POLITECNICO DI TORINO
SORBONNE UNIVERSITÉ

Master of Science in Physics of Complex Systems

Spectral Analysis of Infinitely Wide Convolutional Neural Networks

Author:
Alessandro Favero

Supervisors:
Prof. Alfredo Braunstein, Politecnico di Torino
Prof. Matthieu Wyart, EPFL

October 2020

Abstract

Recent works have shown the equivalence between training infinitely wide fully connected neural networks (FCNs) by gradient descent and kernel regression with the neural tangent kernel (NTK). This kernel can also be extended to convolutional neural networks (CNNs), modern architectures that achieve remarkable performance in image recognition, and other translational-invariant pattern detection tasks. The resulting convolutional NTKs have been shown to perform strongly in classification experiments. Still, we lack a quantitative understanding of the generalization capabilities of these models. In this thesis, we introduce a minimal convolutional architecture, and we compute the associated NTK. Following recent works on the statistical mechanics of generalization in kernel methods, we study this kernel's performance in a teacher-student setting, comparing it with the NTK of a two-layer FCN when learning translational-invariant data. Finally, we test our predictions with numerical experiments both on synthetic and real data. Our results show that these kernels cannot compress invariant dimensions and escape the curse of dimensionality. However, the convolutional kernel's eigenfunctions are better aligned with translational-invariant data, effectively lowering the generalization error by a dimensional-dependent prefactor.

Acknowledgements

First, I want to thank my supervisor, Matthieu, for believing in me and giving me the great opportunity to do this thesis with his group. His insights, questions, and sharp reasoning have been invaluable.

A special thank goes to Stefano for his availability and the many discussions that have been fundamental for the results in this thesis.

I am also grateful to the other members of the PCSL group at EPFL, particularly to Leonardo and Mario for their tips.

Many thanks go to my friends and the companions who shared with me this master's unforgettable adventure across Trieste, Torino, and Paris.

Finally, I am deeply indebted to my parents and my family for their constant support and caring.

Contents

Chapter 1

Introduction

Deep learning is revolutionizing the world around us. Self-driving cars and natural language processing are examples of the outstanding achievements of deep neural networks. Despite their success, a theoretical understanding of these complex models still eludes us, and many fundamental questions related to how they work remain open. Answering them could provide significant benefits in terms of performance and reliability.

Deep neural networks are parametrized models that can represent very complex non-linear functions. Learning corresponds to improving the parameters' values to fit a set of training data. The best parameters are found by descending a very high-dimensional loss function. This dynamics can be studied making an analogy with glasses, which are complex physical systems with a non-convex energy landscape and exponentially many local minima. Recent studies show that in the overparametrized regime, when the number of parameters is bigger than the number of data points, the landscape is not glassy, but instead it is characterized by connected level sets and many flat directions, allowing for convergence to a global optimum [1, 2, 3, 4, 5]. Remarkably, in this regime the networks do not overfit the data as one can expect from classical statistics, and the generalization performance keeps increasing with the number of parameters [6, 7, 8, 9]. Indeed, practitioners often train these models using billions of parameters. These observations underline the importance of studying neural networks in the limit in which they have an infinite number of parameters. Depending on the initialization of the parameters, two distinct regimes emerge. In the first, the learning dynamics simplifies, and the output of the network becomes a linear function of the parameters around initialization, i.e. the network learns with very small changes of the parameters. Formally, in this regime the evolution in time of the network's function can be described in terms of a fixed kernel matrix, called neural tangent kernel (NTK) [10, 11, 12]. In the second, the dynamics is richer, and the network's parameters behave as interacting particles in a time-varying velocity field determined by the optimization algorithm [13, 14, 15, 16]. In this regime the pa-

rameters evolve significantly, and the neural network's components learn to respond to semantically-meaningful features (from simple edges to human faces). It has been argued that, because of its "lazy" dynamics, the NTK regime is unlikely to explain the success of deep networks [17]. Nevertheless, experiments show that networks in this regime can still achieve remarkable performances, beating all traditional kernel methods [18]. This is the case of convolutional neural networks (CNNs), that are architectures inspired by the human visual cortex that can take advantage of the fact that many datasets posses a lot of structure and symmetries. More precisely, these models implement invariance to translations (using convolutional filters) and locality (limiting the filters' support). CNNs achieve state-of-the-art results in many fields and are among the most successful deep architectures. However, a quantitative understanding of their impressive performance is still missing. How many data points do they need to learn a given task? What does determine the gap with other architectures (e.g. fully connected networks) that are even more expressive than CNNs? How strong are the priors of locality and translational invariance?
Motivated by the remarkable performance of CNNs' kernels, this thesis aims to start investigating these questions in the NTK regime. Specifically, we want to investigate if and how much the prior of translational invariance affects the sample complexity (how many training data are needed to learn a task) in this regime.

Thesis structure. In chapter 2, we provide the necessary background in deep learning theory. In chapter 3, we introduce an analytically-tractable model of a simple CNN, and we study its performance in the NTK limit. In chapter 4, we present numerical experiments to check our predictions and to extend them to more complex data. Finally, in chapter 5 we discuss our findings and possible future directions.

Chapter 2

Background

This chapter introduces the necessary background in deep learning. For further details, we refer the reader to [19, 20, 21]. Other references to research articles are given inside the sections.

2.1 Supervised Learning

Supervised learning is essentially a high-dimensional interpolation problem. We are given a training set $\{(\boldsymbol{x}_i, f^\star(\boldsymbol{x}_i))\}_{i=1}^p$, with $\boldsymbol{x}_i \in \mathcal{X} \subseteq \mathbb{R}^d$ (input space), $f^\star(\boldsymbol{x}_i) \in \mathcal{Y}$ (label space), and $(\boldsymbol{x}_i, f^\star(\boldsymbol{x}_i)) \sim \mu$ which is a probability measure over $\mathcal{X} \times \mathcal{Y}$. For example $\boldsymbol{x}_i$ can be a picture of an animal with d pixels, and $f^\star(\boldsymbol{x}_i)$ can indicate if it is present a dog or a cat. Our goal is to find a function $f : \mathcal{X} \to \mathcal{Y}$ that predicts correctly the label of a new instance $\boldsymbol{x}$. We introduce a loss function $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_+$ which measures the cost $\ell(f(\boldsymbol{x}), f^\star(\boldsymbol{x}))$ of predicting the label $f(\boldsymbol{x})$ when the true one is $f^\star(\boldsymbol{x})$. Ideally, we want to choose f minimizing the expected risk (or generalization error) $\mathcal{R}(f) = \mathbb{E}_\mu[\ell(f(\boldsymbol{x}), f^\star(\boldsymbol{x}))]$, but practically we don't have access to μ. Therefore, we minimize the empirical risk (or training error) $\hat{\mathcal{R}}(f)$, which is the expectation of the loss with respect to the empirical distribution $\hat{\mu} = \frac{1}{p}\sum_{i=1}^p \delta_{\boldsymbol{x}, \boldsymbol{x}_i}$

$$\min_{f \in \mathcal{F}} \hat{\mathcal{R}}(f) = \min_{f \in \mathcal{F}} \left(\frac{1}{p} \sum_{i=1}^{p} \ell(f(\boldsymbol{x}_i), f^\star(\boldsymbol{x}_i)) \right) \tag{2.1}$$

where $\mathcal{F} \subseteq \{\mathcal{X} \to \mathcal{Y}\}$ is a family of functions called the hypothesis class. The choices of $\mathcal{F}$ and of the optimization algorithm lie at the heart of machine learning. Once the empirical minimization problem is solved, we can estimate the generalization error by computing the risk of the optimal f over a test set of labelled points different from the ones of the training set. An important performance indicator is the learning curve, which describes how the generalization error decays with the number of training points p. This curve is strongly affected by the regularity assumptions on the target

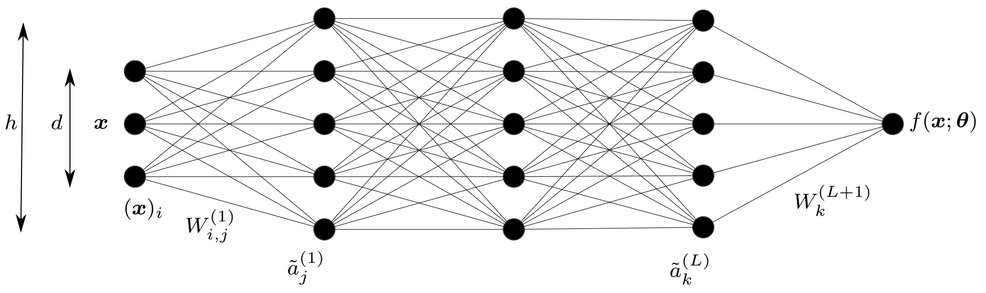


Figure 2.1: Fully connected neural network with L layers of width h as defined in equations (2.2), (2.3), (2.4). Nodes represents neurons, edges are characterized by a weight. Biases are not represented.

