux and M. Talagrand. *Probability in Banach Spaces: Isoperimetry and Processes*. Springer Science & Business, New York, 2013.

Timothée Lesort, Thomas George, and Irina Rish. Continual learning in deep networks: an analysis of the last layer. *arXiv preprint arXiv:2106.01834*, 2021.

Yuanzhi Li, Colin Wei, and Tengyu Ma. Towards explaining the regularization effect of initial large learning rate in training neural networks. *Advances in Neural Information Processing Systems*, 32, 2019.

Tengyuan Liang, Tomaso Poggio, Alexander Rakhlin, and James Stokes. Fisher-rao metric, geometry, and complexity of neural networks. In *Proceedings of Machine Learning Research*, volume 89, pp. 888–896, 2019.

Dong C Liu and Jorge Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical programming*, 1989.

Xialei Liu, Marc Masana, Luis Herranz, Joost Van de Weijer, Antonio M Lopez, and Andrew D Bagdanov. Rotate your networks: Better weight consolidation and less catastrophic forgetting. In *2018 24th International Conference on Pattern Recognition (ICPR)*, pp. 2262–2268. IEEE, 2018.

Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015.

Philip M Long. Properties of the after kernel. *arXiv preprint arXiv:2105.10585*, 2021.

Bruno Loureiro, Cedric Gerbelot, Hugo Cui, Sebastian Goldt, Florent Krzakala, Marc Mezard, and Lenka Zdeborová. Learning curves of generic features maps for realistic datasets with a teacher-student model. *Advances in Neural Information Processing Systems*, 34:18137–18151, 2021.

Siyuan Ma and Mikhail Belkin. Diving into the shallows: a computational perspective on large-scale shallow learning. In *Advances in Neural Information Processing Systems*, pp. 3781–3790, 2017.

Wesley J Maddox, Pavel Izmailov, Timur Garipov, Dmitry P Vetrov, and Andrew Gordon Wilson. A simple baseline for bayesian uncertainty in deep learning. In *Advances in Neural Information Processing Systems*, pp. 13153–13164, 2019.

Maren Mahsereci and Philipp Hennig. Probabilistic line searches for stochastic optimization. *Advances in neural information processing systems*, 28, 2015.

Julien Mairal. Incremental majorization-minimization optimization with application to large-scale machine learning. *SIAM Journal on Optimization*, 25(2):829–855, 2015. doi: 10.1137/140957639. URL https://doi.org/10.1137/140957639.

Julien Mairal, Piotr Koniusz, Zaid Harchaoui, and Cordelia Schmid. Convolutional kernel networks. *Advances in neural information processing systems*, 27, 2014.

Eran Malach, Pritish Kamath, Emmanuel Abbe, and Nathan Srebro. Quantifying the benefit of using differentiable learning over tangent kernels. In Marina Meila and Tong Zhang (eds.), *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 7379–7389. PMLR, 18–24 Jul 2021. URL https://proceedings.mlr.press/v139/malach21a.html.

James Martens. New Insights and Perspectives on the Natural Gradient Method. *Journal of Machine Learning Research*, 21(146):1–76, 2020. URL http://jmlr.org/papers/v21/17-678.html.

James Martens and Roger Grosse. Optimizing neural networks with kronecker-factored approximate curvature. In *ICML*, 2015.

James Martens, Jimmy Ba, and Matt Johnson. Kronecker-factored curvature approximations for recurrent neural networks. In *International Conference on Learning Representations*, 2018.

James Martens et al. Deep learning via hessian-free optimization. 2010.

Song Mei and Andrea Montanari. The generalization error of random features regression: Precise asymptotics and the double descent curve. *Communications on Pure and Applied Mathematics*, 75(4):667–766, 2022.

Song Mei, Andrea Montanari, and Phan-Minh Nguyen. A mean field view of the landscape of two-layer neural networks. *Proceedings of the National Academy of Sciences*, 115(33): E7665–E7671, 2018. ISSN 0027-8424. doi: 10.1073/pnas.1806579115. URL https://www.pnas.org/content/115/33/E7665.

Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of Machine Learning*. The MIT Press, 2012. ISBN 026201825X, 9780262018258.

Vidya Muthukumar, Kailas Vodrahalli, Vignesh Subramanian, and Anant Sahai. Harmless interpolation of noisy data in regression. *arXiv preprint arXiv:1903.09139[cs.LG]*, 2019.

Vidya Muthukumar, Adhyyan Narang, Vignesh Subramanian, Mikhail Belkin, Daniel Sahai, Hsu, and Anant Sahai. Classification vs regression in overparameterized regimes: Does the loss function matter? *arXiv preprint arXiv:2005.08054 [cs.LG]*, 2020.

Preetum Nakkiran, Gal Kaplun, Yamini Bansal, Tristan Yang, Boaz Barak, and Ilya Sutskever. Deep double descent: Where bigger models and more data hurt. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=B1g5sA4twr.

Brady Neal, Sarthak Mittal, Aristide Baratin, Vinayak Tantia, Matthew Scicluna, Simon Lacoste-Julien, and Ioannis Mitliagkas. A modern take on the bias-variance tradeoff in neural networks. In *ICML 2019 Workshop on Identifying and Understanding Deep Learning Phenomena*, 2019. URL `https://openreview.net/forum?id=B1guPVr2h4`.

Behnam Neyshabur, Ryota Tomioka, and Nathan Srebro. In search of the real inductive bias: On the role of implicit regularization in deep learning. *arXiv preprint arXiv:1412.6614*, 2014.

Behnam Neyshabur, Srinadh Bhojanapalli, David McAllester, and Nati Srebro. Exploring generalization in deep learning. In *Advances in Neural Information Processing Systems*, pp. 5949–5958, 2017a.

Behnam Neyshabur, Ryota Tomioka, Ruslan Salakhutdinov, and Nathan Srebro. Geometry of optimization and implicit regularization in deep learning. *arXiv:1705.03071 [cs.LG]*, 2017b.

Behnam Neyshabur, Zhiyuan Li, Srinadh Bhojanapalli, Yann LeCun, and Nathan Srebro. Towards understanding the role of over-parametrization in generalization of neural networks. *International Conference on Learning Representations (ICLR)*, 2019.

Roman Novak, Lechao Xiao, Jiri Hron, Jaehoon Lee, Alexander A Alemi, Jascha Sohl-Dickstein, and Samuel S Schoenholz. Neural tangents: Fast and easy infinite neural networks in python. In *International Conference on Learning Representations*, 2019.

Roman Novak, Jascha Sohl-Dickstein, and Samuel S Schoenholz. Fast finite width neural tangent kernel. In *International Conference on Machine Learning*, pp. 17018–17044. PMLR, 2022.

Chris Olah, Alexander Mordvintsev, and Ludwig Schubert. Feature visualization. *Distill*, 2017. doi: 10.23915/distill.00007. https://distill.pub/2017/feature-visualization.

Yann Ollivier. Riemannian metrics for neural networks i: feedforward networks. *Information and Inference: A Journal of the IMA*, 2015.

Guillermo Ortiz-Jiménez, Seyed-Mohsen Moosavi-Dezfooli, and Pascal Frossard. What can linearized neural networks actually say about generalization? *Advances in Neural Information Processing Systems*, 34, 2021.

Jonas Paccolat, Leonardo Petrini, Mario Geiger, Kevin Tyloo, and Matthieu Wyart. Geometric compression of invariant manifolds in neural networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2021(4):044001, 2021.

Razvan Pascanu and Yoshua Bengio. Revisiting natural gradient for deep networks. *arXiv preprint arXiv:1301.3584*, 2013.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An

imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates, Inc., 2019a.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in neural information processing systems*, pp. 8026–8037, 2019b.

Garima Pruthi, Frederick Liu, Satyen Kale, and Mukund Sundararajan. Estimating training data influence by tracing gradient descent. *Advances in Neural Information Processing Systems*, 33:19920–19930, 2020.

Nasim Rahaman, Aristide Baratin, Devansh Arpit, Felix Draxler, Min Lin, Fred Hamprecht, Yoshua Bengio, and Aaron Courville. On the spectral bias of neural networks. In *International Conference on Machine Learning*, pp. 5301–5310. PMLR, 2019.

Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In *NIPS*, 2007.

Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents. *arXiv preprint arXiv:2204.06125*, 2022.

Gaspar Rochette, Andre Manoel, and Eric W Tramel. Efficient per-example gradient computations in convolutional neural networks. *arXiv preprint arXiv:1912.06015*, 2019.

Olivier Roy and Martin Vetterli. The effective rank: A measure of effective dimensionality. In *2007 15th European Signal Processing Conference*, pp. 606–610. IEEE, 2007.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

Shiori Sagawa, Pang Wei Koh, Tatsunori B. Hashimoto, and Percy Liang. Distributionally robust neural networks. In *International Conference on Learning Representations*, 2020a. URL `https://openreview.net/forum?id=ryxGuJrFvS`.

Shiori Sagawa, Aditi Raghunathan, Pang Wei Koh, and Percy Liang. An investigation of why overparameterization exacerbates spurious correlations. In *International Conference on Machine Learning*, pp. 8346–8356. PMLR, 2020b.

Pedro Savarese, Itay Evron, Daniel Soudry, and Nathan Srebro. How do infinite width bounded norm networks look in function space? In *Proceedings of the 32nd Annual Conference on Learning Theory (COLT)*, volume PMLR 99, pp. 2667–2690, 2019.

Andrew M. Saxe, James L. Mcclelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural network. In *In International Conference on Learning Representations*, 2014.

Mark Schmidt, Nicolas Le Roux, and Francis Bach. Minimizing finite sums with the stochastic average gradient. *Mathematical Programming*, 162(1):83–112, 2017.

B. Schölkopf, S. Mika, C. J.C. Burges, P. Knirsch, K. R. Muller, G. Ratsch, and A. J. Smola. Input space versus feature space in kernel-based methods. *Trans. Neur. Netw.*, 10 (5):1000–1017, September 1999a. ISSN 1045-9227.

B. Schölkopf, J. Shawe-Taylor, AJ. Smola, and RC. Williamson. Kernel-dependent support vector error bounds. In *Artificial Neural Networks, 1999. ICANN 99*, volume 470 of *Conference Publications*, pp. 103–108. Max-Planck-Gesellschaft, IEEE, 1999b.

Nicol N Schraudolph. Fast curvature matrix-vector products. In *International Conference on Artificial Neural Networks*. Springer, 2001.

Andrew W Senior, Richard Evans, John Jumper, James Kirkpatrick, Laurent Sifre, Tim Green, Chongli Qin, Augustin Zidek, Alexander WR Nelson, Alex Bridgland, et al. Improved protein structure prediction using potentials from deep learning. *Nature*, 577(7792): 706–710, 2020.

Haozhe Shan and Blake Bordelon. Rapid feature evolution accelerates learning in neural networks. *arXiv preprint arXiv:2105.14301*, 2021.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.

Sidak Pal Singh and Dan Alistarh. Woodfisher: Efficient second-order approximations for model compression. In *Advances in Neural Information Processing Systems*, 2020.

Justin A. Sirignano and Konstantinos Spiliopoulos. Mean field analysis of neural networks: A law of large numbers. *SIAM J. Appl. Math.*, 80(2):725–752, 2020. doi: 10.1137/18M1192184. URL `https://doi.org/10.1137/18M1192184`.

Nati Srebro, Karthik Sridharan, and Ambuj Tewari. On the universality of online mirror descent. In *Advances in Neural Information Processing Systems 24*. 2011.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, 27, 2014.

Valentin Thomas, Fabian Pedregosa, Bart van Merriënboer, Pierre-Antoine Mangazol, Yoshua Bengio, and Nicolas Le Roux. Information matrices and generalization. *arXiv:1906.07774 [cs.LG]*, 2019.

Valentin Thomas, Fabian Pedregosa, Bart Merriënboer, Pierre-Antoine Manzagol, Yoshua Bengio, and Nicolas Le Roux. On the interplay between noise and curvature and its effect on optimization and generalization. In *International Conference on Artificial Intelligence and Statistics*, pp. 3503–3513. PMLR, 2020.

Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine*

*learning*, 2012.

Mariya Toneva, Alessandro Sordoni, Remi Tachet des Combes, Adam Trischler, Yoshua Bengio, and Geoffrey J. Gordon. An empirical study of example forgetting during deep neural network learning. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=BJlxm30cKm.

Catherine Wah, Steve Branson, Peter Welinder, Pietro Perona, and Serge Belongie. The caltech-ucsd birds-200-2011 dataset. 2011.

Chaoqi Wang, Roger Grosse, Sanja Fidler, and Guodong Zhang. Eigendamage: Structured pruning in the kronecker-factored eigenbasis. 2019.

Francis Williams, Matthew Trager, Daniele Panozzo, Claudio Silva, Denis Zorin, and Joan Bruna. Gradient dynamics of shallow univariate relu networks. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/file/1f6419b1cbe79c71410cb320fc094775-Paper.pdf.

Blake Woodworth, Suriya Gunasekar, Jason D. Lee, Edward Moroshko, Pedro Savarese, Itay Golan, Daniel Soudry, and Nathan Srebro. Kernel and rich regimes in overparametrized models. In Jacob Abernethy and Shivani Agarwal (eds.), *Proceedings of Thirty Third Conference on Learning Theory*, volume 125 of *Proceedings of Machine Learning Research*, pp. 3635–3673. PMLR, 09–12 Jul 2020. URL https://proceedings.mlr.press/v125/woodworth20a.html.

Abraham J Wyner, Matthew Olson, Justin Bleich, and David Mease. Explaining the success of adaboost and random forests as interpolating classifiers. *The Journal of Machine Learning Research*, 18(1):1558–1590, 2017.

Yuege Xie, Rachel Ward, Holger Rauhut, and Chou Hung-Hsu. Weighted optimization: better generalization by smoother interpolation. *arXiv preprint arXiv:2006.08495*, 2020.

Zhi-Qin John Xu, Yaoyu Zhang, and Yanyang Xiao. Training behavior of deep neural network in frequency domain. In Tom Gedeon, Kok Wai Wong, and Minho Lee (eds.), *Neural Information Processing*, pp. 264–274, Cham, 2019. Springer International Publishing. ISBN 978-3-030-36708-4.

Greg Yang and Edward J. Hu. Tensor programs IV: feature learning in infinite-width neural networks. In Marina Meila and Tong Zhang (eds.), *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pp. 11727–11737. PMLR, 2021. URL http://proceedings.mlr.press/v139/yang21c.html.

Greg Yang and Hadi Salman. A fine grained spectral perspective on neural networks. *arxiv preprint arXiv:1907.10599[cs.LG]*, 2019.

Lee Zamparo, Marc-Etienne Brunet, Thomas George, Sepideh Kharaghani, and Gintare Karolina Dziugaite. The dynamics of functional diversity throughout neural network training. *NeurIPS (Workshop)*, 2021.

Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. *ICLR*, 2017a.

Guodong Zhang, James Martens, and Roger Grosse. Fast convergence of natural gradient descent for overparameterized neural networks. *arXiv preprint arXiv:1905.10961*, 2019.

Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*, 2017b.

Xiao Zhang, Haoyi Xiong, and Dongrui Wu. Rethink the connections among generalization, memorization, and the spectral bias of dnns. In Zhi-Hua Zhou (ed.), *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pp. 3392–3398. International Joint Conferences on Artificial Intelligence Organization, 8 2021. doi: 10.24963/ijcai.2021/467. URL `https://doi.org/10.24963/ijcai.2021/467`. Main Track.

# Appendix A

---

# Fast Approximate Natural Gradient Descent in a Kronecker-factored Eigenbasis Supplementary material

## A.1. Kronecker product primer

In chapter 4 we make a heavy usage of the Kronecker product. This appendix gives some relevant properties of this tool.

**Definition 3** (Kronecker product)**.** Let $A$ be a $m \times n$ matrix with coefficients $a_{ij}$ and $B$ be a $p \times q$ matrix with coefficients $b_{ij}$. The Kronecker product $A \otimes B$ of $A$ and $B$ is the following $mp \times nq$ matrix:

$$A \otimes B \;=\; \begin{pmatrix} a_{11}B & a_{12}B & \cdots & a_{1n}B \\ a_{21}B & a_{22}B & \cdots & a_{2n}B \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}B & a_{m2}B & \cdots & a_{mn}B \end{pmatrix}$$

It corresponds to a tensor product with a particular reshaping so as to obtain a matrix (instead of a higher order tensor).

The following properties hold:

- $(A \otimes B)(C \otimes D) = AC \otimes BD$ when the matrix-matrix products $AC$ and $BD$ are defined.
- $(A \otimes B)^{-1} = A^{-1} \otimes B^{-1}$ when $A$ and $B$ are invertible matrices
- $(A \otimes B)^{\top} = A^{\top} \otimes B^{\top}$
- $\operatorname{rank}(A \otimes B) = \operatorname{rank}(A) \operatorname{rank}(B)$

- if $\lambda_A$ is a singular value of $A$ with corresponding singular vector $\mathbf{v}_A$ and $\lambda_B$ is a singular value of $B$ with corresponding singular vector $\mathbf{v}_B$ then $\lambda_A \lambda_B$ is a singular value of $A \otimes B$ with corresponding singular vector $\mathbf{v}_A \otimes \mathbf{v}_B$

# A.2. Proofs

## A.2.1. Proof that EKFAC does optimal diagonal rescaling in the KFE

**Lemma 1.** Let $G = \mathbb{E}\left[\nabla_\theta \nabla_\theta^\top\right]$ a real positive semi-definite matrix. And let $Q$ a given orthogonal matrix. Among diagonal matrices, diagonal matrix $D$ with diagonal entries $D_{ii} = \mathbb{E}\left[\left(Q^\top \nabla_\theta\right)_i^2\right]$ minimize approximation error $e = \left\|G - QDQ^\top\right\|_F$ (measured as Frobenius norm).