$f^\star$ and therefore by the choice of the family $\mathcal{F}$. If we assume $f^\star$ to be just Lipschitz continuous, the generalization error cannot be guaranteed to decay faster than $p^{-\beta}$, with $\beta = O(1/d)$ [22]. This is due to the fact that we need $p \sim \varepsilon^{-d}$ points to ε-cover the data (assuming compactness). Thus, learning is practically impossible in high-dimensional spaces without stronger regularity priors, and one needs to restrict the hypothesis class to beat this curse of dimensionality, for instance leveraging the symmetries and the invariants of the task to learn.

2.2 Neural Networks

Architectures

A fully connected neural network (FCN) with L layers of width h corresponds to the graph shown in figure 2.1. Nodes in the graph represent neurons, and we associate a weight $W^{(t)}_{i,j}$ with each edge connecting them. Neurons take the weighted sum of the outputs of all the neurons in the previous layer, add a bias $b^{(t)}_i$ to obtain the so-called pre-activations $\tilde{a}^{(t)}_i$, and apply a non-linear function $\sigma : \mathbb{R} \to \mathbb{R}$. Commonly used non-linearities are the rectified linear unit (ReLU) $\sigma(a) = \max(0, a)$ or the hyperbolic tangent $\sigma(a) = \tanh(a)$. These computations are done iteratively from layer to layer, and the output function of the network $f(\boldsymbol{x};\boldsymbol{\theta})$ in response to an input $\boldsymbol{x} \in \mathbb{R}^d$ can be written recursively as

$$f(\boldsymbol{x};\boldsymbol{\theta}) = \tilde{a}^{(L+1)} \tag{2.2}$$

$$\tilde{a}^{(t)}_j = \sum_i W^{(t)}_{i,j} \sigma\left(\tilde{a}^{(t-1)}_i\right) + b^{(t)}_j \tag{2.3}$$

$$\tilde{a}^{(1)}_j = \sum_i W^{(1)}_{i,j} (\boldsymbol{x})_i + b^{(1)}_j \tag{2.4}$$

where $\boldsymbol{\theta}$ denotes the parameters $\{W_{i,j}^{(t)} \in \mathbb{R}\}_{i,j,t}$ and $\{b_i^{(t)} \in \mathbb{R}\}_{i,t}$ collectively.
The universal approximation theorem states that a neural network with a single hidden layer ($L = 2$) can approximate any continuous function on a compact domain with arbitrary precision given enough neurons [23, 24]. However, this result does not specify the number of neurons needed to approximate a given function with a shallow network, and deeper networks ($L > 2$) can be more efficient in terms of the network's size. For instance, the number of hidden neurons needed to approximate a radial function $f^\star(\|\boldsymbol{x}\|^2)$ is exponential in the dimension d of the input $\boldsymbol{x}$ using a single hidden layer, but polynomial in d using two hidden layers [25].
As we commented at the end of the previous section, when learning in high-dimensions it is crucial to use the prior knowledge on the data. Convolutional neural networks (CNNs) are an important variant of FCNs that exploit locality and translational invariance. Therefore, these architectures are particularly suited for image analysis, for which they achieve state-of-the-art results. A CNN alternates convolutional layers and subsampling (pooling) layers, followed by some dense (fully connected) layers. A convolutional layer at depth t is composed by $h^{(t)}$ channels, and each channel c is obtained convolving a filter $W^{(t,c,c')}$ of support Ω with the outputs of the channels c' of the previous layer at depth $t-1$ (which are called features). Mathematically,

$$\tilde{a}_{i,j}^{(t,c)} = \sum_{c'=1}^{h^{(t-1)}} \sum_{(i'-i,j'-j)\in\Omega} W_{i'-i,j'-j}^{(t,c,c')} \sigma\left(\tilde{a}_{i',j'}^{(t-1,c')}\right) + b^{(t,c)}. \tag{2.5}$$

A padding scheme determines how borders are treated. Usual choices are adding pixels and setting them to zero, or imposing periodic boundary conditions. The stride controls if the filters are convolved with the input at all the possible locations or skip a number of pixels at each movement. Pooling layers reduce the resolution performing a spatial coarse-graining. The most common procedures are maximum pooling, where a subregion of outputs at neighboring locations j is substituted with the maximum element, and average pooling, which corresponds to substituting with the average value. The output features are (locally) translational invariant depending on how much spatial resolution is lost by stride and pooling. Convolutions (which correspond to sparse matrices with shared parameters) and pooling reduce the computational complexity of CNNs significantly compared to FCNs. Furthermore, deep layers are able to combine the features of the previous layers and learn to activate for very complex and often semantically meaningful patterns. Figure 2.2 represents LeNet-5, a simple convolutional architecture introduced by LeCun et al. for handwritten character recognition [26].

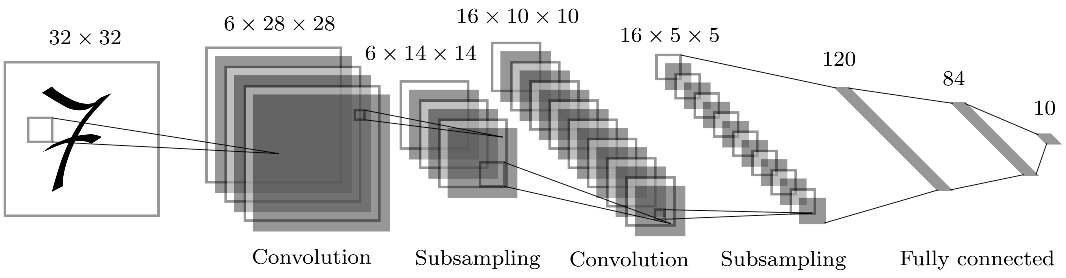


Figure 2.2: Convolutional neural network (LeNet-5) composed by two convolutional layers with filters of size 5×5, two subsampling layers performing 2×2 average pooling, and three fully connected layers.

Optimization Dynamics

Neural networks are trained via empirical risk minimization, i.e. minimizing a loss function over a training set. However, the loss is not convex with respect to the parameters $\boldsymbol{\theta}$, and in general minimizing a non-convex function is computationally intractable. Indeed, this problem is $\mathcal{NP}$-hard. In practice, this minimization is successfully done using stochastic gradient descent (SGD), without any theoretical guarantee on the convergence to a global minimizer. SGD is a variant of gradient descent that starts from a random initialization of the parameters and updates them moving in the negative gradient's direction iteratively. In contrast with gradient descent, the gradient at each step is computed only using a small subset of samples extracted randomly from the training set, called a mini-batch $\mathcal{B}_t$. Given the loss function $\ell(\cdot,\cdot)$, the update rule reads

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta_t \frac{1}{|\mathcal{B}_t|} \sum_{\boldsymbol{x}_i \in \mathcal{B}_t} \nabla_{\boldsymbol{\theta}} \ell(f(\boldsymbol{x}_i; \boldsymbol{\theta}_t), f^\star(\boldsymbol{x}_i)) \tag{2.6}$$

where η_t is the learning rate, and the gradient is efficiently computed using a procedure called back-propagation, which is based on the chain rule for derivatives. Compared to gradient descent, SGD is noisier due to its stochastic nature, but it is also much more efficient in terms of computational time and memory.

2.3 Kernel Methods and Wide Neural Networks

Reproducing Kernel Hilbert Spaces

We define a positive definite symmetric (PDS) kernel $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ as a function of two variables which satisfies for any $c \in L_2(\mathcal{X})$

$$\iint_{\mathcal{X}\times\mathcal{X}} d\boldsymbol{x}\, d\boldsymbol{x}'\, K(\boldsymbol{x}, \boldsymbol{x}') c(\boldsymbol{x}) c(\boldsymbol{x}') \geq 0. \tag{2.7}$$

The Reproducing Kernel Hilbert Space (RKHS) $\mathcal{H}_K$ associated to the PDS kernel $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is the completion of the space of functions of the form $f(\boldsymbol{x}) = \sum_{i=1}^{n} \alpha_i K(\boldsymbol{x}_i, \boldsymbol{x})$ with the inner product $\langle \cdot, \cdot \rangle_{\mathcal{H}_K}$. Given two functions $f(\boldsymbol{x}) = \sum_i \alpha_i K(\boldsymbol{x}_i, \boldsymbol{x})$, $g(\boldsymbol{x}) = \sum_j \beta_j K(\boldsymbol{x}_j, \boldsymbol{x})$, their inner product in $\mathcal{H}_K$ is defined as

$$\langle f, g \rangle_{\mathcal{H}_K} = \sum_{i,j} \alpha_i \beta_j K(\boldsymbol{x}_i, \boldsymbol{x}_j). \tag{2.8}$$

This inner product induces the norm

$$\begin{aligned} \|f\|_{\mathcal{H}_K} &= \sqrt{\langle f, f \rangle_{\mathcal{H}_K}} && (2.9) \\ &= \sqrt{\sum_{i,j} \alpha_i \alpha_j K(\boldsymbol{x}_i, \boldsymbol{x}_j)} && (2.10) \\ &= \sqrt{\boldsymbol{\alpha}^\top \mathbb{K} \boldsymbol{\alpha}} && (2.11) \end{aligned}$$

where $(\mathbb{K})_{ij} = K(\boldsymbol{x}_i, \boldsymbol{x}_j)$ is called the Gram matrix. The kernel K is called reproducing for $\mathcal{H}_K$ since $\langle K(\cdot, \boldsymbol{x}), f \rangle_{\mathcal{H}_K} = f(\boldsymbol{x})$, for every $f \in \mathcal{H}_K$ and $\boldsymbol{x} \in \mathcal{X}$. If $\mathcal{X}$ is compact, K admits the Mercer's decomposition [27]

$$K(\boldsymbol{x}, \boldsymbol{x}') = \sum_{\rho} \lambda_\rho \phi_\rho(\boldsymbol{x}) \phi_\rho(\boldsymbol{x}') \tag{2.12}$$

with eigenvalues λ_ρ and eigenfunctions ϕ_ρ defined by

$$\int dx' K(\boldsymbol{x}, \boldsymbol{x}') \phi_\rho(\boldsymbol{x}') = \lambda_\rho \phi_\rho(\boldsymbol{x}). \tag{2.13}$$

Therefore, we can also define a RKHS as the space of functions of the form $f(\boldsymbol{x}) = \sum_\rho a_\rho \phi_\rho(\boldsymbol{x})$ with finite norm $\|f\|_{\mathcal{H}_K} = \sum_\rho \frac{a_\rho^2}{\lambda_\rho}$. If the eigenvalues decay asymptotically fast, in order to have a finite norm, also the coefficients a_ρ must decay fast. Thus, the smoothness of the functions in a RKHS is controlled by kernel eigenvalues' decay.
Learning in a RKHS consists in mapping the data from the input space into a higher-dimensional feature space via the map

$$\Psi(\boldsymbol{x}) = \left(\sqrt{\lambda_1} \phi_1(\boldsymbol{x}), \sqrt{\lambda_2} \phi_2(\boldsymbol{x}), \dots \right) \tag{2.14}$$

for which $K(\boldsymbol{x}, \boldsymbol{x}') = \langle \Psi(\boldsymbol{x}), \Psi(\boldsymbol{x}') \rangle$. Then, for any algorithm where the inputs appear only in inner products, we can map the inputs into the feature space by replacing their inner products with the kernel $K(\boldsymbol{x}, \boldsymbol{x}')$. With this method, called the kernel trick, we obtain a linear model in the feature space, which is equivalent to a non-linear model in the original space, without significantly increasing the computational complexity. Let consider a learning problem using the RKHS $\mathcal{H}_K$ as the hypothesis class $\mathcal{F}$. We want to solve an empirical minimization problem of the form

$$\min_{f\in\mathcal{H}_K}\left(\hat{\mathcal{R}}(f)+\lambda\|f\|^2_{\mathcal{H}_K}\right) \tag{2.15}$$

where we introduced the regularization term $\lambda\|f\|^2_{\mathcal{H}_K}$ ($\lambda > 0$) in order to control the complexity of the function f and to avoid overfitting. The representer theorem states that the optimal solution to this problem can always be written as

$$f(\boldsymbol{x})=\sum_{i=1}^{p}\alpha_i K(\boldsymbol{x}_i,\boldsymbol{x}) \tag{2.16}$$

where $\{\boldsymbol{x}_i\}_{i=1}^p$ are the p points of the training set, and K is the reproducing kernel associated to $\mathcal{H}_K$. As an example, we consider regularized least square kernel regression

$$\begin{aligned}\frac{1}{p}\sum_{i=1}^{p}(f(\boldsymbol{x}_i)-f^\star(\boldsymbol{x}_i))^2+\lambda\|f\|^2_{\mathcal{H}_K} &= \frac{1}{p}\sum_{i=1}^{p}\left(\sum_{i=1}^{p}\alpha_i K(\boldsymbol{x}_i,\boldsymbol{x})-f^\star(\boldsymbol{x}_i)\right)^2 && (2.17)\\ &+\lambda\sum_{i,j}\alpha_i\alpha_j K(\boldsymbol{x}_i,\boldsymbol{x}_j) && (2.18)\\ &=\boldsymbol{\alpha}^\top\mathbb{K}^2\boldsymbol{\alpha}-2\boldsymbol{y}^\top\mathbb{K}\boldsymbol{\alpha}+\boldsymbol{y}^\top\boldsymbol{y}+\lambda\boldsymbol{\alpha}^\top\mathbb{K}\boldsymbol{\alpha} && (2.19)\end{aligned}$$

where we defined $(\boldsymbol{y})_i = f^\star(\boldsymbol{x}_i)$. Minimizing with respect to $\boldsymbol{\alpha}$ we get

$$\boldsymbol{\alpha}_*=\boldsymbol{y}^\top(\mathbb{K}+\lambda\mathbb{I})^{-1} \tag{2.20}$$

and so the optimal estimator can be written as

$$f(\boldsymbol{x})=\boldsymbol{y}^\top(\mathbb{K}+\lambda\mathbb{I})^{-1}\boldsymbol{k}(\boldsymbol{x}) \tag{2.21}$$

with $(\boldsymbol{k}(\boldsymbol{x}))_i = K(\boldsymbol{x}_i,\boldsymbol{x})$. Taking the limit $\lambda\to 0^+$ we recover ordinary least square kernel regression (without the regularization term), which has solution

$$f(\boldsymbol{x})=\boldsymbol{y}^\top\mathbb{K}^{-1}\boldsymbol{k}(\boldsymbol{x}). \tag{2.22}$$

Kernel Analysis of Neural Networks

Jacot et al. in [10] show an equivalence between kernel regression and training infinitely wide neural networks via gradient descent. Let $f(\boldsymbol{x};\boldsymbol{\theta})$ be the output function of deep neural network. We train this network minimizing the squared loss over the training set $\{\boldsymbol{x}_i, y_i\}_{i=1}^n$ by (full-batch) gradient descent with a vanishing learning rate η. Thus, the parameters are updated iteratively with the rule

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \nabla_{\boldsymbol{\theta}} \frac{1}{2} \sum_{i=1}^{p} \left(f(\boldsymbol{x}_i; \boldsymbol{\theta}_t) - y_i\right)^2 . \tag{2.23}$$

Taking the limit $\eta \to 0$, the evolution of the parameters follows the gradient flow equation

$$\frac{d\boldsymbol{\theta}(t)}{dt} = -\frac{1}{p} \nabla_{\boldsymbol{\theta}} \sum_{i=1}^{p} \left(f(\boldsymbol{\theta}(t), \boldsymbol{x}_i) - y_i\right)^2 . \tag{2.24}$$

The function $f(\boldsymbol{\theta}(t), \boldsymbol{x}_i)$ evolves according to

$$\frac{df(\boldsymbol{\theta}(t), \boldsymbol{x}_i)}{dt} = \left\langle \frac{\partial f(\boldsymbol{\theta}, \boldsymbol{x}_i)}{\partial \boldsymbol{\theta}(t)}, \frac{d\boldsymbol{\theta}(t)}{dt} \right\rangle \tag{2.25}$$

$$= -\sum_{j=1}^{p} \left(f(\boldsymbol{\theta}(t), \boldsymbol{x}_i) - y_i\right) \left\langle \frac{\partial f(\boldsymbol{\theta}(t), \boldsymbol{x}_i)}{\partial \boldsymbol{\theta}}, \frac{\partial f(\boldsymbol{\theta}(t), \boldsymbol{x}_j)}{\partial \boldsymbol{\theta}} \right\rangle . \tag{2.26}$$