Proof. Since the Frobenius norm remains unchanged through multiplication by an orthogonal matrix we can write

$$\begin{aligned}
e^2 &= \left\|G - QDQ^\top\right\|_F^2 \\
&= \left\|Q^\top \left(G - QDQ^\top\right) Q\right\|_F^2 \\
&= \left\|Q^\top G Q - D\right\|_F^2 \\
&= \underbrace{\sum_i (Q^\top G Q - D)_{ii}^2}_{\text{diagonal}} + \underbrace{\sum_i \sum_{j \neq i} (Q^\top G Q)_{ij}^2}_{\text{off-diagonal}}
\end{aligned}$$

Since $D$ is diagonal, it does not affect the off-diagonal terms.

The squared diagonal terms all reach their minimum value 0 by setting $D_{ii} = \left(Q^\top G Q\right)_{ii}$ for all $i$:

$$\begin{aligned}
D_{ii} &= \left(Q^\top G Q\right)_{ii} \\
&= \left(Q^\top \mathbb{E}\left[\nabla_\theta \nabla_\theta^\top\right] Q\right)_{ii} \\
&= \left(\mathbb{E}\left[Q^\top \nabla_\theta \nabla_\theta^\top Q\right]\right)_{ii} \\
&= \left(\mathbb{E}\left[Q^\top \nabla_\theta \left(Q^\top \nabla_\theta\right)^\top\right]\right)_{ii} \\
&= \mathbb{E}\left[\left(Q^\top \nabla_\theta\right)_i^2\right] \text{ since } Q^\top \nabla_\theta \text{ is a vector}
\end{aligned}$$

We have thus shown that diagonal matrix $D$ with diagonal entries $D_{ii} = \mathbb{E}\left[\left(Q^\top \nabla_\theta\right)_i^2\right]$ minimize $e^2$. Since Frobenius norm $e$ is non-negative this implies that $D$ also minimizes $e$. □

**Theorem 1.** Let $G = \mathbb{E}\left[\nabla_\theta \nabla_\theta^\top\right]$ the matrix we want to approximate. Let $G_{\text{KFAC}} = A \otimes B$ the approximation of $G$ obtained by KFAC. Let $A = U_A S_A U_A^\top$ and $B = U_B S_B U_B^\top$ eigendecomposition of $A$ and $B$. The diagonal rescaling that EKFAC does in the Kronecker-factored Eigenbasis (KFE) $U_A \otimes U_B$ is optimal in the sense that it minimizes the Frobenius norm of the approximation error: among diagonal matrices $D$, approximation error $e = \left\|G - (U_A \otimes U_B) D (U_A \otimes U_B)^\top\right\|_F$ is minimized by the matrix with with diagonal entries $D_{ii} = s_i^* = \mathbb{E}\left[\left((U_A \otimes U_B)^\top \nabla_\theta\right)_i^2\right]$.

Proof. This follows directly by setting $Q = U_A \otimes U_B$ in Lemma 1. Note that the Kronecker product of two orthogonal matrices yields an orthogonal matrix.

□

**Theorem 2.** Let $G_{\text{KFAC}}$ the KFAC approximation of $G$ and $G_{\text{EKFAC}}$ the EKFAC approximation of $G$, we always have $\|G - G_{\text{EKFAC}}\|_F \leq \|G - G_{\text{KFAC}}\|_F$.

Proof. This follows trivially from Theorem 1 on the optimality of the EKFAC diagonal rescaling.

Since $D = S^*$, with the $S^* = \text{diag}(s^*)$ of EKFAC, minimizes $\left\|G - (U_A \otimes U_B) D (U_A \otimes U_B)^\top\right\|_F$, it implies that:

$$\begin{aligned}
\|G - (U_A \otimes U_B) S^* (U_A \otimes U_B)^\top\|_F &\leq \|G - (U_A \otimes U_B)(S_A \otimes S_B)(U_A \otimes U_B)^\top\|_F \\
\|G - G_{\text{EKFAC}}\|_F &\leq \|G - (U_A S_A U_A^\top) \otimes (U_B S_B U_B^\top)\|_F \\
\|G - G_{\text{EKFAC}}\|_F &\leq \|G - A \otimes B)\|_F \\
\|G - G_{\text{EKFAC}}\|_F &\leq \|G - G_{\text{KFAC}}\|_F
\end{aligned}$$

□

We have thus demonstrated that EKFAC yields a better approximation (more precisely: at least as good in Frobenius norm error) of $G$ than KFAC.

## A.2.2. Proof that $S_{ii} = \mathbb{E}\left[\left(U^\top \nabla_\theta\right)_i^2\right]$

**Theorem 3.** Let $G = \mathbb{E}\left[\nabla_\theta \nabla_\theta^\top\right]$ and $G = USU^\top$ its eigendecomposition.
Then $S_{ii} = \mathbb{E}\left[\left(U^\top \nabla_\theta\right)_i^2\right]$.

PROOF. Starting from eigendecomposition $G = USU^\top$ and the fact that $U$ is orthogonal so that $U^\top U = I$ we can write

$$\begin{aligned} G &= USU^\top \\ U^\top G U &= U^\top U S U^\top U \\ U^\top \underbrace{G}_{\mathbb{E}\left[\nabla_\theta \nabla_\theta^\top\right]} U &= S \end{aligned}$$

so that

$$\begin{aligned} S_{ii} &= \left(U^\top \mathbb{E}\left[\nabla_\theta \nabla_\theta^\top\right] U\right)_{ii} \\ &= \left(\mathbb{E}\left[U^\top \nabla_\theta \nabla_\theta^\top U\right]\right)_{ii} \\ &= \left(\mathbb{E}\left[U^\top \nabla_\theta \left(U^\top \nabla_\theta\right)\right]\right)_{ii} \\ &= \mathbb{E}\left[\left(U^\top \nabla_\theta \left(U^\top \nabla_\theta\right)\right)_{ii}\right] \\ &= \mathbb{E}\left[\left(U^\top \nabla_\theta\right)_i^2\right] \end{aligned}$$

where we obtained the last equality by observing that $U^\top \nabla_\theta$ is a vector and that the diagonal entries of the matrix $aa^\top$ for any vector $a$ are given by $a^2$ where the square operation is element-wise. □

## A.3. Residual network initialization

To train residual networks without using BN, one need to initialize them carefully, so we used the following procedure, denoting $n$ the fan-in of the layer:

(1) We use the He initialization for each layer directly preceded by a ReLU (He et al., 2015): $W \sim \mathcal{N}(0, 2/n)$, $b = 0$.

(2) Each layer not directly preceded by an activation function (for example the convolution in a skip connection) is initialized as: $W \sim \mathcal{N}(0, 1/n)$, $b = 0$. This can be derived from the He initialization, using the Identity as activation function.

(3) Inspired from Goyal et al. (2017), we divide the scale of the last convolution in each residual block by a factor 10: $W \sim \mathcal{N}(0, 0.2/n)$, $b = 0$. This not only helps preserving the variance through the network but also eases the optimization at the beginning of the training.

## A.4. Initialization of $\epsilon$

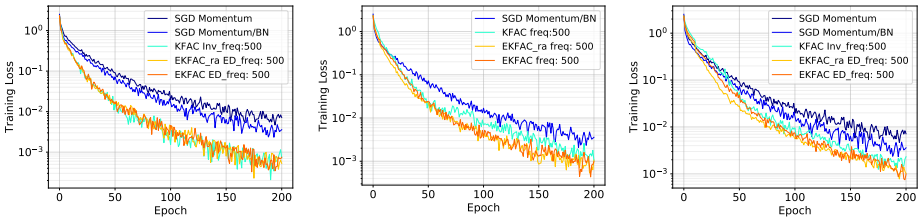
Both KFAC and EKFAC are very sensitive to the the damping parameter $\epsilon$. When EKFAC is applied to deep neural networks, we empirically observe that adding $\epsilon$ to the eigenvalues of both the activation covariance and gradient covariance matrices lead to better

optimization than using a fixed $\epsilon$ added directly on the eigenvalues in KFE space. KFAC uses also a similar initialization for $\epsilon$. By setting $\text{diag}(\epsilon') = \epsilon I + S_A \otimes \sqrt{\epsilon} I_{d_{\text{out}}} + \sqrt{\epsilon} I_{d_{\text{in}}} \otimes S_B$ , we can implement this initialization strategy directly in the KFE.

# A.5. Additional empirical results

## A.5.1. Impact of batch size

In this section, we evaluate the impact of the batch size on the optimization performances for KFAC and EKFAC. In Figure 1, we reports the training loss performance per epochs for different batch sizes for VGG11. We observe that the optimization gain of EKFAC with respect of KFAC diminishes as the batch size gets smaller.




**(a)** Batch size = 200 **(b)** Batch size = 500 **(c)** Batch size = 1000

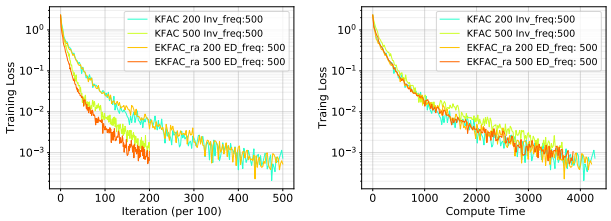
**Figure 1.** VGG11 on CIFAR-10. ED_freq (Inv_freq) corresponds to eigendecomposition (inverse) frequency. We perform model selections according to best training loss at each epoch. On this setting, we observe that the optimization gain of EKFAC with respect of KFAC diminishes as the batch size reduces.

In Figure 2, we look at the training loss per iterations and the training loss per computation time for different batch sizes, again on VGG11. EKFAC shows optimization benefits over KFAC as we increase the batch size (thus reducing the number of inverse/eigendecomposition per epoch). This gain does not translate in faster training in term of computation time in that setting. VGG11 is a relatively small network by modern standard and the SUA approximation remains computationally bearable on this model. We hypothesize that using smaller batches, KFAC can be updated often enough per epoch to have a reasonable estimation error while not paying a computational price too high.

In Figure 3, we perform a similar experiment on the Resnet34. In this setting, we observe that the optimization gain of EKFAC with respect of KFAC remains consistent across batch sizes.

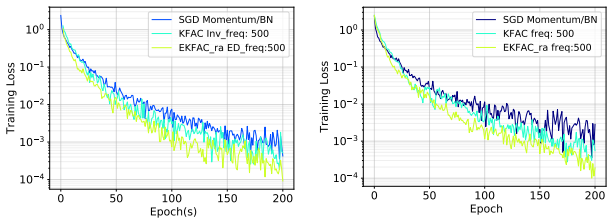
## A.5.2. Learning rate schedule

In this section we investigate the impact of a learning rate schedule on the optimization of EKFAC. We use a similar setting than the CIFAR-10 experiment, except that we decay




**(a)** Iterations **(b)** Wall-clock time

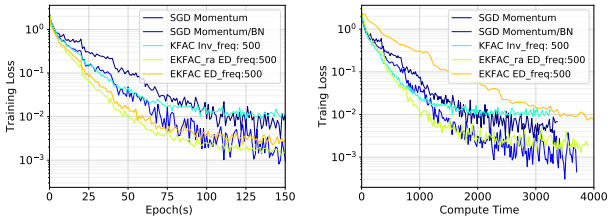
**Figure 2.** VGG11 on CIFAR-10. ED_freq (Inv_freq) corresponds to eigendecomposition (inverse) frequency. We perform model selections according to best training loss at each epoch. **(a)** Training loss per iterations for different batch sizes. **(b)** Training loss per computation time for different batch sizes. EKFAC shows optimization benefits over KFAC as we increases the batch size (thus reducing the number of inverse/eigendecomposition per epoch). This gain does not translate in faster training in terms of wall-clock time in that setting.




**(a)** Batch size = 200 **(b)** Batch size = 500

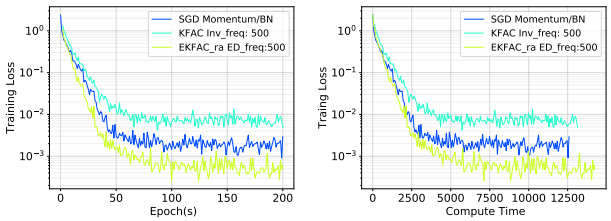
**Figure 3.** Resnet34 on CIFAR-10. ED_freq (Inv_freq) corresponds to eigendecomposition (inverse) frequency. We perform model selections according to best training loss at each epoch. In this setting, we observe that the optimization gain of EKFAC with respect of KFAC remains consistent across batch sizes.

the learning by 2 every 20 epochs. Figure 4 and 5 show that EKFAC still highlight some optimization benefit, relatively to the baseline, when combined with a learning rate schedule.




**(a)** Training loss **(b)** Wall-clock time

**Figure 4.** VGG11 on CIFAR10 using a learning rate schedule. ED_freq (Inv_freq) corresponds to eigendecomposition (inverse) frequency. In **(a)** and **(b)**, we report the performance of the hyperparameters reaching the lowest training loss for each epoch (to highlight which optimizers perform best given a fixed epoch budget).




**(a)** Training loss **(b)** Wall-clock time

**Figure 5.** Resnet34 on CIFAR10 using a learning rate schedule. ED_freq (Inv_freq) corresponds to eigendecomposition (inverse) frequency. In **(a)** and **(b)**, we report the performance of the hyper-parameters reaching the lowest training loss for each epoch (to highlight which optimizers perform best given a fixed epoch budget).

# Appendix B

---

# Implicit Regularization via Neural Feature Alignment Supplementary material

## B.1. Tangent Features and Geometry

We describe in more formal detail some of the notions introduced in Section 6.2 of the paper. We will consider general classes of vector-valued predictors:

$$\mathcal{F} = \{f_{\mathbf{w}} \colon \mathcal{X} \to \mathbb{R}^c \mid \mathbf{w} \in \mathcal{W}\}, \tag{B.1.1}$$

where the parameter space $\mathcal{W}$ is a finite dimensional manifold of dimension $P$ (typically $\mathbb{R}^P$). For multiclass classification, $f_{\mathbf{w}}$ outputs a score $f_{\mathbf{w}}(\mathbf{x})[y]$ for each class $y \in \{1 \cdots c\}$. Each function can also be viewed as a scalar function on $\mathcal{X} \times \mathcal{Y}$ where $\mathcal{Y} = \{1 \cdots c\}$ is the set of classes.

### B.1.1. Metric

We assume that $\mathbf{w} \to f_{\mathbf{w}}$ is a smooth mapping from $\mathcal{W}$ to $L^2(\rho, \mathbb{R}^c)$, where $\rho$ is some input data distribution. The inclusion $\mathcal{F} \subset L^2(\rho, \mathbb{R}^c)$ equips $\mathcal{F}$ with the $L^2$ scalar product and corresponding norm:

$$\langle f, g \rangle_\rho := \mathbb{E}_{\mathbf{x} \sim \rho}[f(\mathbf{x})^\top g(\mathbf{x})], \qquad \|f\|_\rho := \sqrt{\langle f, f \rangle_\rho} \tag{B.1.2}$$

The parameter space $\mathcal{W}$ inherits a **metric tensor** $g_{\mathbf{w}}$ by pull-back of the scalar product $\langle f, g \rangle_\rho$ on $\mathcal{F}$. That is, given $\boldsymbol{\zeta}, \boldsymbol{\xi} \in \mathcal{T}_{\mathbf{w}}\mathcal{W} \cong \mathbb{R}^P$ on the tangent space at $\mathbf{w}$ (Lang, 2012),

$$g_{\mathbf{w}}(\boldsymbol{\zeta}, \boldsymbol{\xi}) = \langle \partial_{\boldsymbol{\zeta}} f_{\mathbf{w}}, \partial_{\boldsymbol{\xi}} f_{\mathbf{w}} \rangle_\rho \tag{B.1.3}$$

where $\partial_{\boldsymbol{\zeta}} f_{\mathbf{w}} = \langle df_{\mathbf{w}}, \boldsymbol{\zeta} \rangle$ is the directional derivative in the direction of $\boldsymbol{\zeta}$. Concretely, in a given basis of $\mathbb{R}^P$, the metric is represented by the matrix of gradient second moments:

$$(g_{\mathbf{w}})_{pq} = \mathbb{E}_{\mathbf{x} \sim \rho}\left[\left(\frac{\partial f_{\mathbf{w}}(\mathbf{x})}{\partial w_p}\right)^\top \frac{\partial f_{\mathbf{w}}(\mathbf{x})}{\partial w_q}\right] \tag{B.1.4}$$

where $w_p$, $p = 1, \cdots, P$ are the parameter coordinates. The metric shows up by spelling out the line element $ds^2 := \|df_{\mathbf{w}}\|_\rho^2$, since we have,

$$\|df_{\mathbf{w}}\|_\rho^2 = \sum_{p,q=1}^{P} \langle \frac{\partial f_{\mathbf{w}}}{\partial w_p} dw_p, \frac{\partial f_{\mathbf{w}}}{\partial w_q} dw_q \rangle_\rho = \sum_{p,q=1}^{P} (g_{\mathbf{w}})_{pq}\, dw_p dw_q \tag{B.1.5}$$

### B.1.2. Tangent Kernels

This geometry has a dual description in function space in terms of *kernels*. The idea is to view the differential of the mapping $\mathbf{w} \to f_{\mathbf{w}}$ at each $\mathbf{w}$ as a map $df_{\mathbf{w}} \colon \mathcal{X} \times \mathcal{Y} \to \mathcal{T}_{\mathbf{w}}^*\mathcal{W} \cong \mathbb{R}^p$ defining (joined) features in the (co)tangent space. In a given basis, this yields the **tangent features** given by the function derivatives w.r.t the parameters,

$$\Phi_{w_p}(\mathbf{x})[y] := \frac{\partial f_{\mathbf{w}}(\mathbf{x})[y]}{\partial w_p} \tag{B.1.6}$$

The tangent feature map $\Phi_{\mathbf{w}}$ can be viewed as a function mapping each pair $(\mathbf{x}, y)$ to a vector in $\mathbb{R}^P$. It defines the so-called **tangent kernel** (Jacot et al., 2018) through the Euclidean dot product $\langle \cdot, \cdot \rangle$ in $\mathbb{R}^P$:

$$k_{\mathbf{w}}(\mathbf{x}, y; \tilde{\mathbf{x}}, y') = \langle \Phi_{\mathbf{w}}(\mathbf{x})[y], \Phi_{\mathbf{w}}(\tilde{\mathbf{x}})[y'] \rangle = \sum_{p=1}^{P} \Phi_{w_p}(\mathbf{x})[y] \Phi_{w_p}(\tilde{\mathbf{x}})[y'] \tag{B.1.7}$$

It induces an integral operator on $L^2(\rho, \mathbb{R}^c)$ acting as

$$(k_{\mathbf{w}} \triangleright f)(\mathbf{x})[y] = \langle k_{\mathbf{w}}(\mathbf{x}, y; \cdot), f \rangle \tag{B.1.8}$$

The metric tensor (B.1.4) is expressed in terms of the tangent features as $(g_{\mathbf{w}})_{pq} = \langle \Phi_{w_p}, \Phi_{w_q} \rangle_\rho$.

## B.1.3. Spectral Decomposition

The local metric tensor (as symmetric $P \times P$ matrix) and tangent kernel (as rank $P$ integral operator) share the same spectrum. More generally, let

$$g_{\mathbf{w}} = \sum_{j=1}^{P} \lambda_{\mathbf{w}j} \boldsymbol{v}_{\mathbf{w}j} \boldsymbol{v}_{\mathbf{w}j}^{\top} \tag{B.1.9}$$

be the eigenvalue decomposition of the positive (semi-)definite symmetric matrix (B.1.4), where $\boldsymbol{v}_{\mathbf{w}j}^{\top} \boldsymbol{v}_{\mathbf{w}j'} = \delta_{jj'}$. Assuming non-degeneracy, i.e $\lambda_{\mathbf{w}j} > 0$, let $u_{\mathbf{w}j}, j \in \{1 \cdots P\}$ be the functions in $L^2(\rho, \mathbb{R}^c)$ defined as:

$$u_{\mathbf{w}j}(\mathbf{x})[y] = \frac{1}{\sqrt{\lambda_{\mathbf{w}j}}} \boldsymbol{v}_{\mathbf{w}j}^{\top} \Phi_{\mathbf{w}}(\mathbf{x})[y] \tag{B.1.10}$$

The following result holds.

**Proposition B.1.1** (Spectral decomposition)**.** *The functions $(u_{j\mathbf{w}})_{j=1}^{P}$ form an orthonormal family in $L^2(\rho, \mathbb{R}^c)$. They are eigenfunctions of the tangent kernel as an integral operator, which admits the spectral decomposition:*

$$k_{\mathbf{w}}(\mathbf{x}, y; \tilde{\mathbf{x}}, y') = \sum_{j=1}^{P} \lambda_{\mathbf{w}j} \, u_{\mathbf{w}j}(\mathbf{x})[y] \, u_{\mathbf{w}j}(\tilde{\mathbf{x}})[y'] \tag{B.1.11}$$

*In particular metric tensor and tangent kernels share the same spectrum.*

PROOF. We first show orthonormality, i.e $\langle u_{\mathbf{w}j} u_{\mathbf{w}j'} \rangle_\rho = \delta_{jj'}$. We have indeed,

$$\langle u_{\mathbf{w}j}, u_{\mathbf{w}j'} \rangle_\rho = \frac{1}{\sqrt{\lambda_{\mathbf{w}j}\lambda_{\mathbf{w}j'}}} \sum_{p,q=1}^{P} (\boldsymbol{v}_{\mathbf{w}j})_p (\boldsymbol{v}_{\mathbf{w}j})_q \langle \Phi_{w_p}, \Phi_{w_q} \rangle_\rho \tag{B.1.12}$$

$$= \frac{1}{\sqrt{\lambda_{\mathbf{w}j}\lambda_{\mathbf{w}j'}}} \boldsymbol{v}_{\mathbf{w}j}^\top \, g_{\mathbf{w}} \, \boldsymbol{v}_{\mathbf{w}j'} \tag{B.1.13}$$

$$= \frac{1}{\lambda_{\mathbf{w}j}} \lambda_{\mathbf{w}j} \delta_{jj'} \tag{B.1.14}$$

$$= \delta_{jj'} \tag{B.1.15}$$

where we used the definition of the matrix $(g_{\mathbf{w}})_{pq}$ and its eigenvalue decomposition. Next, using the action (B.1.8) of the tangent kernel, we prove that the functions $u_{\mathbf{w}j}$ defined in (B.1.10) is an eigenfunction with eigenvalue $\lambda_{\mathbf{w}j}$:

$$(k_{\mathbf{w}} \triangleright u_{\mathbf{w}j})(\mathbf{x})[y] = \sum_{p=1}^{P} \Phi_{w_p}(\mathbf{x})[y] \langle \Phi_{w_p}, u_{\mathbf{w}j} \rangle_\rho \tag{B.1.16}$$

$$= \frac{1}{\sqrt{\lambda_{\mathbf{w}j}}} \sum_{p,q=1}^{P} (\boldsymbol{v}_{\mathbf{w}j})_q \, \Phi_{w_p}(\mathbf{x})[y] \langle \Phi_{w_p}, \Phi_{w_q} \rangle \tag{B.1.17}$$

$$= \frac{1}{\sqrt{\lambda_{\mathbf{w}j}}} \boldsymbol{v}_{\mathbf{w}j}^\top \, g_{\mathbf{w}} \, \Phi_{\mathbf{w}}(\mathbf{x})[y] \tag{B.1.18}$$

$$= \frac{1}{\sqrt{\lambda_{\mathbf{w}j}}} (\lambda_{\mathbf{w}j} \boldsymbol{v}_{\mathbf{w}j}^\top) \, \Phi_{\mathbf{w}}(\mathbf{x})[y] \tag{B.1.19}$$

$$= \lambda_{\mathbf{w}j} \frac{1}{\sqrt{\lambda_{\mathbf{w}j}}} \boldsymbol{v}_{\mathbf{w}j}^\top \, \Phi_{\mathbf{w}}(\mathbf{x})[y] \tag{B.1.20}$$

$$= \lambda_{\mathbf{w}j} \, u_{\mathbf{w}j} \tag{B.1.21}$$

Inserting the resolution of unity $\mathrm{Id}_P = \sum_{j=1}^{P} \boldsymbol{v}_{\mathbf{w}j} \boldsymbol{v}_{\mathbf{w}j}^\top$ in the expression (B.1.7) of the tangent kernel directly yields the spectral decomposition (B.1.11). □

## B.1.4. Sampled Versions

Given $n$ input samples $\mathbf{x}_1, \cdots \mathbf{x}_n$, any function $f \colon \mathcal{X} \to \mathbb{R}^c$ yields a vector $\boldsymbol{f} \in \mathbb{R}^{nc}$ obtained by concatenating the outputs $f(\mathbf{x}_i) \in \mathbb{R}^c$ of the $n$ input samples $\mathbf{x}_i$. The sample output scores $f_{\mathbf{w}}(\mathbf{x}_i)[y]$ thus yields $\boldsymbol{f}_{\mathbf{w}} \in \mathbb{R}^{nc}$; and the tangent features $\Phi_{w_p}(\mathbf{x}_i)[y]$ are represented as a $nc \times P$ matrix $\boldsymbol{\Phi}_{\mathbf{w}}$. Using this notation, (B.1.4) and (B.1.7) yield the sample covariance $P \times P$ matrix and kernel (Gram) $nc \times nc$ matrix:

$$\boldsymbol{G}_{\mathbf{w}} = \boldsymbol{\Phi}_{\mathbf{w}}^\top \boldsymbol{\Phi}_{\mathbf{w}}, \quad \boldsymbol{K}_{\mathbf{w}} = \boldsymbol{\Phi}_{\mathbf{w}} \boldsymbol{\Phi}_{\mathbf{w}}^\top \tag{B.1.22}$$

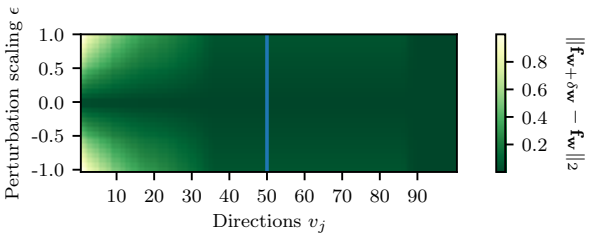



**Figure 1.** Variations of $\boldsymbol{f}_{\mathbf{w}}$ (evaluated on a test set) when perturbing the parameters in the directions given by the right singular vectors of the Jacobian (first 50 directions) or in randomly sampled directions (last 50 directions) on a VGG11 network trained for 10 epochs on CIFAR10. We observe that perturbations in most directions have almost no effect, except in those aligned with the top singular vectors.

The eigenvalue decompositions of $\boldsymbol{G}_{\mathbf{w}}$ and $\boldsymbol{K}_{\mathbf{w}}$ follow from the (SVD) of $\boldsymbol{\Phi}_{\mathbf{w}}$: assuming $P > nc$, we can write this SVD by indexing the singular values by a pair $J = (i, y)$ with $i = 1, \cdots n$ and $y = 1 \cdots c$ as

$$\boldsymbol{\Phi}_{\mathbf{w}} = \sum_{J=1}^{nc} \sqrt{\hat{\lambda}_{\mathbf{w}J}} \hat{\boldsymbol{u}}_{\mathbf{w}J} \hat{\boldsymbol{v}}_{\mathbf{w}J}^{\top} \tag{B.1.23}$$

Such decompositions summarize the predominant directions both in parameter and feature space, in the neighborhood of $\mathbf{w}$: a small variation $\delta\mathbf{w}$ induces the first order variation $\delta\boldsymbol{f}_{\mathbf{w}}$ of the function,

$$\delta\boldsymbol{f}_{\mathbf{w}} := \Phi_{\mathbf{w}}\delta\mathbf{w} = \sum_{J=1}^{nc} \sqrt{\hat{\lambda}_{\mathbf{w}J}} (\hat{\boldsymbol{v}}_{\mathbf{w}J}^{T}\delta\mathbf{w}) \hat{\boldsymbol{u}}_{\mathbf{w}J} \tag{B.1.24}$$

Fig. 1 illustrates this 'hierarchy' for a VGG11 network (Simonyan & Zisserman, 2015) trained for 10 epoches on CIFAR10 (Krizhevsky & Hinton, 2009). We observe that perturbations in most directions have almost no effect, except in those aligned with the top singular vectors. This is reflected by a strong anisotropy of the tangent kernel spectrum. Recent analytical results for wide random neural networks also point to such a pathological structure of the spectrum (Karakida et al., 2019a,b).

## B.1.5. Spectral Bias

**B.1.5.1. Proof of Lemma 6.2.1.** We consider parameter updates $\delta\mathbf{w}_{\mathrm{GD}} := -\eta\nabla_{\mathbf{w}}L$ for gradient descent w.r.t a loss $L := L(\boldsymbol{f}_{\mathbf{w}})$, which is a function of the vector $\boldsymbol{f}_{\mathbf{w}} \in \mathbb{R}^{nc}$ of sample output scores. We reformulate Lemma 6.2.1, extended to the multiclass setting.

**Proposition B.1.2** (Lemma 6.2.1 restated)**.** *The gradient descent function updates in first order Taylor approximation,* $\delta f_{GD}(\mathbf{x})[y] := \langle \delta\mathbf{w}_{GD}, \Phi_{\mathbf{w}}(\mathbf{x})[y] \rangle$*, decompose as,*

$$\delta f_{GD}(\mathbf{x})[y] = \sum_{j=1}^{P} \delta f_j \, u_{\mathbf{w}j}(\mathbf{x})[y], \qquad \delta f_j = -\eta\lambda_{\mathbf{w}j}(\boldsymbol{u}_{\mathbf{w}j}^{\top}\nabla_{\boldsymbol{f}_{\mathbf{w}}}L) \tag{B.1.25}$$

*where $u_{\mathbf{w}j}$ are the eigenfunctions (B.1.10) of the tangent kernel and $\boldsymbol{u}_{\mathbf{w}j} \in \mathbb{R}^{nc}$ are their corresponding sample vector.*

PROOF. Inserting the resolution of unity $\mathrm{Id}_P = \sum_{j=1}^P \boldsymbol{v}_{\mathbf{w}j}\boldsymbol{v}_{\mathbf{w}j}^\top$ in the expression for $\delta f_{\mathrm{GD}}$ yields

$$\delta f_{\mathrm{GD}}(\mathbf{x})[y] = \sum_{j=1}^P (\boldsymbol{v}_{\mathbf{w}j}^\top \delta \mathbf{w}_{\mathrm{GD}})\, \boldsymbol{v}_{\mathbf{w}j}^\top \Phi_{\mathbf{w}}(\mathbf{x})[y] \tag{B.1.26}$$

$$= \sum_{j=1}^P \sqrt{\lambda_{\mathbf{w}j}} (\boldsymbol{v}_{\mathbf{w}j}^\top \delta \mathbf{w}_{\mathrm{GD}})\, u_{\mathbf{w}j}(\mathbf{x})[y] \tag{B.1.27}$$

Next, by the chain rule $\nabla_{\mathbf{w}} L = \boldsymbol{\Phi}_{\mathbf{w}}^\top \nabla_{\boldsymbol{f}_{\mathbf{w}}} L$, so we can spell out:

$$\delta \mathbf{w}_{\mathrm{GD}} = -\eta \sum_{j=1}^P \sqrt{\lambda_{\mathbf{w}j}} (\boldsymbol{u}_{\mathbf{w}j}^\top \nabla_{\boldsymbol{f}_{\mathbf{w}}} L)\, \boldsymbol{v}_{\mathbf{w}j}, \tag{B.1.28}$$

which implies that $(\boldsymbol{v}_{\mathbf{w}j}^\top \delta \mathbf{w}_{\mathrm{GD}}) = \sqrt{\lambda_{\mathbf{w}j}} (\boldsymbol{u}_{\mathbf{w}j}^\top \nabla_{\boldsymbol{f}_{\mathbf{w}}} L)$. Substituting in (B.1.26) gives the desired result. □

The decomposition (B.1.25) has a *sampled* version in terms of tangent feature and kernel matrices. Using the notation of SVD (B.1.23), let $\hat{\lambda}_{\mathbf{w}J}, \hat{\boldsymbol{u}}_{\mathbf{w}j}$ and $\hat{\boldsymbol{v}}_{\mathbf{w}j}$ be correspond to the (non-zero) eigenvalues and eigenvectors of the sample covariance and kernel (B.1.22). We consider the tangent kernel **principal components**, defined as the functions

$$\hat{u}_{\mathbf{w}J}(\mathbf{x})[y] = \frac{1}{\sqrt{\lambda_{\mathbf{w}J}}} \langle \hat{\boldsymbol{v}}_{\mathbf{w}J}, \Phi_{\mathbf{w}}(\mathbf{x})[y] \rangle, \tag{B.1.29}$$

which form an orthonormal family for the in-sample scalar product $\langle f, g \rangle_{\mathrm{in}} = \sum_{i=1}^n f(\mathbf{x}_i) g(\mathbf{x}_i)$ and approximate the true kernel eigenfunctions (B.1.10) (e.g., Bengio et al., 2004; Braun, 2005). One can easily check from (B.1.23) that the vector $\hat{\boldsymbol{u}}_{\mathbf{w}J} \in \mathbb{R}^{nc}$ of sample outputs $\hat{u}(\mathbf{x}_i)[y]$ coincides with the $J$-th eigenvector of the tangent kernel matrix.

**Proposition B.1.3** (Sampled version of Prop B.1.2)**.** *The gradient descent function updates in first order Taylor approximation, $\delta f_{GD}(\mathbf{x})[y] := \langle \delta \mathbf{w}_{GD}, \Phi_{\mathbf{w}}(\mathbf{x})[y] \rangle$ decompose as,*

$$\delta f_{GD}(\mathbf{x})[y] = \sum_{j=1}^{nc} \delta f_J\, \hat{u}_{\mathbf{w}J}(\mathbf{x})[y], \qquad \delta f_J = -\eta \hat{\lambda}_{\mathbf{w}J} (\hat{\boldsymbol{u}}_{\mathbf{w}J}^\top \nabla_{\boldsymbol{f}_{\mathbf{w}}} L) \tag{B.1.30}$$

*in terms of the principal components (B.1.29) of the tangent kernel.*

PROOF. Same proof as for the previous Proposition, using the resolution of unity $\mathrm{Id}_{nc} = \sum_{J=1}^{nc} \hat{\boldsymbol{v}}_{\mathbf{w}J} \hat{\boldsymbol{v}}_{\mathbf{w}J}^\top$. □

**B.1.5.2. The Case of Linear Regression.** The previous Proposition gives a 'local' version of a classic decomposition of the training dynamics in linear regression (e.g., Advani & Saxe, 2017)). In such a setting, $f_{\mathbf{w}} = \langle \mathbf{w}, \Phi(\mathbf{x}) \rangle$ are linearly parametrized scalar functions

($c = 1$) and $L = \frac{1}{2}\|\boldsymbol{f}_{\mathbf{w}} - \boldsymbol{y}\|^2$. We denote by $\boldsymbol{\Phi} = \sum_{j=1}^{n} \hat{\lambda}_j \hat{\boldsymbol{u}}_j \hat{\boldsymbol{v}}_j^\top$ the $n \times P$ feature matrix and its SVD.