Introducing the vector of predictors $(\boldsymbol{u}(t))_i = f(\boldsymbol{\theta}(t), \boldsymbol{x}_i)$,

$$\frac{d\boldsymbol{u}(t)}{dt} = -\Theta_h(t)(\boldsymbol{u}(t) - \boldsymbol{y}) \tag{2.27}$$

where Θ_h is a kernel matrix with elements

$$(\Theta_h(t))_{ij} = \left\langle \frac{\partial f(\boldsymbol{\theta}(t), \boldsymbol{x}_i)}{\partial \boldsymbol{\theta}}, \frac{\partial f(\boldsymbol{\theta}(t), \boldsymbol{x}_j)}{\partial \boldsymbol{\theta}} \right\rangle . \tag{2.28}$$

In the limit in which the width h of the network goes to infinity, $\Theta_h(t)$ does not evolve and thus it remains equal to $\Theta_h(0)$. Morover, in this limit, for a random initialization of the parameters this kernel converges in probability to a deterministic kernel Θ_∞, called the neural tangent kernel (NTK), and equation (2.27) reads

$$\frac{d\boldsymbol{u}(t)}{dt} = -\Theta_\infty(\boldsymbol{u}(t) - \boldsymbol{y}) \tag{2.29}$$

This dynamics corresponds to the kernel regression dynamics (with vanishing learning rate). In particular, for $t \to \infty$ the network's output is the kernel regression predictor that we found in equation (2.22), where the kernel $K(\boldsymbol{x}, \boldsymbol{x}')$ now corresponds to the NTK $\Theta_\infty(\boldsymbol{x}, \boldsymbol{x}')$. Thus, in this limit deep neural networks behave as affine models around the values of the parameters at initialization. For a full derivation of the NTK dynamics we refer the reader to [10] and [18].

Chapter 3

Theoretical Analysis

This chapter studies the impact of learning shift-invariant data on the sample complexity of fully connected and convolutional networks in the over-parametrized regime. As we discussed in the previous chapter, in this limit under some assumptions the dynamics simplifies and can be entirely described by a frozen kernel called by Jacot et al. the Neural Tangent Kernel (NTK) [10]. In the first section, we review the computation of the NTK for a two-layer fully connected network, we introduce a minimal model of a convolutional neural network, and we compute the corresponding NTK. In the second section, we study these kernels' performances in a teacher-student setting using recent results of Bordelon et al. on the average-case generalization properties of kernel regression [28].

3.1 Neural Tangent Kernels of Simple Architectures

Two-Layer Fully Connected Network

Consider a fully connected network with one hidden layer of width $h \in \mathbb{N}$

$$f^{FC}(\boldsymbol{x};\boldsymbol{\theta}) = \frac{1}{\sqrt{h}} \sum_{i=1}^{h} \beta_i \sigma(\boldsymbol{w}_i^\top \boldsymbol{x}). \tag{3.1}$$

Here $\boldsymbol{x} \in \mathbb{R}^d$ is the input, $\boldsymbol{\theta} = (\boldsymbol{w}_1^\top, \dots, \boldsymbol{w}_h^\top, \boldsymbol{\beta}^\top)$ is a vector with all the parameters $\{\boldsymbol{w}_i \in \mathbb{R}^d, \beta_i \in \mathbb{R}\}_{i=1}^h$ initialized as $\mathcal{N}(0,1)$, and $\sigma(a) = \max(0,a)$ is the ReLU activation fuction. Biases can be added by appending to the vector $\boldsymbol{x}$ an additional coordinate with value fixed to 1.
The neural tangent kernel associated to this architecture is given by

$$\Theta_h^{FC}(\boldsymbol{\theta}) = \nabla f^{FC}(\boldsymbol{\theta})^\top \nabla f^{FC}(\boldsymbol{\theta}). \tag{3.2}$$

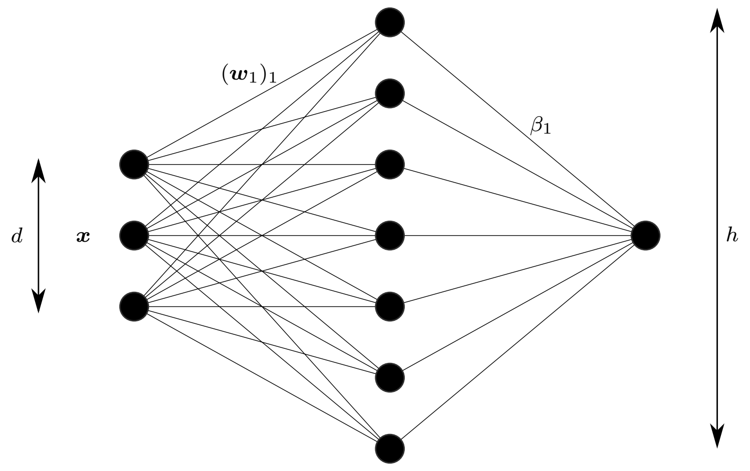


Figure 3.1: Fully connected network with one hidden layer of width h as defined in equation (3.1).

Taking the gradients,

$$\Theta_h^{FC}(\boldsymbol{x},\boldsymbol{x}';\boldsymbol{\theta}) = \frac{1}{h}\sum_{i=1}^{h}\sigma(\boldsymbol{w}_i^\top\boldsymbol{x})\sigma(\boldsymbol{w}_i^\top\boldsymbol{x}') + \frac{1}{h}\sum_{i=1}^{h}\beta_i^2\dot\sigma(\boldsymbol{w}_i^\top\boldsymbol{x})\dot\sigma(\boldsymbol{w}_i^\top\boldsymbol{x}')\boldsymbol{x}^\top\boldsymbol{x}' \tag{3.3}$$

with $\dot\sigma(a) = 1[a \geq 0]$ the derivative of the ReLU function with respect to its argument. As $h \to \infty$ this kernel converges to

$$\Theta_\infty^{FC}(\boldsymbol{x},\boldsymbol{x}') = \mathbb{E}[\sigma(\boldsymbol{w}_i^\top\boldsymbol{x})\sigma(\boldsymbol{w}_i^\top\boldsymbol{x}')] + \mathbb{E}[\beta_i^2\dot\sigma(\boldsymbol{w}_i^\top\boldsymbol{x})\dot\sigma(\boldsymbol{w}_i^\top\boldsymbol{x}')\boldsymbol{x}^\top\boldsymbol{x}']. \tag{3.4}$$

The expectation values can be computed using techniques taken form the literature of arc-cosine kernels [29],

$$\int d\boldsymbol{w}\,(2\pi)^{-\frac{d}{2}}e^{-\frac{\|\boldsymbol{w}\|^2}{2}}\sigma(\boldsymbol{w}^\top\boldsymbol{x})\sigma(\boldsymbol{w}^\top\boldsymbol{x}') = \frac{1}{2\pi}\|\boldsymbol{x}\|\|\boldsymbol{x}'\|(\sin\varphi + (\pi-\varphi)\cos\varphi) \tag{3.5}$$

$$\int d\boldsymbol{w}\,(2\pi)^{-\frac{d}{2}}e^{-\frac{\|\boldsymbol{w}\|^2}{2}}\dot\sigma(\boldsymbol{w}_i^\top\boldsymbol{x})\dot\sigma(\boldsymbol{w}_i^\top\boldsymbol{x}') = \frac{1}{2\pi}(\pi-\varphi) \tag{3.6}$$

where φ is the angle between $\boldsymbol{x}$ and $\boldsymbol{x}'$

$$\varphi = \arccos\left(\frac{\boldsymbol{x}^\top\boldsymbol{x}'}{\|\boldsymbol{x}\|\|\boldsymbol{x}'\|}\right). \tag{3.7}$$

Finally,

$$\Theta_\infty^{FC}(\boldsymbol{x},\boldsymbol{x}') = \frac{1}{2\pi}\|\boldsymbol{x}\|\|\boldsymbol{x}'\|(\sin\varphi + (\pi-\varphi)\cos\varphi) + \frac{1}{2\pi}\boldsymbol{x}^\top\boldsymbol{x}'(\pi-\varphi) \tag{3.8}$$

For further considerations on this kernel we refer the reader to [17].