**Proposition B.1.4.** *Gradient descent of the squared loss yields the function iterates,*

$$f_{\mathbf{w}_t} = f_{\mathbf{w}^*} + (\mathrm{Id} - \eta K)^t (f_{\mathbf{w}_0} - f_{\mathbf{w}^*}) \tag{B.1.31}$$

*where Id is the identity map and $K$ is the operator acting on functions as $(K \triangleright f)(\mathbf{x}) = \sum_{i=1}^{n} k(\mathbf{x}, \mathbf{x}_i) f(\mathbf{x}_i)$ in terms of the kernel $k(\mathbf{x}, \tilde{\mathbf{x}}) = \langle \Phi(\mathbf{x}), \Phi(\tilde{\mathbf{x}}) \rangle$.*

Proof. The updates $\delta \mathbf{w}_{\mathrm{GD}} := -\eta \nabla_{\mathbf{w}} L$ induce the (exact) functional updates $\delta f_{\mathrm{GD}} = f_{\mathbf{w}_{t+1}} - f_{\mathbf{w}_t}$ given by

$$\delta f_{\mathrm{GD}}(\mathbf{x}) = -\eta \sum_{i=1}^{n} k(\mathbf{x}, \mathbf{x}_i)(f_{\mathbf{w}_t}(\mathbf{x}_i) - \boldsymbol{y}_i) \tag{B.1.32}$$

Substituting $\boldsymbol{y}_i = f_{\mathbf{w}^*}(\mathbf{x}_i)$ gives $f_{\mathbf{w}_{t+1}} - f_{\mathbf{w}^*} = (\mathrm{id} - \eta K)(f_{\mathbf{w}_t} - f_{\mathbf{w}^*})$. Equ. B.1.31 follows by induction. □

**Lemma B.1.5.** *The kernel principal components $\hat{u}_j(\mathbf{x}) = \frac{1}{\sqrt{\hat{\lambda}_j}} \langle \hat{\boldsymbol{v}}_j, \Phi_{\mathbf{w}}(\mathbf{x}) \rangle$ are eigenfunctions of the operator $K$ with corresponding eigenvalues $\hat{\lambda}_j$.*

Proof. By inserting $Id_n = \sum_j \hat{\boldsymbol{v}}_j \hat{\boldsymbol{v}}_j^\top$ in the expression of the kernel, one can write $k(\mathbf{x}, \mathbf{x}_i) = \sum_{j=1}^{n} \hat{u}_j(\mathbf{x}) \hat{u}_j(\mathbf{x}_i)$. Subsituting in the definition of $K$ and using the orthonormality of $\hat{u}_j$ for the in-sample scalar product yield $K \triangleright \hat{u}_j = \hat{\lambda}_j \hat{u}_j$. □

Together with(B.1.31), this directly leads to the decoupling of the training dynamics in the basis of kernel principal components.

**Proposition B.1.6** (Spectral Bias for Linear Regression)**.** *By initializing $\mathbf{w}_0 = \boldsymbol{\Phi}^\top \boldsymbol{\alpha}_0$ in the span of the features, the function iterates in (B.1.31) uniquely decompose as,*

$$f_{\mathbf{w}_t}(\mathbf{x}) = \sum_{j=1}^{n} f_{jt} \hat{u}_j(\mathbf{x}), \quad f_{jt} = f_j^* + (1 - \eta\lambda_j)^t (f_{j0} - f_j^*) \tag{B.1.33}$$

*where $f_j^*$ are the coefficients of the (mininum $\ell_2$-norm) interpolating solution.*

This standard result shows how each independant mode labelled by $j$ has its own linear convergence rate For example setting $\eta = 1/\lambda_1$, this gives $f_{jt} - f_j^* \propto e^{-t/\tau_j}$, where $\tau_j = -\log(1 - \frac{\lambda_j}{\lambda_1})$ is the time constant (number of iterations) for the mode $j$. Top modes $f_j^*$ of the target function are learned faster than low modes.

In linearized regimes where deep learning reduces to kernel regression (Jacot et al., 2018; Du et al., 2019b; Allen-Zhu et al., 2019), one can dwell further the nature of such a bias by analyzing the eigenfunctions of the neural tangent kernel (e.g., Yang & Salman, 2019). As a simple example, for a randomly initialized MLP on 1D uniform data, Fig. 2 shows the Fourier decomposition of such eigenfunctions, ranked in nonincreasing order of the eigenvalues. We observe that eigenfunctions with increasing index $j$ (hence decreasing eigenvalues) correspond to modes with increasing Fourier frequency, with a remarkable alignment with Fourier modes

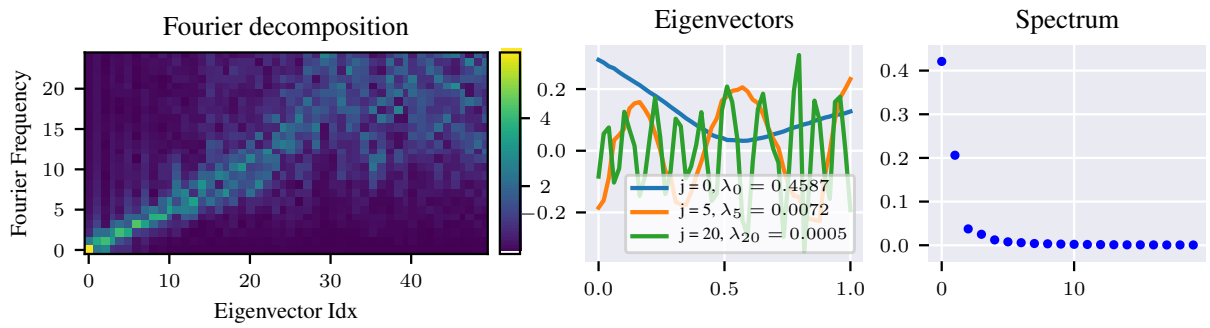



**Figure 2.** Eigendecomposition of the tangent kernel matrix of a random 6-layer deep 256-unit wide MLP on 1D uniform data (50 equally spaced points in [0,1]). **(left)** Fourier decomposition ($y$-axis for frequency, colorbar for magnitude) of each eigenvector ($x$-axis), ranked in nonincreasing order of the eigenvalues. We observe that eigenvectors with increasing index $j$ (hence decreasing eigenvalues) correspond to modes with increasing Fourier frequency. **(middle)** Plot of the $j$-th eigenvectors with $j \in \{0, 5, 20\}$ and **(right)** distribution of eigenvalues. We note the fast decay (e.g $\lambda_{10}/\lambda_1 \approx 4‰$).

for the first half of the spectrum. This in line with observations (e.g., Rahaman et al., 2019) that deep networks tend to prioritize learning low frequency modes during training.

## B.2. Complexity Bounds

In this section, we spell out details and proofs for the content of Section 6.4.

### B.2.1. Rademacher Complexity

Given a family $\mathcal{G} \subset \mathbb{R}^{\mathcal{Z}}$ of real-valued functions on a probability space $(\mathcal{Z}, \rho)$, the empirical Rademacher complexity of $\mathcal{G}$ with respect to a sample $\mathcal{S} = \{\mathbf{z}_1, \cdots \mathbf{z}_n\} \sim \rho^n$ is defined as (Mohri et al., 2012):

$$\widehat{\mathcal{R}}_{\mathcal{S}}(\mathcal{G}) = \mathbb{E}_{\boldsymbol{\sigma} \in \{\pm 1\}^n} \left[ \sup_{g \in \mathcal{G}} \frac{1}{n} \sum_{i=1}^{n} \sigma_i g(\mathbf{z}_i) \right], \tag{B.2.1}$$

where the expectation is over $n$ i.i.d uniform random variables $\sigma_1, \cdots \sigma_n \in \{\pm 1\}$. For any $n \geq 1$, the Rademacher complexity with respect to samples of size $n$ is then $\mathcal{R}_n(\mathcal{G}) = \mathbb{E}_{\mathcal{S} \sim \rho^n} \widehat{\mathcal{R}}_{\mathcal{S}}(\mathcal{G})$.

### B.2.2. Generalization Bounds

Generalization bounds based on Rademacher complexity are standard (Bartlett et al., 2017; Mohri et al., 2012). We give here one instance of such a bound, relevant for classification task.

**Setup.** We consider a family $\mathcal{F}$ of functions $f_{\mathbf{w}} \colon \mathcal{X} \to \mathbb{R}^c$ that output a score or probability $f_{\mathbf{w}}(\mathbf{x})[y]$ for each class $y \in \{1 \cdots c\}$ (we take $c = 1$ for binary classification). The task

is to find a predictor $f_{\mathbf{w}} \in \mathcal{F}$ with small expected classification error, which can be expressed e.g. as

$$L_0(f_{\mathbf{w}}) = \mathbb{P}_{(\mathbf{x},y)\sim\rho}\left\{\mu(f_{\mathbf{w}}(\mathbf{x}), y) < 0\right\} \tag{B.2.2}$$

where $\mu(f(\mathbf{x}), y)$ denotes the **margin**,

$$\mu(f(\mathbf{x}), y) = \begin{cases} f(\mathbf{x})y & \text{binary case} \\ f(\mathbf{x})[y] - \max_{y' \neq y} f(\mathbf{x})[y'] & \text{multiclass case} \end{cases} \tag{B.2.3}$$

**Margin Bound.** We consider the **margin loss**,

$$\ell_\gamma(f_{\mathbf{w}}(\mathbf{x}), y)) = \phi_\gamma(\mu(f_{\mathbf{w}}(\mathbf{x}), y)) \tag{B.2.4}$$

where $\gamma > 0$, and $\phi_\gamma$ is the **ramp** function: $\phi_\gamma(u) = 1$ if $u \leq 0$, $\phi(u) = 0$ if $u > \gamma$ and $\phi(u) = 1 - u/\gamma$ otherwise. We have the following bound for the expected error (B.2.2). With probability at least $1 - \delta$ over the draw $\mathcal{S} = \{\mathbf{z}_i = (\mathbf{x}_i, y_i)\}_{i=1}^n$ of size $n$, the following holds for all $f_{\mathbf{w}} \in \mathcal{F}$ (Mohri et al., 2012, Theorems 4.4.and 8.1):

$$L_0(f_{\mathbf{w}}) \leq \widehat{L}_\gamma(f_{\mathbf{w}}) + 2\widehat{\mathcal{R}}_{\mathcal{S}}(\ell_\gamma(\mathcal{F}, \cdot)) + 3\sqrt{\frac{\log \frac{2}{\delta}}{2n}} \tag{B.2.5}$$

where $\widehat{L}_\gamma(f_{\mathbf{w}}) = \frac{1}{n}\sum_{i=1}^n \ell_\gamma(f_{\mathbf{w}}(\mathbf{x}_i), y_i)$ is the empirical margin error and $\ell_\gamma(\mathcal{F}, \cdot)$ is the **loss class**,

$$\ell_\gamma(\mathcal{F}, \cdot) = \{(\mathbf{x}, y) \mapsto \ell_\gamma(f_{\mathbf{w}}(\mathbf{x}), y) \,|\, f_{\mathbf{w}} \in \mathcal{F}\} \tag{B.2.6}$$

For binary classifiers, because $\phi_\gamma$ is $1/\gamma$-Lipschitz, we have in addition

$$\mathcal{R}_{\mathcal{S}}(\ell_\gamma(\mathcal{F}, \cdot)) \leq \frac{1}{\gamma}\mathcal{R}_{\mathcal{S}}(\mathcal{F}) \tag{B.2.7}$$

by Talagrand's contraction lemma (Ledoux & Talagrand, 2013) (see e.g. Mohri et al., 2012, lemma 4.2 for a detailed proof).

## B.2.3. Complexity Bounds: Proofs

We first derive standard bounds for the linear classes of scalar functions,

$$\mathcal{F}_{M_A}^A = \{f_{\mathbf{w}} \colon \mathbf{x} \mapsto \langle \mathbf{w}, \Phi(\mathbf{x})\rangle \mid \|\mathbf{w}\|_A \leq M_A\} \tag{B.2.8}$$

**Proposition B.2.1.** *The empirical Rademacher complexity of $\mathcal{F}_{M_A}^A$ is bounded as,*

$$\widehat{\mathcal{R}}_{\mathcal{S}}(\mathcal{F}_{M_A}^A) \leq (M_A/n)\sqrt{\mathrm{Tr}\boldsymbol{K}_A} \tag{B.2.9}$$

*where $(\boldsymbol{K}_A)_{ij} = k_A(\mathbf{x}_i, \mathbf{x}_j)$ is the kernel matrix associated to the rescaled features $A^{-1}\Phi$.*

PROOF. We use the notation of Section 6.4. For given Rademacher variables $\boldsymbol{\sigma} \in \{\pm 1\}^n$, we have,

$$\begin{aligned}
\sup_{f \in \mathcal{F}^A_{M_A}} \sum_{i=1}^n \sigma_i f(\mathbf{x}_i) &= \sup_{\|\mathbf{w}\|_A \leq M_A} \sum_{i=1}^n \sigma_i \langle \mathbf{w}, \Phi(\mathbf{x}_i) \rangle \\
&= \sup_{\|A^\top \mathbf{w}\|_2 \leq M_A} \sum_{i=1}^n \sigma_i \langle A^\top \mathbf{w}, A^{-1}\Phi(\mathbf{x}_i) \rangle \\
&= \sup_{\|\tilde{\mathbf{w}}\|_2 \leq M_A} \langle \tilde{\mathbf{w}}, \sum_{i=1}^n \sigma_i A^{-1}\Phi(\mathbf{x}_i) \rangle \\
&= M_A \left\| \sum_{i=1}^n \sigma_i A^{-1}\Phi(\mathbf{x}_i) \right\|_2 \\
&= M_A \sqrt{\boldsymbol{\sigma}^\top \boldsymbol{K}_A \boldsymbol{\sigma}} \qquad \text{(B.2.10)}
\end{aligned}$$

From (B.2.10) and the definition (6.4.1) we obtain:

$$\begin{aligned}
\widehat{\mathcal{R}}_{\mathcal{S}}(\mathcal{F}^A_{M_A}) &= \frac{M_A}{n} \mathbb{E}_{\boldsymbol{\sigma}} \left[ \sqrt{\boldsymbol{\sigma}^\top \boldsymbol{K}_A \boldsymbol{\sigma}} \right] \\
&\leq \frac{M_A}{n} \sqrt{\mathbb{E}_{\boldsymbol{\sigma}} \left[ \boldsymbol{\sigma}^\top \boldsymbol{K}_A \boldsymbol{\sigma} \right]} \\
&\leq \frac{M_A}{n} \sqrt{\mathrm{Tr} \boldsymbol{K}_A} \qquad \text{(B.2.11)}
\end{aligned}$$

where we used Jensen's inequality to pass $\mathbb{E}_\sigma$ under the root, and that $\mathbb{E}[\sigma_i] = 0$ and $\sigma_i^2 = 1$ for all $i$. □

We now extend the result to the families (6.4.4) of learning flows:

$$\mathcal{F}^{\boldsymbol{A}}_{\boldsymbol{m}} = \{ f_{\mathbf{w}} \colon \mathbf{x} \mapsto \textstyle\sum_t \langle \delta \mathbf{w}_t, \Phi(\mathbf{x}) \rangle \mid \|\delta \mathbf{w}_t\|_{A_t} \leq m_t \} \qquad \text{(B.2.12)}$$

**Theorem B.2.2** (Theorem 6.4.1 restated)**.** *The empirical Rademacher complexity of $\mathcal{F}^{\boldsymbol{A}}_{\boldsymbol{m}}$ is bounded as,*

$$\widehat{\mathcal{R}}_{\mathcal{S}}(\mathcal{F}^{\boldsymbol{A}}_{\boldsymbol{m}}) \leq \textstyle\sum_t (m_t/n) \sqrt{\mathrm{Tr} \boldsymbol{K}_{A_t}} \qquad \text{(B.2.13)}$$

*where $(\boldsymbol{K}_{A_t})_{ij} = k_{A_t}(\mathbf{x}_i, \mathbf{x}_j)$ is the kernel matrix associated to the rescaled features $A_t^{-1}\Phi$.*

PROOF. This is simple extension of the previous proof:

$$\begin{aligned}
\sup_{f \in \mathcal{F}^{\boldsymbol{A}}_{\boldsymbol{m}}} \sum_{i=1}^n \sigma_i f(\mathbf{x}_i) &= \sup_{\|\delta \mathbf{w}_t\|_{A_t} \leq m_t} \sum_{i=1}^n \sigma_i \sum_t \langle \delta \mathbf{w}_t, \Phi(\mathbf{x}_i) \rangle \\
&= \sum_t \sup_{\|\tilde{\delta} \mathbf{w}_t\|_2 \leq m_t} \langle \tilde{\delta} \mathbf{w}_t, \sum_{i=1}^n \sigma_i A_t^{-1} \Phi(\mathbf{x}_i) \rangle \\
&= \sum_t m_t \sqrt{\boldsymbol{\sigma}^\top \boldsymbol{K}_{A_t} \boldsymbol{\sigma}} \qquad \text{(B.2.14)}
\end{aligned}$$

and we conclude as in (B.2.11). □

Finally, we note that the same result can be formulated in terms of an evolving feature map $\Phi_t = A_t^{-1}\Phi$ with kernel $k_t(\mathbf{x}, \tilde{\mathbf{x}}) = \langle \Phi_t(\mathbf{x}), \Phi_t(\tilde{\mathbf{x}})\rangle$ In fact by reparametrization invariance, the function updates can also be written as $\delta f_{\mathbf{w}_t}(\mathbf{x}) = \langle \tilde{\delta}\mathbf{w}_t, \Phi_t(\mathbf{x})\rangle$ where $\tilde{\delta}\mathbf{w}_t = A_t^\top \delta \mathbf{w}_t$. The function class (6.4.4) can equivalently be written as $\mathcal{F}_{\boldsymbol{m}}^{\boldsymbol{A}} = \mathcal{F}_{\boldsymbol{m}}^{\boldsymbol{\Phi}}$ where $\boldsymbol{\Phi}$ denotes a fixed sequence of feature maps, $\boldsymbol{\Phi} = \{\Phi_t\}_t$ and

$$\mathcal{F}_{\boldsymbol{m}}^{\boldsymbol{\Phi}} = \{f_{\mathbf{w}} \colon \mathbf{x} \mapsto \textstyle\sum_t \langle \tilde{\delta}\mathbf{w}_t, \Phi_t(\mathbf{x})\rangle \,|\, \|\tilde{\delta}\mathbf{w}_t\|_2 \leq m_t\} \tag{B.2.15}$$

In this formulation, the result (B.2.13) is expressed as,

$$\widehat{\mathcal{R}}_{\mathcal{S}}(\mathcal{F}_{\boldsymbol{m}}^{\boldsymbol{\Phi}}) \leq \textstyle\sum_t (m_t/n)\sqrt{\mathrm{Tr}\boldsymbol{K}_t} \tag{B.2.16}$$

where $(\boldsymbol{K}_t)_{ij} = k_t(\mathbf{x}_i, \tilde{\mathbf{x}_j})$ is the kernel matrix associated to the feature map $\Phi_t$.

## B.2.4. Bounds for Multiclass Classification

The generalization bound (B.2.5) is based on the **margin loss class** (B.2.6). In this section, we show how to bound $\widehat{\mathcal{R}}_{\mathcal{S}}(\ell_\gamma(\mathcal{F}, \cdot))$ in terms of tangent kernels for the original class $\mathcal{F}$ of functions $f_{\mathbf{w}} \colon \mathcal{X} \to \mathbb{R}^c$ instead. Although the proof is adapted from standard techniques, to our knowledge Lemma B.2.3 and Theorem B.2.1 below are new results. In what follows, we denote by $\mu_{\mathcal{F}}$ the margin class,

$$\mu_{\mathcal{F}} = \{(\mathbf{x}, y) \to \mu(f_{\mathbf{w}}(\mathbf{x}), y) \,|\, f_{\mathbf{w}} \in \mathcal{F}\} \tag{B.2.17}$$

where $\mu(f_{\mathbf{w}}(\mathbf{x}), y))$ is the margin (B.2.3). We also define, for each $y \in \{1 \cdots c\}$,

$$\mathcal{F}_y = \{\mathbf{x} \mapsto f_{\mathbf{w}}(\mathbf{x})[y] \,|\, f_{\mathbf{w}} \in \mathcal{F}\}, \quad \mu_{\mathcal{F},y} = \{\mathbf{x} \mapsto \mu(f_{\mathbf{w}}(\mathbf{x}), y) \,|\, f_{\mathbf{w}} \in \mathcal{F}\} \tag{B.2.18}$$

**Lemma B.2.3.** *The following inequality holds:*

$$\widehat{\mathcal{R}}_{\mathcal{S}}(\ell_\gamma(\mathcal{F}, \cdot)) \leq \frac{c}{\gamma} \sum_{y=1}^{c} \widehat{\mathcal{R}}_{\mathcal{S}}(\mathcal{F}_y) \tag{B.2.19}$$

Proof. We first follow the first steps of the proof of (Mohri et al., 2012, Theorem 8.1) to show that

$$\widehat{\mathcal{R}}_{\mathcal{S}}(\ell_\gamma(\mathcal{F}, \cdot)) \leq \frac{1}{\gamma} \sum_{y=1}^{c} \widehat{\mathcal{R}}_{\mathcal{S}}(\mu_{\mathcal{F},y}) \tag{B.2.20}$$

We reproduce these steps here for completeness: first, it follows from the $1/\gamma$-Lipschitzness of the ramp loss $\phi_\gamma$ in (B.2.4) and Talagrand's contraction lemma (Mohri et al., 2012, lemma 4.2) that

$$\widehat{\mathcal{R}}_{\mathcal{S}}(\ell_\gamma(\mathcal{F}, \cdot)) \leq \frac{1}{\gamma} \widehat{\mathcal{R}}_{\mathcal{S}}(\mu_{\mathcal{F}}) \tag{B.2.21}$$

Next, we write

$$\begin{aligned}\widehat{\mathcal{R}}_{\mathcal{S}}(\mu_{\mathcal{F}}) &:= \frac{1}{n}\mathbb{E}_{\boldsymbol{\sigma}}\left[\sup_{f_{\mathbf{w}}\in\mathcal{F}}\sum_{i=1}^{n}\sigma_i\mu(f_{\mathbf{w}}(\mathbf{x}_i),y_i)\right]\\ &= \frac{1}{n}\mathbb{E}_{\boldsymbol{\sigma}}\left[\sup_{f_{\mathbf{w}}\in\mathcal{F}}\sum_{i=1}^{n}\sigma_i\sum_{y=1}^{c}\mu(f_{\mathbf{w}}(\mathbf{x}_i),y)\,\delta_{y,y_i}\right]\\ &= \frac{1}{n}\sum_{y=1}^{c}\mathbb{E}_{\boldsymbol{\sigma}}\left[\sup_{f_{\mathbf{w}}\in\mathcal{F}}\sum_{i=1}^{n}\sigma_i\mu(f_{\mathbf{w}}(\mathbf{x}_i),y)\,\delta_{y,y_i}\right] \qquad \text{(B.2.22)}\end{aligned}$$

where $\delta_{y,y_i}=1$ if $y=y_i$ and 0 otherwise; the second inequality follows from the sub-additivity of sup. Substituting $\delta_{y,y_i}=\frac{1}{2}(\epsilon_i+\frac{1}{2})$ where $\epsilon_i=2\delta_{y,y_i}-1\in\{\pm1\}$, we obtain

$$\begin{aligned}\widehat{\mathcal{R}}_{\mathcal{S}}(\mu_{\mathcal{F}}) &\leq \frac{1}{2n}\sum_{y=1}^{c}\mathbb{E}_{\boldsymbol{\sigma}}\left[\sup_{f_{\mathbf{w}}\in\mathcal{F}}\sum_{i=1}^{n}(\epsilon_i\sigma_i)\mu(f_{\mathbf{w}}(\mathbf{x}_i),y)\right]+\frac{1}{2n}\sum_{y=1}^{c}\mathbb{E}_{\boldsymbol{\sigma}}\left[\sup_{f_{\mathbf{w}}\in\mathcal{F}}\sum_{i=1}^{n}\sigma_i\mu(f_{\mathbf{w}}(\mathbf{x}_i),y)\right]\\ &= \sum_{y=1}^{c}\frac{1}{n}\mathbb{E}_{\boldsymbol{\sigma}}\left[\sup_{f_{\mathbf{w}}\in\mathcal{F}}\sum_{i=1}^{n}\sigma_i\mu(f_{\mathbf{w}}(\mathbf{x}_i),y)\right]\\ &= \sum_{y=1}^{c}\widehat{\mathcal{R}}_{\mathcal{S}}(\mu_{\mathcal{F},y}) \qquad \text{(B.2.23)}\end{aligned}$$

Together with (B.2.21), this leads to (B.2.20).

Now, spelling out $\mu(f_{\mathbf{w}}(\mathbf{x}_i,y))$ gives

$$\begin{aligned}\widehat{\mathcal{R}}_{\mathcal{S}}(\mu_{\mathcal{F},y}) &= \frac{1}{n}\mathbb{E}_{\boldsymbol{\sigma}}\left[\sup_{f_{\mathbf{w}}\in\mathcal{F}}\sum_{i=1}^{n}\sigma_i(f_{\mathbf{w}}(\mathbf{x}_i)[y]-\max_{y'\neq y}f_{\mathbf{w}}(\mathbf{x}_i)[y'])\right]\\ &= \widehat{\mathcal{R}}_{\mathcal{S}}(\mathcal{F}_y)+\frac{1}{n}\mathbb{E}_{\boldsymbol{\sigma}}\left[\sup_{f_{\mathbf{w}}\in\mathcal{F}}\sum_{i=1}^{n}(-\sigma_i)\max_{y'\neq y}f_{\mathbf{w}}(\mathbf{x}_i)[y']\right]\\ &= \widehat{\mathcal{R}}_{\mathcal{S}}(\mathcal{F}_y)+\frac{1}{n}\mathbb{E}_{\boldsymbol{\sigma}}\left[\sup_{f_{\mathbf{w}}\in\mathcal{F}}\sum_{i=1}^{n}\sigma_i\max_{y'\neq y}f_{\mathbf{w}}(\mathbf{x}_i)[y']\right]\\ &\leq \widehat{\mathcal{R}}_{\mathcal{S}}(\mathcal{F}_y)+\widehat{\mathcal{R}}_{\mathcal{S}}(\mathcal{G}_y) \qquad \text{(B.2.24)}\end{aligned}$$

where $\mathcal{G}_y=\{\max\{f_{y'}:y'\neq y\}\,|\,f_{y'}\in\mathcal{F}_{y'}\}$. Now (Mohri et al., 2012, lemma 8.1) show that $\widehat{\mathcal{R}}_{\mathcal{S}}(\mathcal{G}_y)\leq\sum_{y'\neq y}\widehat{\mathcal{R}}_{\mathcal{S}}(\mathcal{F}_{y'})$. This leads to

$$\begin{aligned}\sum_{y=1}^{c}\widehat{\mathcal{R}}_{\mathcal{S}}(\mu_{\mathcal{F},y}) &\leq \sum_{y=1}^{c}\widehat{\mathcal{R}}_{\mathcal{S}}(\mathcal{F}_y)+\sum_{y=1}^{c}\sum_{\substack{y'=1\\y'\neq y}}^{c}\widehat{\mathcal{R}}_{\mathcal{S}}(\mathcal{F}_{y'})\\ &= \sum_{y=1}^{c}\widehat{\mathcal{R}}_{\mathcal{S}}(\mathcal{F}_y)+(c-1)\sum_{y=1}^{c}\widehat{\mathcal{R}}_{\mathcal{S}}(\mathcal{F}_y)\\ &= c\sum_{y=1}^{c}\widehat{\mathcal{R}}_{\mathcal{S}}(\mathcal{F}_y) \qquad \text{(B.2.25)}\end{aligned}$$

Substituting in (B.2.20) finishes the proof. □

In the linear case, this results leads to analogous theorems as in B.2.3 in the multiclass setting. For example, considering the linear families of functions $\mathcal{X} \to \mathbb{R}^c$,

$$\mathcal{F}^A_{M_A} = \{\mathbf{x} \mapsto f_{\mathbf{w}}(\mathbf{x})[y] := \langle \mathbf{w}, \Phi(\mathbf{x})[y] \rangle \mid \|\mathbf{w}\|_A \leq M_A\} \tag{B.2.26}$$

where $(\mathbf{x}, y) \mapsto \Phi(\mathbf{x})[y]$ is some joint feature map, we have the following

**Theorem B.2.4.** *The emp. Rademacher complexity of the margin loss class $\ell_\gamma(\mathcal{F}^A_{M_A}, \cdot)$ is bounded as,*

$$\widehat{\mathcal{R}}_{\mathcal{S}}(\ell_\gamma(\mathcal{F}^A_{M_A}, \cdot)) \leq (c^{3/2} M_A/\gamma n)\sqrt{\mathrm{Tr}\boldsymbol{K}_A} \tag{B.2.27}$$

*where $(\boldsymbol{K}_A)^{yy'}_{ij}$ is the kernel $nc \times nc$ matrix associated to the rescaled features $A^{-1}\Phi(\mathbf{x})[y]$.*

Proof. Eq.B.2.19, and Theorem B.2.1 applied to each linear family $\mathcal{F}_y$ of (scalar) functions leads to

$$\widehat{\mathcal{R}}_{\mathcal{S}}(\ell_\gamma(\mathcal{F}^A_{M_A}, \cdot)) \leq \frac{c}{\gamma} \sum_{y=1}^{c} \frac{M_A}{n} \sqrt{\mathrm{Tr}\boldsymbol{K}^{yy}_A} \tag{B.2.28}$$

where $\mathrm{Tr}\boldsymbol{K}^{yy}_A := \sum_{i=1}^n (\boldsymbol{K}_A)^{yy}_{ii}$ is computed w.r.t to the indices $i = 1,...,n$ for fixed $y$. Passing the average $\frac{1}{c}\sum_{y=1}^c$ under the root using Jensen inequality, we conclude:

$$\begin{aligned} \widehat{\mathcal{R}}_{\mathcal{S}}(\ell_\gamma(\mathcal{F}^A_{M_A}, \cdot)) &\leq \frac{c^2 M_A}{\gamma n} \sqrt{\frac{1}{c}\sum_{y=1}^{c} \mathrm{Tr}\boldsymbol{K}^{yy}_A} \\ &= \frac{c^{3/2} M_A}{\gamma n} \sqrt{\mathrm{Tr}\boldsymbol{K}_A} \end{aligned} \tag{B.2.29}$$

□

The proof of the extension of these bounds to families learning flows follows the same line as in B.2.3.

## B.2.5. Which Norm for Measuring Capacity?

Implicit biases of gradient descent are relatively well understood in linear models (e.g Gunasekar et al. (2018)). For example when using square loss, it is well-known that gradient descent (initialized in the span of the data) converges to minimum $\ell^2$ norm (resp. RKHS norm) solutions in parameter space (resp. function space). Yet, as pointed out by Belkin et al. (2018); Muthukumar et al. (2020), measuring capacity in terms of such norms is not coherently linked with generalization in practice. Here we discuss this issue by highlighting the critical dependence of meaningful norm-based capacity on the geometry defined by the features. We use the notation of Section 6.4.1: $\boldsymbol{\Phi} = \sum_{j=1}^n \sqrt{\lambda_j}\boldsymbol{u}_j\boldsymbol{v}_j^\top$ denote the $n \times P$ feature matrix and its SVD decomposition.

A standard approach is to measure capacity in terms of the $\ell^2$ norm the weight vector, e.g using bounds (6.4.3) with $A = \mathrm{Id}$. If the distribution of solutions $\mathbf{w}^*_{\mathcal{S}}$, where $\mathcal{S} \sim \rho^n$ is sampled from the input distribution, is reasonably isotropic, taking the smallest $\ell^2$ ball

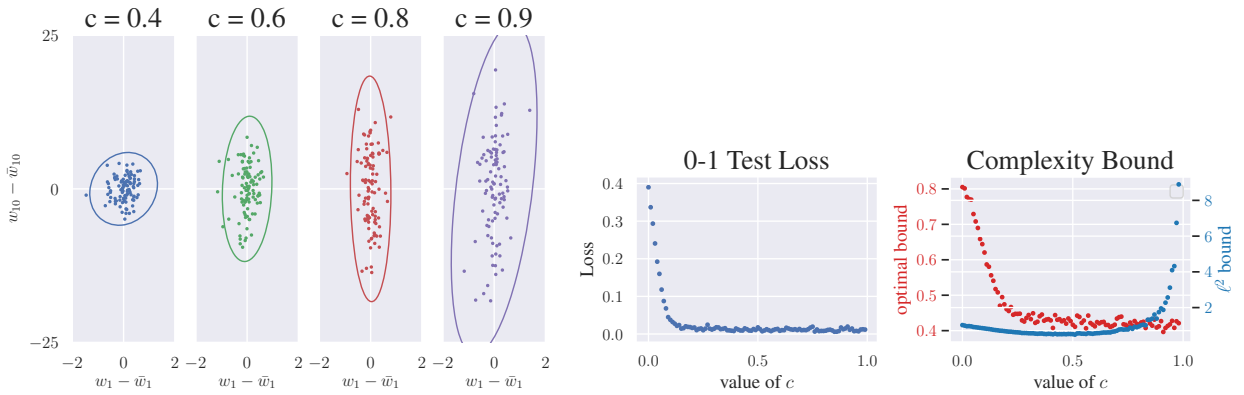



**Figure 3. Left:** 2D projection of the minimum $\ell^2$-norm interpolators $\mathbf{w}^*_{\mathcal{S}}$, $\mathcal{S} \sim \rho^n$, for linear models $f_{\mathbf{w}} = \langle \mathbf{w}, \Phi_c \rangle$, as the feature scaling factor varies from 0 (white features) to 1 (original, anisotropic features). For larger $c$, the solutions scatter in a very anisotropic way. **Right:** Average test classification loss and complexity bounds (B.2.27) with $A = \mathrm{Id}$ (blue plot) for the solution vectors $\mathbf{w}^*_{\mathcal{S}}$, as we increase the scaling factor $c$. As feature anisotropy increases, the bound becomes increasingly loose and fails to reflect the shape of the test error. By contrast, the bound (6.4.3) with $A$ optimized as in Proposition B.2.5 (red plot) does not suffer from this problem.

containing them (with high probability) gives an accurate description of the class of trained models. However for very anisotropic distributions, the solutions do not fill any such ball so describing trained models in terms of $\ell^2$ balls is wasteful (Schölkopf et al., 1999a).

Now, for minimum $\ell^2$ norm interpolators (Hastie et al., 2009),

$$\mathbf{w}^* = \boldsymbol{\Phi}^\top \boldsymbol{K}^{-1} \boldsymbol{y} = \sum_{j=1}^{n} \frac{\boldsymbol{u}_j^\top \boldsymbol{y}}{\sqrt{\lambda_j}} \boldsymbol{v}_j, \tag{B.2.30}$$

where $\boldsymbol{K} = \boldsymbol{\Phi}\boldsymbol{\Phi}^\top$ is the kernel matrix, the solution distribution typically inherits the anisotropy of the features. For example, if $y_i = \bar{y}(\mathbf{x}_i) + \varepsilon_i$ where $\varepsilon_i \sim \mathcal{N}(0, \sigma^2)$, the covariance of the solutions with respect to noise is $\mathrm{cov}_{\boldsymbol{\varepsilon}}[\mathbf{w}^*, \mathbf{w}^*] = \sum_j \frac{\sigma^2}{\lambda_j} \boldsymbol{v}_j \boldsymbol{v}_j^\top$, which scales as $1/\lambda_j$ along $\boldsymbol{v}_j$.