Minimal Convolutional Network

Recently Arora et al. in [18] developed a method to compute the kernels induced by arbitrary convolutional networks (CNTK), and showed experimentally that these achieve very good performances. Compared with their work, here we focus on a much simpler and theoretically tractable convolutional architecture, intending to study the generalization capabilities of these kernels analytically.
First, we introduce the discrete shift operator t_a, which applied to a vector $\boldsymbol{v} \in \mathbb{R}^d$ shifts circularly (i.e. with periodic boundary conditions) all the vector's components by a positions

$$(t_a[\boldsymbol{v}])_i = (\boldsymbol{v})_{(i+a) \bmod d} \,. \tag{3.9}$$

For instance, if $\boldsymbol{x} \in \mathbb{R}^d$ is a one-dimensional image with d pixels, $t_a[\boldsymbol{x}]$ shifts all the pixels to the right a times, filling the first pixel with the one that is shifted out of the array each time.
Taking inspiration from the previous fully connected architecture, we define a minimal model of a convolutional neural network with one convolutional layer made of h channels, filters of size d, periodic padding, stride one, and average pooling

$$f^{CN}(\boldsymbol{x};\boldsymbol{\theta}) = \frac{1}{\sqrt{h}} \sum_{i=1}^{h} \beta_i \frac{1}{d} \sum_a \sigma(t_a[\boldsymbol{w}_i]^\top \boldsymbol{x}) \tag{3.10}$$

where the second sum is over the d possible shifts. Since $t_a[\boldsymbol{w}_i]^\top \boldsymbol{x} = \boldsymbol{w}_i^\top t_{-a}[\boldsymbol{x}]$, the output of this network is shift-invariant by construction

$$f^{CN}(t_a[\boldsymbol{x}];\boldsymbol{\theta}) = f^{CN}(\boldsymbol{x};\boldsymbol{\theta}). \tag{3.11}$$

Switching the action of the shift operator from the weights to the inputs, the associated neural tangent kernel reads

$$\begin{aligned}\Theta_h^{CN}(\boldsymbol{x},\boldsymbol{x}';\boldsymbol{\theta}) = \frac{1}{d^2}\sum_a\sum_b \Bigg(&\frac{1}{h}\sum_{i=1}^{h} \sigma(\boldsymbol{w}_i^\top t_a[\boldsymbol{x}])\sigma(\boldsymbol{w}_i^\top t_b[\boldsymbol{x}']) \\ &+ \frac{1}{h}\sum_{i=1}^{h} \beta_i^2 \dot\sigma(\boldsymbol{w}_i^\top t_a[\boldsymbol{x}])\dot\sigma(\boldsymbol{w}_i^\top t_b[\boldsymbol{x}']) t_a[\boldsymbol{x}]^\top t_b[\boldsymbol{x}']\Bigg)\,.\end{aligned} \tag{3.12}$$

In the limit $h \to \infty$,

$$\begin{aligned}\Theta_\infty^{CN}(\boldsymbol{x},\boldsymbol{x}') = \frac{1}{d^2}\sum_a\sum_b \Big(&\mathbb{E}[\sigma(\boldsymbol{w}_i^\top t_a[\boldsymbol{x}])\sigma(\boldsymbol{w}_i^\top t_b[\boldsymbol{x}'])) \\ &+ \mathbb{E}[\beta_i^2 \dot\sigma(\boldsymbol{w}_i^\top t_a[\boldsymbol{x}])\dot\sigma(\boldsymbol{w}_i^\top t_b[\boldsymbol{x}']) t_a[\boldsymbol{x}]^\top t_b[\boldsymbol{x}']]\Big)\,.\end{aligned} \tag{3.13}$$

Since the shift operator does not affect norms and inner products of two shifted vectors depend only on the relative shift between them, we get

$$\begin{aligned}\Theta_\infty^{CN}(\boldsymbol{x},\boldsymbol{x}') = \frac{1}{d}\sum_a \Bigg(&\frac{1}{2\pi}\|\boldsymbol{x}\|\|t_a[\boldsymbol{x}']\|(\sin\varphi_a + (\pi-\varphi_a)\cos\varphi_a) \\ &+ \frac{1}{2\pi}\boldsymbol{x}^\top t_a[\boldsymbol{x}'](\pi-\varphi_a)\Bigg)\end{aligned} \tag{3.14}$$

with φ_a the angle between $\boldsymbol{x}$ and $t_a[\boldsymbol{x}']$. Recognizing the form of the neural tangent kernel of the two-layer fully connected network, we can write in compact form

$$\Theta_\infty^{CN}(\boldsymbol{x},\boldsymbol{x}') = \frac{1}{d}\sum_a \Theta_\infty^{FN}(\boldsymbol{x}, t_a[\boldsymbol{x}']). \tag{3.15}$$

Intuitively, a kernel corresponds to a notion of similarity. In contrast with the vanilla NTK $\Theta_\infty^{FN}(\boldsymbol{x},\boldsymbol{x}')$, the convolutional NTK $\Theta_\infty^{CN}(\boldsymbol{x},\boldsymbol{x}')$ encodes a shift-invariant notion of similarity. Since images are inherently shift-invariant (e.g. a dog remains a dog regardless of its position in a picture), we expect this convolutional NTK to perform better than the vanilla NTK for image classification and shift-invariant pattern recognition tasks.

3.2 NTK Regression in a Teacher-Student Setting

We study the performance of the neural tangent kernels of the fully connected and the convolutional networks in a teacher-student setting. In the statistical physics of machine learning literature, this setting corresponds to training a student neural network on data generated by a teacher neural network. Here, we leverage the equivalence between these models in the infinite-width limit and interpolating kernel regression, and instead of the neural networks we use the corresponding neural tangent kernels to generate and learn the data. This teacher-student setting for kernel regression was introduced first by Sollich in [30, 31], and recently adopted by Spigler et at. in [32] to study the learning curves of isotropic kernels, and by Paccolat et al. in [33] to study how kernel methods learn simple invariant tasks. In the following, we will address how the kernels that we derived in the previous section deal with invariance by translations.

Setup

We generate the target function $f^\star$ using a positive definite teacher kernel K_T to sample a Gaussian random field

$$f^\star(\boldsymbol{x}) \sim \mathcal{N}(0, K_T). \tag{3.16}$$

The training set consists of p examples $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^p$, with $\boldsymbol{x}_i$ sampled uniformly in the d-dimensional hypercube $\mathcal{V}_d = [-\frac{L}{2}, \frac{L}{2}]^d$, and $y_i = f^\star(\boldsymbol{x}_i)$.
The goal of kernel regression is to infer the value of the field $f(\boldsymbol{x})$ at an out-of-sample point $\boldsymbol{x}$ using a positive definite student kernel K_S and minimizing the empirical mean squared error

$$\min_{f \in \mathcal{H}_{K_S}} \sum_{i=1}^{p} (f(\boldsymbol{x}_i) - y_i)^2, \tag{3.17}$$

with $\mathcal{H}_{K_S}$ the Reproducing Kernel Hilbert Space associated to K_S. The minimizer of this convex problem is unique, and can be written using the representer theorem as

$$f(\boldsymbol{x}) = \boldsymbol{y}^\top \mathbb{K}_S^{-1} \boldsymbol{k}_S(\boldsymbol{x}) \tag{3.18}$$

where $\boldsymbol{y}$ is the vector with the target values y_i, $\mathbb{K}_S$ is the $p \times p$ Gram matrix with elements $(\mathbb{K}_S)_{ij} = K_S(\boldsymbol{x}_i, \boldsymbol{x}_j)$, and $\boldsymbol{k}_S(\boldsymbol{x})$ is a vector with elements $(\boldsymbol{k}_S(\boldsymbol{x}))_i = K_S(\boldsymbol{x}_i, \boldsymbol{x})$. Notice that the Gram matrix is always invertible since K_S is positive definite. In order to estimate the performance, we compute the asymptotic behaviour with respect to the number of training samples p of the average-case generalization error, which we define as

$$E_g = \mathbb{E}\left[\int d\mu(\boldsymbol{x})\, (f(\boldsymbol{x}) - f^\star(\boldsymbol{x}))^2\right] \tag{3.19}$$

where the expectation is taken with respect to the teacher random process, and $d\mu(\boldsymbol{x})$ is the probability measure with which points are generated (in our setting the uniform distribution in the hypercube $\mathcal{V}_d$). To estimate this error we compute its spectral decomposition in the eigenbasis of the student kernel. Indeed, using Mercer's theorem it is possible to decompose K_S in terms of its eigenfunctions $\phi_\rho(\boldsymbol{x})$

$$\int d\mu(\boldsymbol{x}')\, K_S(\boldsymbol{x}, \boldsymbol{x}') \phi_\rho(\boldsymbol{x}') = \lambda_\rho \phi_\rho(\boldsymbol{x}) \tag{3.20}$$

$$K_S(\boldsymbol{x}, \boldsymbol{x}') = \sum_\rho \lambda_\rho \phi_\rho(\boldsymbol{x}) \phi_\rho(\boldsymbol{x}') \tag{3.21}$$

$$= \sum_\rho \psi_\rho(\boldsymbol{x}) \psi_\rho(\boldsymbol{x}') \tag{3.22}$$

where $\psi_\rho(\boldsymbol{x}) = \sqrt{\lambda_\rho}\phi_\rho(\boldsymbol{x})$ are the kernel features. Since the kernel eigenfunctions form an eigenbasis of the Reproducing Kernel Hilbert Space $\mathcal{H}_{K_S}$, we can expand the target function $f^\star(\boldsymbol{x})$ and the learned function $f(\boldsymbol{x})$ in terms of the kernel features

$$f^\star(\boldsymbol{x}) = \sum_\rho w^\star_\rho \psi_\rho(\boldsymbol{x}) \tag{3.23}$$

$$w^\star_\rho = \frac{1}{\sqrt{\lambda_\rho}}\langle f^\star, \phi_\rho\rangle \tag{3.24}$$

$$f(\boldsymbol{x}) = \sum_\rho w_\rho \psi_\rho(\boldsymbol{x}) \tag{3.25}$$

$$w_\rho = \frac{1}{\sqrt{\lambda_\rho}}\langle f, \phi_\rho\rangle. \tag{3.26}$$