To visualize this on a simple setting, we consider $P$ random features of a RBF kernel[1], fit on 1D data $\mathbf{x}$ modelled by $N$ equally spaced points in $[-a, a]$. In this setting, the (true) feature map is represented by a $N \times P$ matrix with SVD $\Phi = \sum_j \sqrt{l_j} \boldsymbol{\psi}_j \boldsymbol{\varphi}_j^\top$. We assume the (true) labels are defined by the deterministic function $y(\mathbf{x}) = \mathrm{sign}(\boldsymbol{\psi}_1(\mathbf{x}))$. To highlight the effect of feature anisotropy, we further rescale the singular values as $l_j^c = 1 + c(l_j - 1)$ so as to interpolate between whitened features ($c = 0$) and the original ones ($c = 1$). We set $P = N = 1000$. Fig 3 (left) shows 2D projections in the plane $(\boldsymbol{\varphi}_1, \boldsymbol{\varphi}_{10})$ of the (centered) minimum $\ell_2$ norm solutions $\mathbf{w}^*_{\mathcal{S}} - \mathbb{E}_{\mathcal{S}} \mathbf{w}^*_{\mathcal{S}}$, for a pool of 100 training (sub)samples $\mathcal{S}$ of size $n = 50$, for increasing values of the scaling factor $c$. As $c$ approaches 1, the solutions begin

[1]We used RBFsampler of scikit-learn, which implements a variant of Random Kitchen Sinks (Rahimi & Recht, 2007) to approximate the feature map of a RBF kernel with parameter $\gamma = 1$.

to scatter in a very anisotropic way in parameter space; as shown in Fig 3 (right), the complexity bound (B.2.9) based on the $\ell_2$ norm, i.e $A = \mathrm{Id}$ (blue plot), becomes increasingly loose and fails to reflect the shape of the test error.

To find a more meaningful capacity measure, Prop B.2.1 suggests optimizing the bound (6.4.3) with $M_A = \|\mathbf{w}^*\|_A$, over a given class of rescaling matrices $A$. We give an example of this in the following Proposition.

**Proposition B.2.5.** *Consider the class of matrices $A_\nu = \sum_{j=1}^n \sqrt{\nu_j}\boldsymbol{v}_j\boldsymbol{v}_j^\top + \mathrm{Id}_{\mathrm{span}\{\boldsymbol{v}\}^\perp}$, which act as mere rescaling of the singular values of the feature matrix. Any minimizer of the upper bound (B.2.9) for the mininum $\ell^2$-norm interpolator takes the form*

$$\nu_j^* = \kappa \frac{\sqrt{\lambda_j}}{|\boldsymbol{v}_j^\top \mathbf{w}^*|} = \kappa \frac{\lambda_j}{|\boldsymbol{u}_j^\top \boldsymbol{y}|} \tag{B.2.31}$$

*where $\kappa > 0$ is a constant independent of $j$.*

Proof. From (B.2.30) and the definition of $A_\nu$, we first write

$$\|\mathbf{w}^*\|_{A_\nu}^2 = \sum_{j=1}^n \frac{\nu_j}{\lambda_j}(\boldsymbol{u}_j^\top \boldsymbol{y})^2, \quad \mathrm{Tr}\boldsymbol{K}_{A_\nu} = \sum_{j=1}^n \frac{\lambda_j}{\nu_j} \tag{B.2.32}$$

The product of the above two terms has the critical points $\nu_j^*$, $j = 1 \cdots n$ which satisfy

$$\frac{(\boldsymbol{u}_j^\top \boldsymbol{y})^2}{\lambda_j}\mathrm{Tr}\boldsymbol{K}_{A_\nu} - \frac{\lambda_j}{\nu_j^{*2}}\|\mathbf{w}^*\|_{A_\nu}^2 = 0 \tag{B.2.33}$$

giving the desired result $\nu_j^* \propto \lambda_j/|\boldsymbol{u}_j^\top \boldsymbol{y}|$. □

In the context of Proposition B.2.5, we see that the optimal norm $\|\cdot\|_{A_{\nu^*}}$ depends both on the feature geometry – through the singular values – and on the task – through the labels –. As shown in Fig 1 (right, red plot), in the above RBF feature setting, the resulting optimal bound on the Radecher complexity has a much nicer behaviour than the standard bound based on the $\ell^2$ norm.[2]

## B.2.6. SuperNat: Proof of Prop 6.4.2

Prop. 6.4.2 is a *local* version of Prop B.2.5, where the feature rescaling factors are applied at each step of the training algorithm. The procedure is described in Fig 5 (left); the term to be optimized shows up in Step 2. With the chosen class of matrices described in Prop 6.4.2, the action $\boldsymbol{\Phi}_t \to A_\nu^{-1}\boldsymbol{\Phi}_t$ merely rescale its singular values $\lambda_{jt} \to \lambda_{jt}/\nu_j$, leaving its singular vectors $\boldsymbol{u}_j, \boldsymbol{v}_j$ unchanged.

[2]Note however that, since the optimal norm depends on the sample set $\mathcal{S}$, the resulting complexity bound does not directly yield a high probability bound on the generalization error as in (B.2.5). The more thorough analysis, which requires promoting (B.2.5) to uniform bounds over the choice of matrix $A$, is left for future work.

**Proposition B.2.6** (Prop 6.4.2 restated)**.** *For the class of rescaling matrices $A_\nu$ defined in Prop B.2.5, any minimizer in Step 2 in Fig 5, where $\delta \mathbf{w}_{GD} = -\eta \nabla_{\mathbf{w}} L$, takes the form*

$$\nu_{jt}^* = \kappa \frac{1}{|\boldsymbol{u}_j^\top \nabla_{\boldsymbol{f}_\mathbf{w}} L|} \tag{B.2.34}$$

*where $\kappa > 0$ is a constant independent of $j$.*

Proof. Using the chain rule and the SVD of the feature map $\Phi_t$ we write the gradient descent updates at iteration $t$ of SuperNat as

$$\delta \mathbf{w}_{\mathrm{GD}} = -\eta \boldsymbol{\Phi}_t^\top \nabla_{\boldsymbol{f}_\mathbf{w}} L \tag{B.2.35}$$

$$= -\eta \sum_{j=1}^{n} \sqrt{\lambda_{jt}} (\boldsymbol{u}_j^\top \nabla_{\boldsymbol{f}_\mathbf{w}} L)\, \boldsymbol{v}_j, \tag{B.2.36}$$

From the definition of $A_\nu$, we then spell out

$$\|\delta \mathbf{w}_{\mathrm{GD}}\|_{A_\nu}^2 = \eta^2 \sum_{j=1}^{n} (\nu_j \lambda_j)(\boldsymbol{u}_j^\top \nabla_{\boldsymbol{f}_\mathbf{w}} L)^2, \quad \|A_\nu^{-1} \boldsymbol{\Phi}_t\|_F := \mathrm{Tr} \boldsymbol{K}_{tA_\nu} = \sum_{j=1}^{n} \frac{\lambda_j}{\nu_j} \tag{B.2.37}$$

The product of the above two terms has the critical points $\nu_j^*$, $j = 1 \cdots n$ which satisfy

$$\lambda_j (\boldsymbol{u}_j^\top \nabla_{\boldsymbol{f}_\mathbf{w}} L)^2 \mathrm{Tr} \boldsymbol{K}_{A_\nu} - \frac{\lambda_j}{\nu_j^{*2}} \|\delta \mathbf{w}_{\mathrm{GD}}\|_{A_\nu}^2 = 0 \tag{B.2.38}$$

giving the desired result $\nu_j^* \propto 1/|\boldsymbol{u}_j^\top \nabla_{\boldsymbol{f}_\mathbf{w}} L|$. □

# B.3. Additional experiments

## B.3.1. Synthetic Experiment: Fig. 1

To visualize the adaptation of the tangent kernel to the task during training, we perform the following synthetic experiment. We train a 6-layer deep 256-unit wide MLP on $n = 500$ points of the Disc dataset $(\mathbf{x}, y)$ where $\mathbf{x} \sim \mathrm{Unif}[-1,1]^2$ and $y(\mathbf{x}) = \pm 1$ depending on whether is within the disk of center 0 and radius $\sqrt{2/\pi}$, see Fig 4. Fig. 1 in the main text shows visualizations of eigenfunctions sampled using a grid of $N = 2500$ points on the square, and ranked in non-increasing order of the spectrum $\lambda_1 \geq \cdots \geq \lambda_N$. After a number of iterations, we begin to see the class structure (e.g. boundary circle) emerge in the top eigenfunctions. We note also an increasingly fast spectrum decay (e.g $\lambda_{20}/\lambda_1 = 1.5\%$ at iteration 0 and 0.2% at iteration 2000). The interpretation is that the kernel stretches in directions of high correlation with the labels.

## B.3.2. More Alignment Plots

**Varying datasets and architectures:** Fig 5.

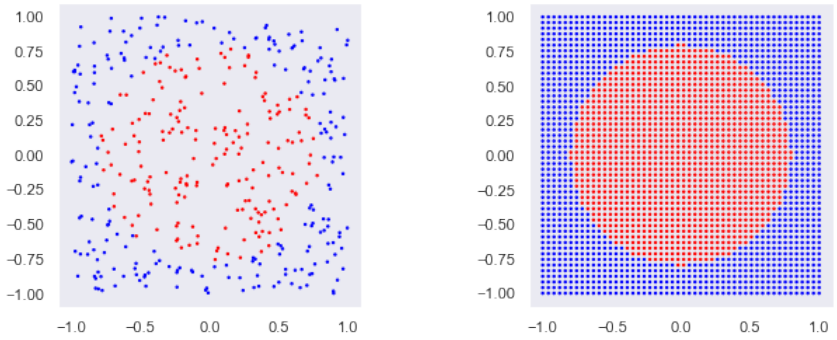


**Figure 4.** Disk dataset. **Left**: Training set of $n = 500$ points $(\mathbf{x}_i, y_i)$ where $\mathbf{x} \sim \text{Unif}[-1,1]^2$, $y_i = 1$ if $\|x_i\|_2 \leq r = \sqrt{2/\pi}$ and $-1$ otherwise. **Right**: Large test sample (2500 points forming a $50 \times 50$ grid) used to evaluate the tangent kernel.

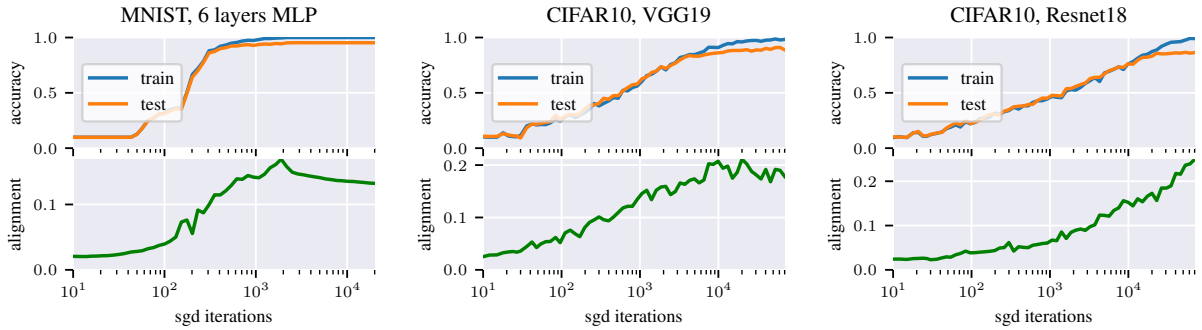



**Figure 5.** Evolution of the CKA between the tangent kernel and the class label kernel $K_Y = YY^T$ measured on a held-out test set for different architectures: **(left)** 6 layers of 80 hidden units MLP on MNIST **(middle)** VGG19 on CIFAR10 **(right)** Resnet18 on CIFAR10. We observe an increase of the alignment to the target function.

**Uncentered kernel Experiments:** Fig 6. The evolution of the alignment to the *uncentered* kernel, in order to assess whether this effect is consistent when removing centering. The experimental details are the same as in the main text; we also observe a similar increase of the alignment as training progresses.

### B.3.3. Effect of depth on alignment

In order to study the influence of the architecture on the alignment effect, we measure the CKA for different networks and different initialization as we increase the depth. The results in Fig 7 suggest that the alignment effect is magnified as depth increases. We also observe that the ratio of the maximum alignment between easy and difficult examples is increased with depth, but stays high for a smaller number of iterations.

### B.3.4. Spectrum Plots with lower learning rate : Fig. 8

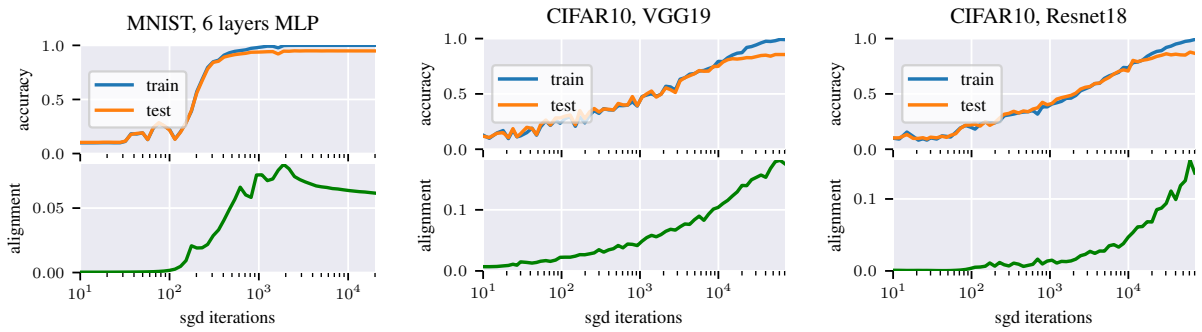



**Figure 6.** Same as figure 5 but without centering the kernel. Evolution of the uncentered kernel alignment between the tangent kernel and the class label kernel $K_Y = YY^T$ measured on a held-out test set for different architectures: **(left)** 6 layers of 80 hidden units MLP on MNIST **(middle)** VGG19 on CIFAR10 **(right)** Resnet18 on CIFAR10. We observe an increase of the alignment to the target function.

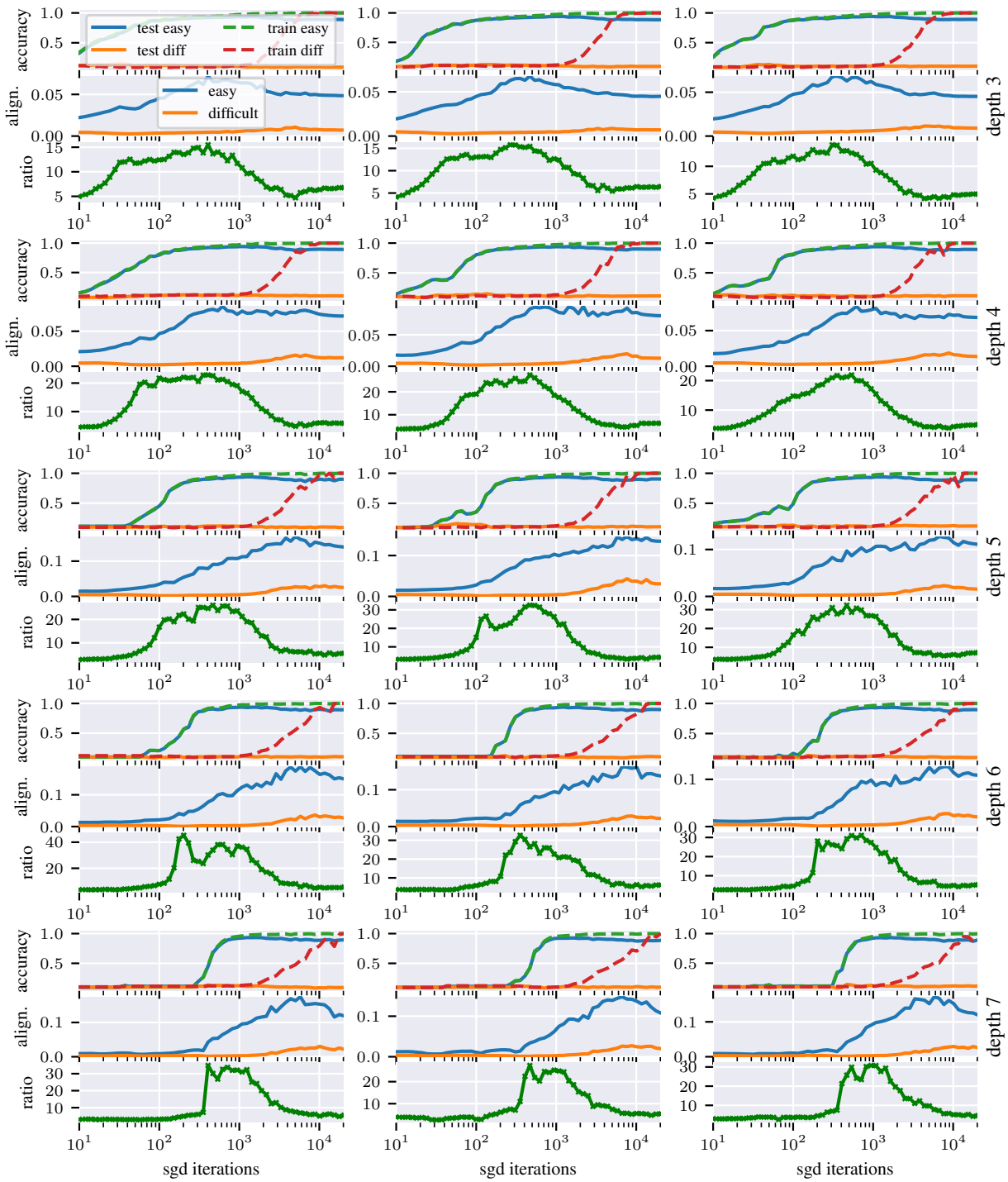



**Figure 7.** Effect of depth on alignment. 10.000 MNIST examples with 1000 random labels MNIST examples trained with learning rate=0.01, momentum=0.9 and batch size=100 for MLP with hidden layers size 60 and **(in rows)** varying depths **(in columns)** varying random initialization/minibatch sampling. As we increase the depth, the alignment starts increasing later in training and increases faster; and the ratio between easy and difficult alignments reaches a higher value.