Using this decomposition, Bordelon et al. in [28] showed that the average-case generalization error associated to each mode ρ can be written as

$$E_\rho(p) = \frac{\mathbb{E}[w^{\star 2}_\rho]}{\lambda_\rho}\left(\frac{1}{\lambda_\rho} + \frac{p}{t(p)}\right)^{-2}\left(1 - \frac{p\gamma(p)}{t^2(p)}\right)^{-1} \tag{3.27}$$

$$t(p) = \sum_\rho \left(\frac{1}{\lambda_\rho} + \frac{p}{t(p)}\right)^{-1} \tag{3.28}$$

$$\gamma(p) = \sum_\rho \left(\frac{1}{\lambda_\rho} + \frac{p}{t(p)}\right)^{-2} \tag{3.29}$$

$$E_g(p) = \sum_\rho E_\rho(p). \tag{3.30}$$

In order to simplify the analytical computations, in what follows we approximate the NTK of the two layer fully connected network $\Theta^{FC}_\infty(\boldsymbol{x}, \boldsymbol{x}')$ with the Laplacian kernel

$$\mathrm{LAP}(\boldsymbol{x}, \boldsymbol{x}') = e^{-\|\boldsymbol{x}-\boldsymbol{x}'\|} \tag{3.31}$$

which has a simpler close form. Indeed, as shown in [34], these two kernels for data normalized on the hypersphere $\mathbb{S}^{d-1}$ have the same eigenfunctions and their eigenvalues decay with the same power law. Furthermore, for real data these kernels behave empirically in the same way. We define the convolutional version of the Laplacian kernel corresponding to $\Theta^{CN}_\infty(\boldsymbol{x}, \boldsymbol{x}')$ as

$$\mathrm{CLAP}(\boldsymbol{x}, \boldsymbol{x}') = \frac{1}{d}\sum_a \mathrm{LAP}(\boldsymbol{x}, t_a[\boldsymbol{x}']). \tag{3.32}$$

To check the validity of this approximation, in the next chapter we present numerical experiments showing that doing regression with these kernels and training the two previously defined architectures in the regime in which they can be described by their NTKs leads to the same results.

Laplacian Teacher

First, we consider the case of using the Laplacian kernel both as the teacher kernel K_T and the student kernel K_S. Since this kernel is isotropic, i.e. it depends only on the difference of its arguments, we can diagonalize it using its Fourier decomposition on the hypercube $\mathcal{V}_d$. Therefore, choosing as eigenfunctions the normalized plane waves

$$\phi_{\boldsymbol{k}}(\boldsymbol{x}) = L^{-\frac{d}{2}} e^{i\boldsymbol{k}^\top \boldsymbol{x}} \tag{3.33}$$

we can write

$$\mathrm{LAP}(\boldsymbol{x}, \boldsymbol{x}') = \sum_{\boldsymbol{k}} \lambda_{\boldsymbol{k}} \phi_{\boldsymbol{k}}(\boldsymbol{x}) \overline{\phi_{\boldsymbol{k}}(\boldsymbol{x}')} \tag{3.34}$$

where the bar denotes complex conjugation, and the eigenvalues $\lambda_{\boldsymbol{k}}$ are given by the Fourier transform of the kernel computed in $\boldsymbol{k} \in \frac{2\pi}{L}\mathbb{Z}^d$

$$\lambda_{\boldsymbol{k}} = L^{-\frac{d}{2}} \int_{\mathcal{V}_d} d\boldsymbol{x}\; e^{-\|\boldsymbol{x}\|} e^{-i\boldsymbol{k}^\top \boldsymbol{x}}. \tag{3.35}$$

Computing the integral

$$\lambda_{\boldsymbol{k}} = \frac{L^{-\frac{d}{2}} \Gamma(\frac{d+1}{2})}{\pi^{\frac{d+1}{2}}} \frac{1}{(1 + \frac{1}{4\pi^2}\|\boldsymbol{k}\|^2)^{\frac{d+1}{2}}} \tag{3.36}$$

which asymptotically decays as $\|\boldsymbol{k}\|^{-\alpha}$, with $\alpha = d+1$.
In order to compute the average-case generalization error, since our eigenfunctions are complex we need to generalize equation (3.27) to complex valued coefficients, obtaining

$$E_{\boldsymbol{k}}(p) = \frac{\mathbb{E}[|w^\star_{\boldsymbol{k}}|^2]}{\lambda_{\boldsymbol{k}}} \left(\frac{1}{\lambda_{\boldsymbol{k}}} + \frac{p}{t(p)}\right)^{-2} \left(1 - \frac{p\gamma(p)}{t^2(p)}\right)^{-1} \tag{3.37}$$

with $t(p)$ and $\gamma(p)$ defined as before. Thus, we need to compute $\mathbb{E}[|w^\star_{\boldsymbol{k}}|^2]$, $t(p)$, and $\gamma(p)$.
The expectation of the squared-modulus of the target function coefficients in the

eigenbasis of the student kernel is given by

$$\mathbb{E}[|w_{\boldsymbol{k}}^\star|^2] = \frac{1}{\lambda_{\boldsymbol{k}}}\int_{\mathcal{V}_d} d\boldsymbol{x}\ \phi_{\boldsymbol{k}}(\boldsymbol{x})\int_{\mathcal{V}_d} d\boldsymbol{x}'\ \overline{\phi_{\boldsymbol{k}}(\boldsymbol{x}')}\mathbb{E}[f^\star(\boldsymbol{x})f^\star(\boldsymbol{x}')] \tag{3.38}$$

$$= \frac{1}{\lambda_{\boldsymbol{k}}}\int_{\mathcal{V}_d} d\boldsymbol{x}\ \phi_{\boldsymbol{k}}(\boldsymbol{x})\int_{\mathcal{V}_d} d\boldsymbol{x}'\ \overline{\phi_{\boldsymbol{k}}(\boldsymbol{x}')}\mathrm{LAP}(\boldsymbol{x},\boldsymbol{x}') \tag{3.39}$$

$$= \frac{1}{\lambda_{\boldsymbol{k}}}\int_{\mathcal{V}_d} d\boldsymbol{x}\ \phi_{\boldsymbol{k}}(\boldsymbol{x})\lambda_{\boldsymbol{k}}\overline{\phi_{\boldsymbol{k}}(\boldsymbol{x})} \tag{3.40}$$

$$= 1. \tag{3.41}$$

To compute $t(p)$ and $\gamma(p)$ we take a continuum limit, and we approximate the sums over the wave-vectors $\boldsymbol{k}$'s with integrals over the eigenvalues. To do so, we introduce the eigenvalue density

$$\rho(\lambda) = \sum_{\boldsymbol{k}} \delta_{\lambda,\lambda_{\boldsymbol{k}}} \tag{3.42}$$

$$\sim \sum_{\boldsymbol{k}} \delta_{\lambda,\|\boldsymbol{k}\|^{-\alpha}} \tag{3.43}$$

$$\sim \int d\boldsymbol{k}\ \delta(\lambda - \|\boldsymbol{k}\|^{-\alpha}). \tag{3.44}$$

Going to spherical coordinates in d dimensions and setting $q = \|\boldsymbol{k}\|$

$$\rho(\lambda) \sim \int_0^\infty dq\ q^{d-1}\delta(\lambda - q^{-\alpha}) \tag{3.45}$$

$$\sim \lambda^{-1-\frac{d}{\alpha}} \tag{3.46}$$

$$\sim \lambda^{-\theta} \tag{3.47}$$

with the exponent $\theta = 1 + \frac{d}{\alpha}$.
Therefore, from equation (3.28)

$$t(p) \sim \int_0^1 d\lambda\ \lambda^{-\theta}\frac{\lambda}{1 + \frac{p}{t(p)}\lambda} \tag{3.48}$$

where we have taken the continuum limit and rescaled the eigenvalues so that the largest equals one (this only affects with a constant prefactor). From this implicit equation it follows that $t(p)$ has to go to zero as $p \to \infty$. Thus, we can compute its asymptotic behaviour assuming it to be small and separating the integration domain in the two intervals where the denominator is dominated by the first or the second term of the sum

$$t(p) \sim \int_0^{t(p)/p} d\lambda\, \lambda^{-\theta+1} + \frac{t(p)}{p} \int_{t(p)/p}^{1} d\lambda\, \lambda^{-\theta} \tag{3.49}$$

$$\sim \left(\frac{t(p)}{p}\right)^{-\theta+2}. \tag{3.50}$$

Finally, rearranging the factors

$$t(p) \sim p^{-\frac{\theta-2}{1-\theta}} \tag{3.51}$$

$$\sim p^{-\chi} \tag{3.52}$$

with the exponent $\chi = \frac{\theta-2}{1-\theta}$. Then, we can insert this scaling relation in equation (3.29) and proceed as before to compute $\gamma(p)$

$$\gamma(p) \sim \int_0^1 d\lambda\, \lambda^{2-\theta} \frac{1}{(1+p^{\chi+1}\lambda)^2} \tag{3.53}$$

$$\sim p^{-\frac{\theta-3}{1-\theta}}. \tag{3.54}$$

Substituting in equation (3.37), summing over all modes, and taking the continuum limit again

$$E_g \sim \int_0^1 d\lambda\, \lambda^{-\theta+1}(1+p^{-\frac{1}{1-\theta}}\lambda)^{-2} \tag{3.55}$$

$$\sim p^{-\frac{\theta-2}{1-\theta}}. \tag{3.56}$$

Recalling that $\theta = 1 + \frac{1}{\alpha}$, and for the Laplacian kernel $\alpha = 1 + d$, we obtain

$$E_g \sim p^{-\frac{1}{d}}, \tag{3.57}$$

which is a manifestation of the curse of dimensionality that we mentioned in the background chapter.

Convolutional Laplacian Teacher

In the following, we investigate whether the presence of translational invariance in the target function can cure this curse of dimensionality. Therefore, we consider a shift-invariant kernel teacher (the convolutional Laplacian), and we compute the generalization error when using a vanilla Laplacian kernel or a convolutional Laplacian kernel as a student. In the first case the student kernel is not aware of the invariant's presence, while in the second case it has the correct prior.

For the Laplacian student, we already know the eigendecomposition, and we can proceed as in the previous case.

$$\mathbb{E}[|w_{\boldsymbol{k}}^{\star}|^2] = \frac{1}{\lambda_{\boldsymbol{k}}} \int_{\mathcal{V}_d} d\boldsymbol{x}\, \phi_{\boldsymbol{k}}(\boldsymbol{x}) \int_{\mathcal{V}_d} d\boldsymbol{x}'\, \overline{\phi_{\boldsymbol{k}}(\boldsymbol{x}')} \mathbb{E}[f^{\star}(\boldsymbol{x}) f^{\star}(\boldsymbol{x}')] \tag{3.58}$$

$$= \frac{1}{\lambda_{\boldsymbol{k}}} \int_{\mathcal{V}_d} d\boldsymbol{x}\, \phi_{\boldsymbol{k}}(\boldsymbol{x}) \int_{\mathcal{V}_d} d\boldsymbol{x}'\, \overline{\phi_{\boldsymbol{k}}(\boldsymbol{x}')} \mathrm{CLAP}(\boldsymbol{x}, \boldsymbol{x}') \tag{3.59}$$

$$= \frac{1}{\lambda_{\boldsymbol{k}}} \int_{\mathcal{V}_d} d\boldsymbol{x}\, \phi_{\boldsymbol{k}}(\boldsymbol{x}) \int_{\mathcal{V}_d} d\boldsymbol{x}'\, \overline{\phi_{\boldsymbol{k}}(\boldsymbol{x}')} \frac{1}{d} \sum_a \mathrm{LAP}(\boldsymbol{x}, t_a[\boldsymbol{x}']) \tag{3.60}$$

$$= \frac{1}{d} \sum_a \delta_{\boldsymbol{k}, t_a[\boldsymbol{k}]} \tag{3.61}$$

$$t(p) \sim p^{-\frac{\theta-2}{1-\theta}} \tag{3.62}$$

$$\gamma(p) \sim p^{-\frac{\theta-3}{1-\theta}}. \tag{3.63}$$

Thus, the average-case generalization scales as

$$E_g \sim \frac{1}{d} \sum_{\boldsymbol{k}} \sum_a \delta_{\boldsymbol{k}, t_a[\boldsymbol{k}]} \frac{1}{\lambda_{\boldsymbol{k}}} \left(\frac{1}{\lambda_{\boldsymbol{k}}} + p^{-\frac{1}{1-\theta}} \right)^{-2} \tag{3.64}$$

$$\sim \frac{1}{d} \int_0^1 d\lambda\, \lambda^{-\theta+1} (1 + p^{-\frac{1}{1-\theta}} \lambda)^{-2} \tag{3.65}$$

$$\sim \frac{1}{d} p^{-\frac{\theta-2}{1-\theta}}, \tag{3.66}$$

where we used the fact that the fraction of $\boldsymbol{k}$'s for which $\sum_a \delta_{\boldsymbol{k}, t_a[\boldsymbol{k}]} \neq 1$ is a null subset, and so it can be safely neglected. Finally, using the value of θ of the Laplacian, we get $E_g \sim \frac{1}{d} p^{-\frac{1}{d}}$. So, if the student has no prior for the translational invariance of the teacher, the exponent of the error remains the same (the curse of dimensionality applies again), but we gain a constant prefactor $\frac{1}{d}$.

In contrast with the previous student kernel, the convolutional Laplacian is not isotropic, therefore we cannot diagonalize it with plane waves. Thus, we introduce the equivalence class

$$[\boldsymbol{k}] = \{\boldsymbol{k}' \in \mathcal{K} \mid t_a[\boldsymbol{k}'] = \boldsymbol{k}\} \tag{3.67}$$

where $\mathcal{K} = \frac{2\pi}{L}\mathbb{Z}^d$ is the set of all the allowed wave-vectors, and we define the shift-invariant eigenfunctions

$$\phi_{\boldsymbol{k}}^C(\boldsymbol{x}) = \frac{1}{\sqrt{L^d d \sum_a \delta_{\boldsymbol{k}, t_a[\boldsymbol{k}]}}} \sum_a e^{i t_a[\boldsymbol{k}]^\top \boldsymbol{x}} \tag{3.68}$$

restricting $\boldsymbol{k}$ to the partition set of equivalent classes and therefore considering only one element per class since all the $\boldsymbol{k}' \in [\boldsymbol{k}]$ have the same eigenfunction $\phi^C_{\boldsymbol{k}}(\boldsymbol{x})$. These eigenfunctions are orthonormal, indeed the inner product between two eigenfunctions is zero if $[\boldsymbol{k}] \neq [\boldsymbol{k}']$

$$\langle \phi^C_{\boldsymbol{k}}, \phi^C_{\boldsymbol{k}'} \rangle \propto \sum_a \delta_{\boldsymbol{k}, t_a[\boldsymbol{k}']} = 0 \tag{3.69}$$

and is one if $[\boldsymbol{k}] = [\boldsymbol{k}']$, i.e. $\boldsymbol{k} = \boldsymbol{k}'$ since we are considering only one element per equivalence class,

$$\langle \phi^C_{\boldsymbol{k}}, \phi^C_{\boldsymbol{k}} \rangle = 1. \tag{3.70}$$

All in all,

$$\langle \phi^C_{\boldsymbol{k}}, \phi^C_{\boldsymbol{k}'} \rangle = \delta_{\boldsymbol{k}, \boldsymbol{k}'}. \tag{3.71}$$