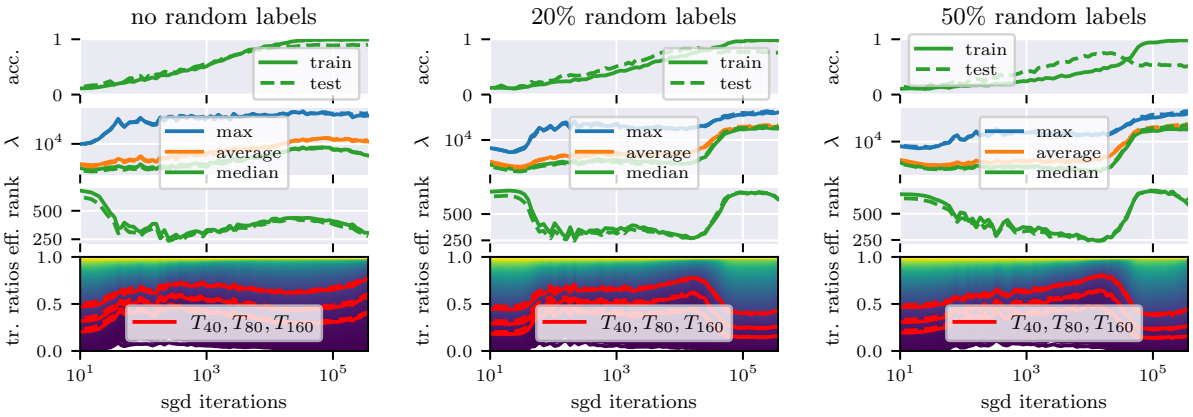



**Figure 8.** Evolution of tangent kernel spectrum, effective rank and trace ratios of a VGG19 trained by SGD with batch size 100, learning rate 0.003 and momentum 0.9 on dataset **(left)** CIFAR10 and **(right)** CIFAR10 with 50% random labels. We highlight the top 40, 80 and 160 trace ratios in **red**.

# Appendix C

---

# Lazy vs hasty: linearization in deep networks impacts learning schedule based on example difficulty Supplementary material

## C.1. Details on the theoretical analysis

We provide some technical details for the statements and proposition of Section 5.4.

### C.1.1. Gradient dynamics

We first derive the equations (5.4.4). Using twice the chain rule yields

$$\dot{\boldsymbol{\theta}} = \sum_{\lambda=1}^{d} \dot{w_\lambda} \nabla_{w_\lambda} \boldsymbol{\theta} = -\sum_{\lambda=1}^{d} \nabla_{w_\lambda} \ell(\boldsymbol{\theta}) \nabla_{w_\lambda} \boldsymbol{\theta} = -\sum_{\lambda=1}^{d} \left( \nabla_{w_\lambda} \boldsymbol{\theta}^\top \nabla_{\boldsymbol{\theta}} \ell(\boldsymbol{\theta}) \right) \nabla_{w_\lambda} \boldsymbol{\theta} \tag{C.1.1}$$

By (5.4.2), we have $\nabla_{w_\lambda} \boldsymbol{\theta} = w_\lambda \boldsymbol{v}_\lambda$; moreover

$$\nabla_{\boldsymbol{\theta}} \ell(\boldsymbol{\theta}) = \boldsymbol{\Sigma} \boldsymbol{\theta} - \boldsymbol{X}^\top \boldsymbol{y} = \boldsymbol{\Sigma}(\boldsymbol{\theta}(t) - \boldsymbol{\theta}^*) \tag{C.1.2}$$

where $\boldsymbol{\Sigma} = \boldsymbol{X}^\top \boldsymbol{X}$ is the input correlation matrix and $\boldsymbol{\theta}^* := \boldsymbol{\Sigma}^+ \boldsymbol{X}^\top \boldsymbol{y} + P_\perp(\boldsymbol{\theta}^0)$ is the solution of the linear dynamics. Substituting into (C.1.1)gives,

$$\dot{\boldsymbol{\theta}}(t) = -\sum_{\lambda=1}^{d} w_\lambda^2(t) \boldsymbol{v}_\lambda \boldsymbol{v}_\lambda^\top \boldsymbol{\Sigma}(\boldsymbol{\theta}(t) - \boldsymbol{\theta}^*) := -\boldsymbol{\Sigma}(t)(\boldsymbol{\theta}(t) - \boldsymbol{\theta}^*) \tag{C.1.3}$$

Since $\boldsymbol{\Sigma}$ is diagonal in the basis $\boldsymbol{v}_\lambda$, $\boldsymbol{\Sigma}(t)$ is obtained from $\boldsymbol{\Sigma}$ by rescaling each of its eigenvalues $\mu_\lambda$ by the time varying factor $w_\lambda^2 = 2\theta_\lambda$.

### C.1.2. Proof of Proposition 5.4.1

The system (C.1.3) is diagonal in the basis $\boldsymbol{v}_\lambda$, so the dynamics decouples for the parameter components $\theta_\lambda$ in this basis. In fact we have,

$$\boldsymbol{\theta} = \sum_{\lambda=1}^{d} \theta_\lambda \boldsymbol{v}_\lambda, \qquad \dot{\theta}_\lambda = -2\mu_\lambda \theta_\lambda (\theta_\lambda - \theta_\lambda^*) \tag{C.1.4}$$

The above equations can be put in the form

$$\frac{\dot{\theta}_\lambda}{\theta_\lambda(\theta_\lambda - \theta_\lambda^*)} = -2\mu_\lambda \tag{C.1.5}$$

We distinguish two cases (note that it follows from our assumptions that $\theta_\lambda^* \geq 0$ for all $\lambda$):

**Case 1**: $\theta_\lambda^* = 0$. In this case, (C.1.5) becomes

$$\frac{d}{dt}\left[-\frac{1}{\theta_\lambda}\right] = -2\mu_\lambda \quad \Rightarrow \quad \frac{1}{\theta_\lambda(t)} = 2\mu_\lambda t + c \tag{C.1.6}$$

where the constant $c$ is fixed by the initial condition $\theta_\lambda(0) = \theta_\lambda^0$ to be $c = 1/\theta_\lambda^0$. Thus,

$$\theta_\lambda(t) = \frac{\theta_\lambda^0}{1 + 2\mu_\lambda \theta_\lambda^0 t} \tag{C.1.7}$$

**Case 2**: $\theta_\lambda^* > 0$. We also focus on the case where $0 < \theta_\lambda^0 < \theta_\lambda^*$; it can be checked (e.g. by a post hoc inspection on the solution) that this implies $0 < \theta_\lambda < \theta_\lambda^*$ for all $t$. In this case, (C.1.5) can be written as

$$\frac{1}{\theta_\lambda^*}\left(\frac{-\dot{\theta}_\lambda}{\theta_\lambda^* - \theta_\lambda} - \frac{\dot{\theta}_\lambda}{\theta_\lambda}\right) = -2\mu_\lambda \tag{C.1.8}$$

and hence

$$\frac{d}{dt}\left[\log(\theta_\lambda^* - \theta_\lambda) - \log(\theta_\lambda)\right] = -2\mu_\lambda \theta_\lambda^* \quad \Rightarrow \quad \frac{\theta_\lambda^* - \theta_\lambda}{\theta_\lambda} = ce^{-2\mu_\lambda \theta_\lambda^* t} \tag{C.1.9}$$

where the constant $c > 0$ is fixed by the initial condition to be $c = \frac{\theta_\lambda^* - \theta_\lambda^0}{\theta_\lambda^0}$. Solving for $\theta_\lambda$ finally gives

$$\theta_\lambda(t) = \frac{\theta_\lambda^0 \theta_\lambda^*}{\theta_\lambda^0 - e^{-2\tilde{y}_\lambda t}(\theta_\lambda^0 - \theta_\lambda^*)} \tag{C.1.10}$$

where we substituted $\tilde{y}_\lambda := \sqrt{\mu_\lambda} y_\lambda = \mu_\lambda \theta_\lambda^*$. The case where $0 < \theta_\lambda^* < \theta_\lambda^0$ is similar and yields the same expression.

## C.2. Code

Code for reproducing the experiments is available at: `https://github.com/tfjgeorge/lazy_vs_hasty`.

In the `linearization_utils.py` file there are 2 new contributed classes extensively used in the project:

- `LinearizationProbe` implements the linearization metrics (end of section 5.2): sign similarity, NTK alignment using NNGeometry (George, 2021) and representation kernel alignment;
- `ModelLinearKnob` implements prediction of a model parameterized by the $\alpha$ scaling (section 5.2).

## C.3. Experimental details

Unless specified otherwise, all model parameters are initialized using the default Pytorch initialization.

### C.3.1. CIFAR10 with C-scores

We train a ResNet18 with SGD with learning rate 0.01, momentum 0.9 and bach size 125. In order to use pre-computed C-scores even for a held-out test set, we split the CIFAR10 dataset into 40 000 train examples and 10 000 test examples.

### C.3.2. CIFAR10 with noisy examples

We train a ResNet18 with SGD with learning rate 0.01, momentum 0.9 and bach size 125.

## C.4. A note on batch norm and $\alpha$ scaling

We chose to refrain from using batch norm in our experiments since it adds unnecessary complexity for analysis, and deep models without batch norm already show intriguing generalization properties that are not fully understood by current theory. Note that most theoretical advances currently focus on models that do not use batch norm.

In addition, the $\alpha$ scaling (section 5.2) is not well defined in the presence of batch norm in training mode. Indeed, in training mode, batch norm works by evaluating the mean and variance using examples of the current mini-batch. When computing $\alpha\left(f_{\boldsymbol{\theta}}\left(\mathbf{x}\right)-f_{\boldsymbol{\theta}_0}\left(\mathbf{x}\right)\right)$ for large $\alpha$, a slight change in parameters is boosted by $\alpha$ to produce a large change in function space. This is expected, and this is balanced by tiny values of the learning rate (shrinked by $\frac{1}{\alpha^2}$). But small changes in mean and variance are also boosted, and this is probably not expected. For this reason, we chose to limit the scope of the current paper to models without batch norm.

As an exception, in the Waterbirds experiment of section 5.3.3, we used a pre-trained ResNet18 network, that includes batch norm layers. We chose to fine-tune in evaluation mode, where the mean and variance are not computed using examples of the mini-batch,

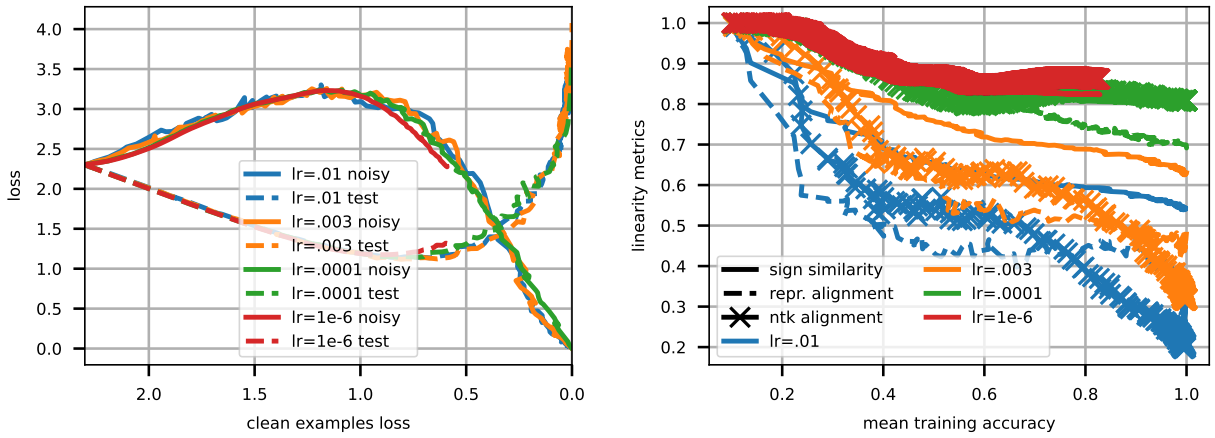



**Figure 1.** same as fig. 2, with $\alpha = 1$ and varying learning rates in $\{0.01, 0.003, 10^{-4}, 10^{-6}\}$. In this experiment, we rule out the role of the learning rate in learning speed of easy/difficult examples, since regardless of the learning rate, all runs follow the same trajectory as measured by noisy examples accuracy during training, and test examples accuracy during training. This shows that modulating the learning rate plays a different role as the $\alpha$ scaling.

but are instead constant, kept at their pre-trained values. This way, all other parameters (including batch norm's $\gamma$ and $\beta$) are "standard" parameters that can be linearized.

## C.5. Interplay between learning rate and $\alpha$ scaling

It has been observed in (Li et al., 2019) that the initial learning rate used during training plays an important role in the final outcome, in that models trained with large initial learning rate usually get better generalization on a test set. In order to rule out the role of the learning rate in our easy/difficult examples setup, we compared models trained with $\alpha = 1$ and varying learning rate in a broad range (4 orders of magnitude).

Fig. 1 shows our result on the noisy vs clean examples experiment (similar to fig. 2 in the main body). Our plots are also normalized by clean example loss, even if runs with small learning rate require many more training iterations to attain a similar progress (measured in e.g. training loss). We observe that even if using a smaller learning rate keeps the training dynamics in a more linearized regime (on the right), it is still less linearized than our runs scaled by $\alpha$: compare the $\text{lr} = 10^{-6}$ to the $\alpha = 100$ run e.g., since both use the same learning rate once rescaled by $\alpha$, but the linearity metrics stay close to 1 in the $\alpha$ scaled run, whereas even with this tiny learning rate, the $\text{lr} = 10^{-6}$ run in 1 is in a slightly non-linear regime as the linearity metrics drop to 0.9. When looking at fig. 1 left, we also see that both the training accuracy on noisy examples, and the test accuracy seem to follow the same trajectory regardless of the learning rate, at least in the first part of training, whereas in fig. 2, modulating the value of $\alpha$ causes the trajectories to diverge.

## C.6. Additional experiments

In order to verify that our findings also hold true on other architectures, we reproduce the CIFAR10 experiments that use a ResNet18 network, with a VGG11 network. Fig. 4 presents results on the C-scores experiment, and figure 3 presents results on the noisy example experiment.

In fig. 2, fig. 7 and fig. 8, we vary the network initial parameters and SGD minibatch in order to check for variability, and in order to verify that our results are not cherry-picked.

In fig. 6, we also include the accuracy plots corresponding to the experiments in sections 5.3.2.2 and 5.3.2.1 and presented in fig. 2. In fig. 5, we plot the difference between non-linear and linear loss for the same experiments.

## C.7. Numerical simulations of the analytical setup

An interesting question is whether our simplified parameterization of section 5.4 reproduces the same qualitative behaviour than vanilla 2-layer networks for the studied settings. In the experiments in fig. 9, we train a 2-layer MLP on the datasets of examples 1, 2 and 3. We do not impose any constraint on the parameters, which are dense weight matrices. We observe similar trends as in fig. 4.

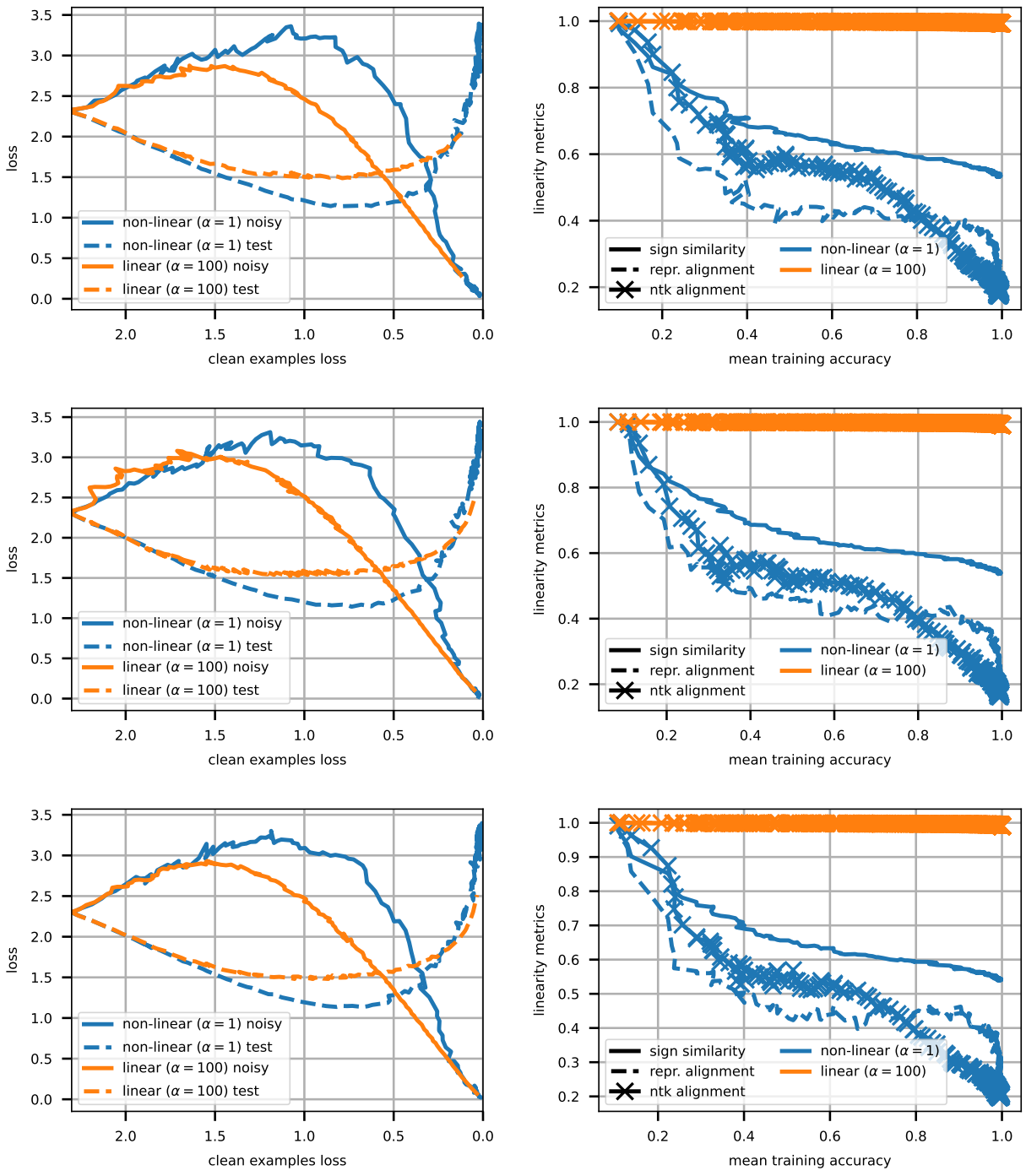



**Figure 2.** same as fig. 2, varying seed (model initialization and mini-batch order)

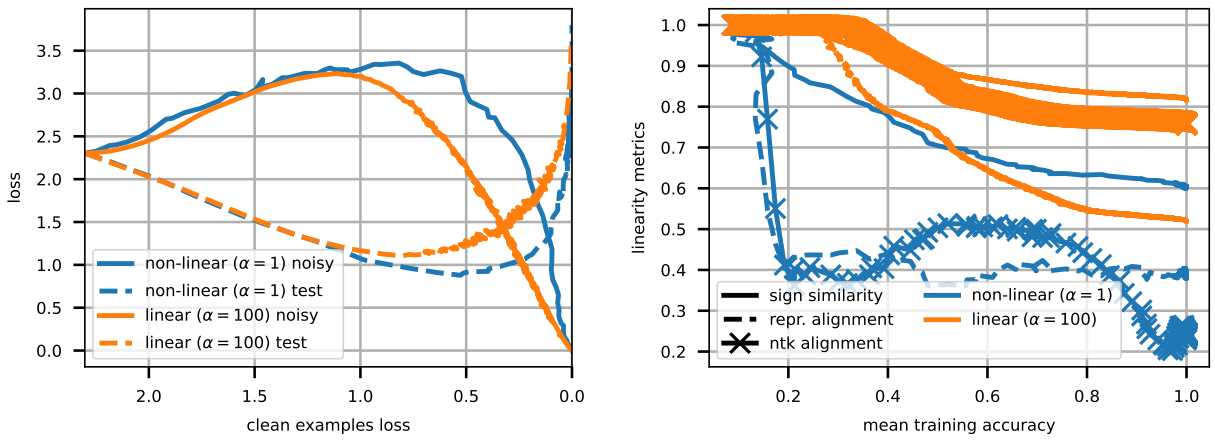



**Figure 3.** same as fig. 2, with a VGG11 network instead

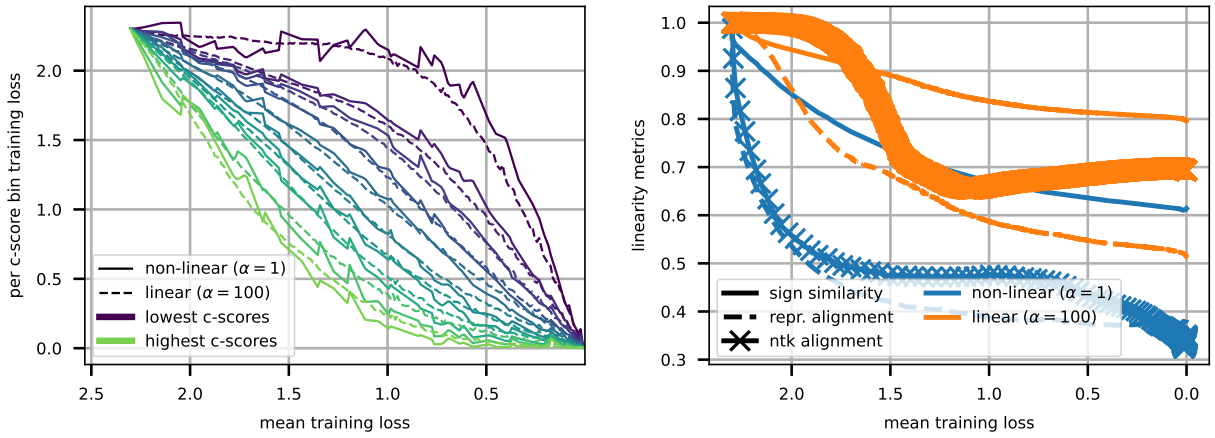



**Figure 4.** same as fig. 2, with a VGG11 network instead

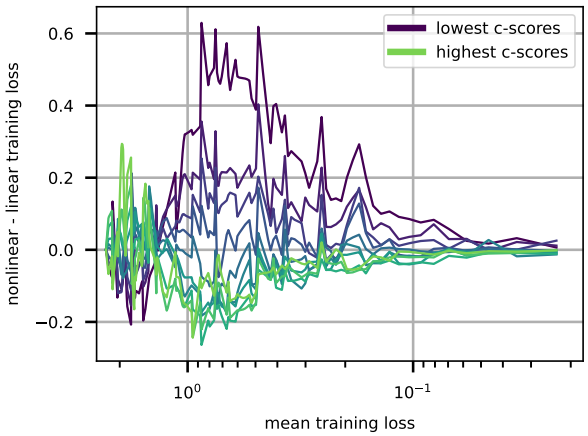



**Figure 5.** same as fig. 2, but we instead plot the differences between non-linear and linear regimes of the loss computed on groups of examples ranked by C-Score. Similarly to figure 2, we observe that at equal training loss (on the x-axis), the loss on high C-score examples is lower for the non-linear regime, and conversely for low C-score examples.

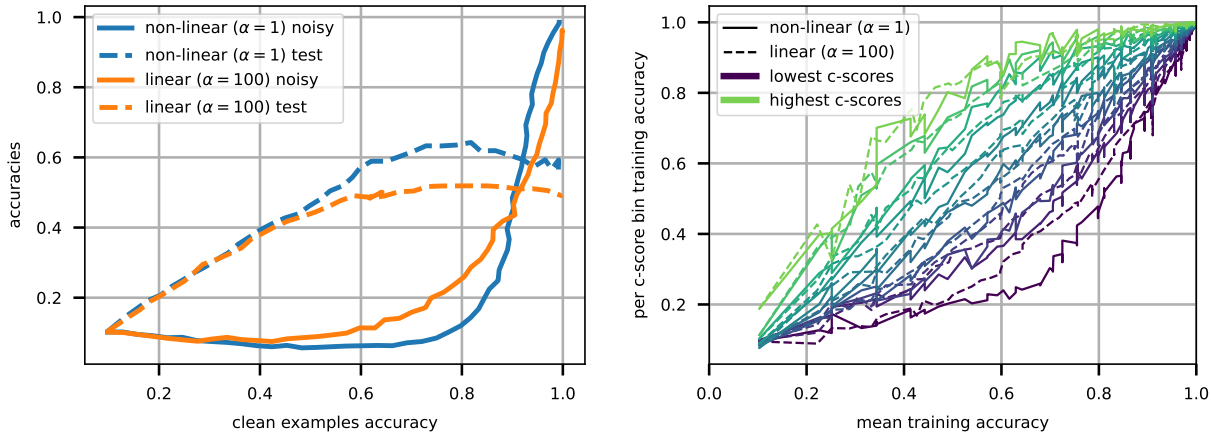



**Figure 6.** Accuracy plots for the experiments of fig. 2. We observe similar trends by looking at the accuracy: **(left)** the non-linear run starts memorizing noisy (difficult) examples later in training than the linear run (solid curve), while simultaneously the test accuracy reaches a higher value for the non-linear run. **(right)** Lower C-score examples are learned comparatively faster in the linear regime, whereas learning curves are more spread in the non-linear regime which indicates a comparatively higher hierarchy between learning groups of examples ranked by their C-score.

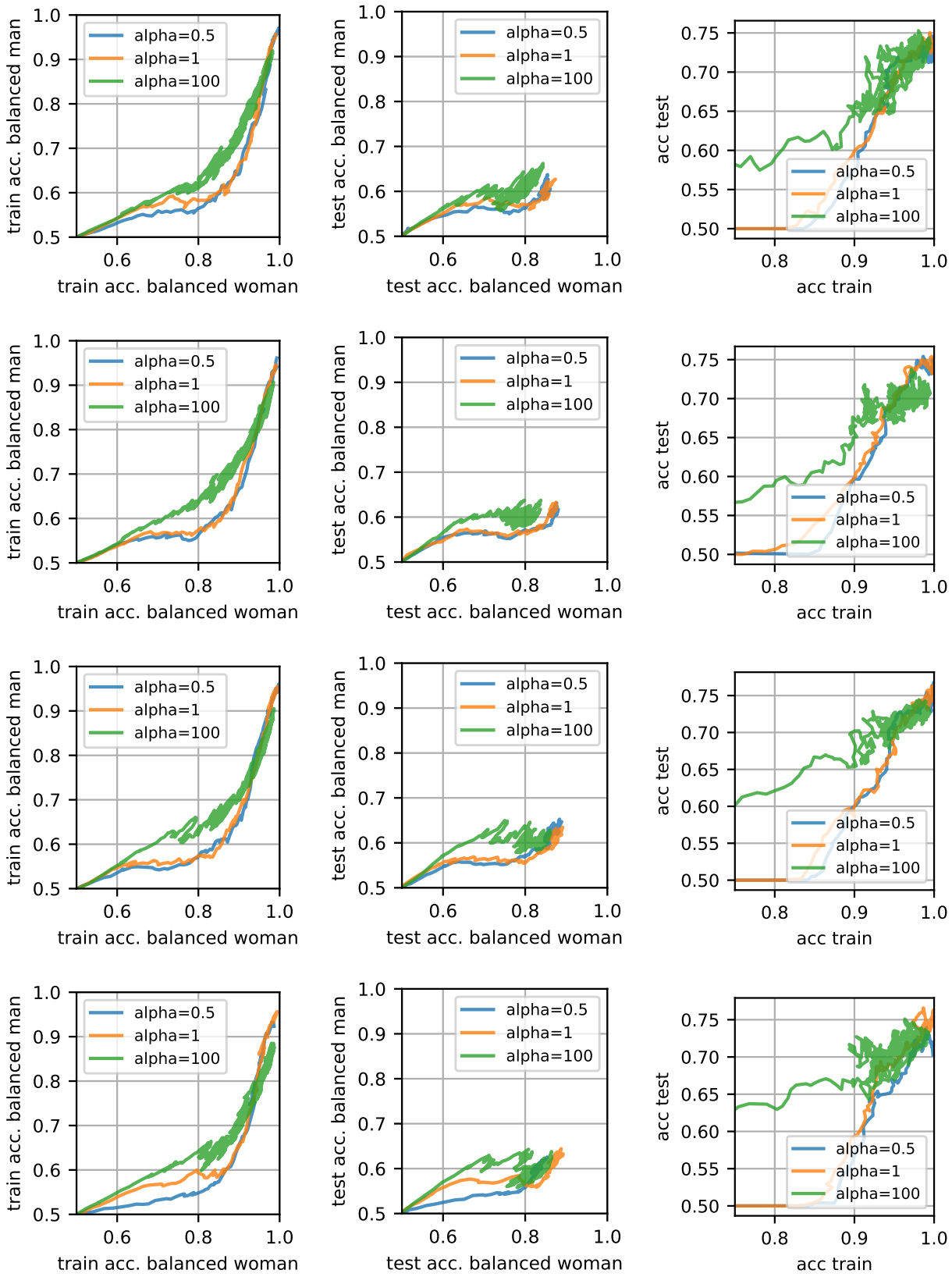



**Figure 7.** same as fig 3 with varying seed (model initialization and minibatch order)

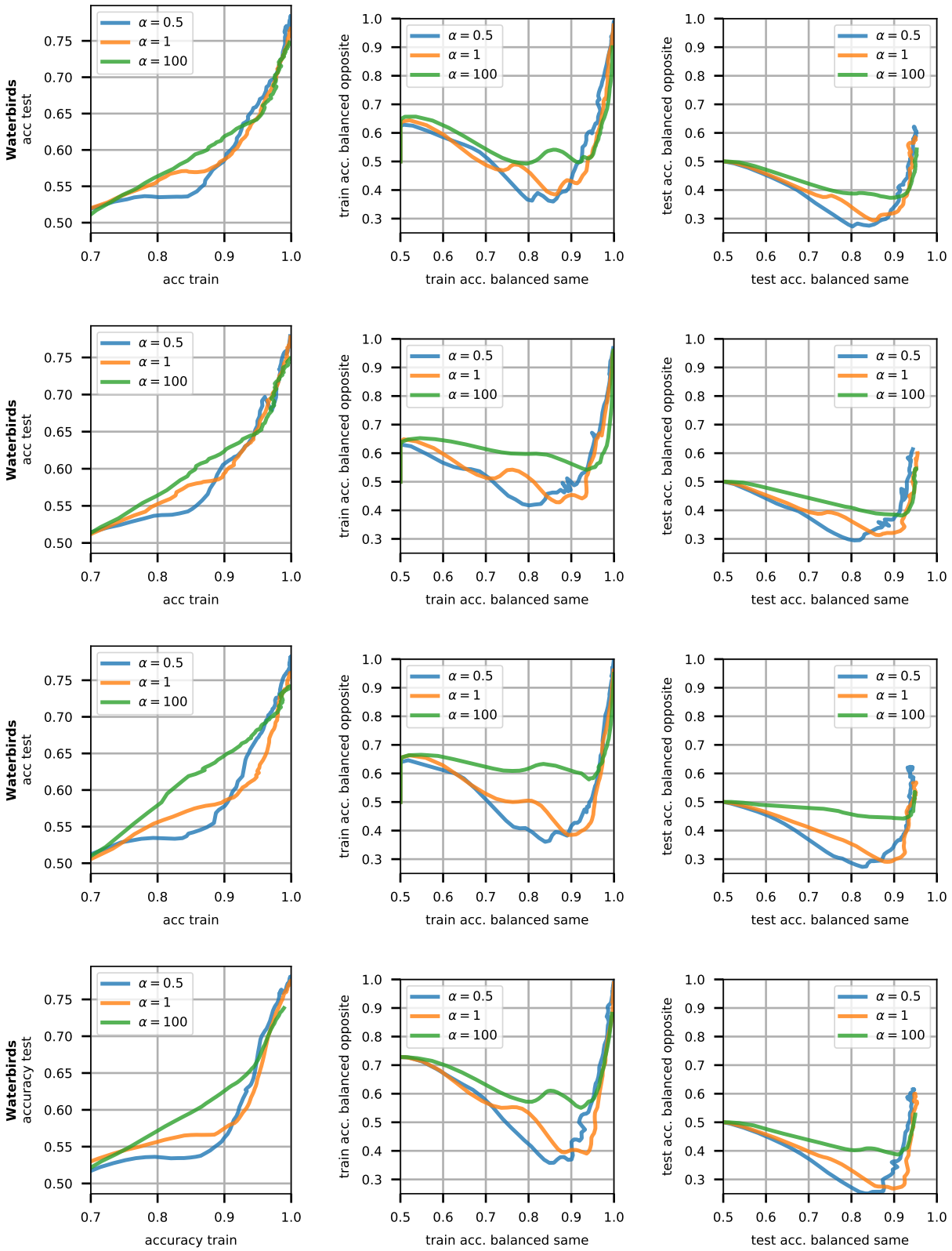



**Figure 8.** same as fig 3 with varying seed (model initialization and minibatch order)

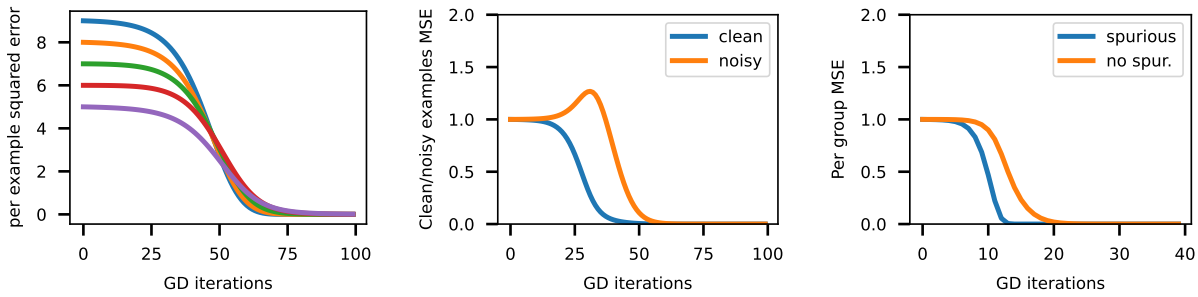



**Figure 9. (left)** On the same datasets as in fig. 4, we train a 2-layer MLP with no additional constraint on the parameterization. Per-group training curves look very similar to the ones obtained in fig. 4, and in particular the learning rank between easy and difficult groups of examples is magnified compared to the linear model of section 5.4. This validates that our analytical model captures the qualitative behaviour highlighted in this work, even though we use a simplified parameterization so as to get closed-form expressions for the training curves.