These eigenfunctions form a complete basis. Indeed, considering the Fourier expansion of the convolutional kernel

$$\begin{aligned}
\mathrm{CLAP}(\boldsymbol{x}, \boldsymbol{x}') &= \frac{1}{d} \sum_a e^{\|\boldsymbol{x} - t_a[\boldsymbol{x}']\|} && (3.72)\\
&= \frac{1}{d} \sum_a \sum_{\boldsymbol{k}} \lambda_{\boldsymbol{k}} L^{-d} e^{i\boldsymbol{k}^\top (\boldsymbol{x} - t_a[\boldsymbol{x}'])} && (3.73)\\
&= \sum_{\boldsymbol{k}} \lambda_{\boldsymbol{k}} \frac{1}{d} \sum_a L^{-d} e^{i\boldsymbol{k}^\top (\boldsymbol{x} - t_a[\boldsymbol{x}'])} && (3.74)\\
&= \sum_{\boldsymbol{k}} \lambda_{\boldsymbol{k}} \frac{1}{d^2} \sum_a \sum_b L^{-d} e^{i\boldsymbol{k}^\top (t_a[\boldsymbol{x}] - t_b[\boldsymbol{x}'])} && (3.75)\\
&= \sum_{\boldsymbol{k}} \lambda_{\boldsymbol{k}} \frac{1}{d^2} L^{-\frac{d}{2}} \sum_a e^{it_a[\boldsymbol{k}]^\top \boldsymbol{x}} L^{-\frac{d}{2}} \sum_b e^{-it_b[\boldsymbol{k}]^\top \boldsymbol{x}'} && (3.76)\\
&= \sum_{\boldsymbol{k}} \lambda_{\boldsymbol{k}} \frac{\sum_a \delta_{\boldsymbol{k}, t_a[\boldsymbol{k}]}}{d} \frac{1}{\sqrt{L^d d \sum_a \delta_{\boldsymbol{k}, t_a[\boldsymbol{k}]}}} \sum_a e^{it_a[\boldsymbol{k}]^\top \boldsymbol{x}} \\
&\times \frac{1}{\sqrt{L^d d \sum_a \delta_{\boldsymbol{k}, t_a[\boldsymbol{k}]}}} \sum_b e^{-it_b[\boldsymbol{k}]^\top \boldsymbol{x}'} && (3.77)\\
&= \sum_{[\boldsymbol{k}]} \lambda_{\boldsymbol{k}} \phi^C_{\boldsymbol{k}}(\boldsymbol{x}) \overline{\phi^C_{\boldsymbol{k}}(\boldsymbol{x}')} && (3.78)
\end{aligned}$$

where with a slight abuse of notation the sum over $[\boldsymbol{k}]$ indicates the sum over one element per equivalence class, and $\lambda_{\boldsymbol{k}}$ are the eigenvalues of the Laplacian kernel, which depend only on the modulus of $\boldsymbol{k}$ and thus satisfy $\lambda_{t_a[\boldsymbol{k}]} = \lambda_{\boldsymbol{k}}$.

Finally, we prove that these eigenfunctions satisfy the kernel eigenvalue problem

$$\int d\mu(\boldsymbol{x}') \,\mathrm{CLAP}(\boldsymbol{x},\boldsymbol{x}')\phi^C_{\boldsymbol{k}}(\boldsymbol{x}') = \int d\mu(\boldsymbol{x}')\,\frac{1}{d}\sum_a e^{\|\boldsymbol{x}-t_a[\boldsymbol{x}']\|}\frac{1}{\sqrt{L^d d\sum_a \delta_{\boldsymbol{k},t_a[\boldsymbol{k}]}}} \times \sum_b e^{it_b[\boldsymbol{k}]^\top \boldsymbol{x}'} \tag{3.79}$$

$$= \frac{1}{d}\frac{1}{\sqrt{L^d d\sum_a \delta_{\boldsymbol{k},t_a[\boldsymbol{k}]}}}\sum_a\sum_b\int d\mu(\boldsymbol{x}')\, e^{\|\boldsymbol{x}-t_a[\boldsymbol{x}']\|} \times e^{it_b[\boldsymbol{k}]^\top \boldsymbol{x}'} \tag{3.80}$$

$$= \frac{1}{d}\frac{1}{\sqrt{L^d d\sum_a \delta_{\boldsymbol{k},t_a[\boldsymbol{k}]}}}\sum_a\sum_b\int d\mu(\boldsymbol{x}')\, e^{\|\boldsymbol{x}-t_a[\boldsymbol{x}']\|} \times e^{i\boldsymbol{k}^\top t_b[\boldsymbol{x}']} \tag{3.81}$$

$$= \frac{1}{\sqrt{L^d d\sum_a \delta_{\boldsymbol{k},t_a[\boldsymbol{k}]}}}\sum_a\int d\mu(\boldsymbol{x}')\, e^{\|\boldsymbol{x}-\boldsymbol{x}'\|}e^{i\boldsymbol{k}^\top t_a[\boldsymbol{x}']} \tag{3.82}$$

$$= \frac{1}{\sqrt{L^d d\sum_a \delta_{\boldsymbol{k},t_a[\boldsymbol{k}]}}}\sum_a\int d\mu(\boldsymbol{x}')\, e^{\|\boldsymbol{x}-\boldsymbol{x}'\|}e^{it_a[\boldsymbol{k}]^\top \boldsymbol{x}'} \tag{3.83}$$

$$= \frac{1}{\sqrt{L^d d\sum_a \delta_{\boldsymbol{k},t_a[\boldsymbol{k}]}}}\sum_a \lambda_{t_a[\boldsymbol{k}]}e^{it_a[\boldsymbol{k}]^\top \boldsymbol{x}} \tag{3.84}$$

$$= \lambda_{\boldsymbol{k}}\frac{1}{\sqrt{L^d d\sum_a \delta_{\boldsymbol{k},t_a[\boldsymbol{k}]}}}\sum_a e^{it_a[\boldsymbol{k}]^\top \boldsymbol{x}} \tag{3.85}$$

$$= \lambda_{\boldsymbol{k}}\phi^C_{\boldsymbol{k}}(\boldsymbol{x}). \tag{3.86}$$

Now we can proceed to compute the average-case generalization error.

$$\mathbb{E}[|w^\star_{\boldsymbol{k}}|^2] = \frac{1}{\lambda_{\boldsymbol{k}}}\int_{\mathcal{V}_d} d\boldsymbol{x}\;\phi^C_{\boldsymbol{k}}(\boldsymbol{x})\int_{\mathcal{V}_d} d\boldsymbol{x}'\;\overline{\phi^C_{\boldsymbol{k}}(\boldsymbol{x}')}\mathbb{E}[f^\star(\boldsymbol{x})f^\star(\boldsymbol{x}')] \tag{3.87}$$

$$= \frac{1}{\lambda_{\boldsymbol{k}}}\int_{\mathcal{V}_d} d\boldsymbol{x}\;\phi^C_{\boldsymbol{k}}(\boldsymbol{x})\int_{\mathcal{V}_d} d\boldsymbol{x}'\;\overline{\phi^C_{\boldsymbol{k}}(\boldsymbol{x}')}\;\mathrm{CLAP}(\boldsymbol{x},\boldsymbol{x}') \tag{3.88}$$

$$= \frac{1}{\lambda_{\boldsymbol{k}}}\int_{\mathcal{V}_d} d\boldsymbol{x}\;\phi^C_{\boldsymbol{k}}(\boldsymbol{x})\lambda_{\boldsymbol{k}}\overline{\phi^C_{\boldsymbol{k}}(\boldsymbol{x})} \tag{3.89}$$

$$= 1. \tag{3.90}$$

In order to compute $t(p)$, $\gamma(p)$ and $E_g(p)$, we recall that the summations which appear are not over all the possible wave-vectors $\boldsymbol{k}$, but only over one element per

equivalence class $[\boldsymbol{k}]$. As we already commented before, the majority of wave-vectors does not have any special symmetry for shifts of the components. Therefore, it is possible to convert the series over $[\boldsymbol{k}]$ in series over $\boldsymbol{k}$ simply dividing by d, since the elements for which $\mathrm{card}[\boldsymbol{k}] \neq d$ belong to a null subset. In order to keep track of possible changes depending on d in the pre-factors, we keep all the constants which depend explicitly on the dimension.
With the ansatz $t(p) = \mathcal{C}p^{-\chi}$

$$\mathcal{C}p^{-\chi} \sim \sum_{[\boldsymbol{k}]} \left(\frac{1}{\lambda_{\boldsymbol{k}}} + \frac{p}{\mathcal{C}p^{-\chi}} \right)^{-1} \tag{3.91}$$

$$\sim \frac{1}{d} \sum_{\boldsymbol{k}} \frac{\mathcal{C}\lambda_{\boldsymbol{k}}}{\mathcal{C} + p^{\chi+1}\lambda_{\boldsymbol{k}}}. \tag{3.92}$$

Taking the continuum limit

$$\mathcal{C}p^{-\chi} \sim \frac{1}{d} \int_0^1 d\lambda\, \lambda^{-\theta} \frac{\mathcal{C}\lambda}{\mathcal{C} + p^{\chi+1}\lambda} \tag{3.93}$$

$$\sim \frac{\mathcal{C}^{2-\theta}}{d} p^{-(\chi+1)(2-\theta)}. \tag{3.94}$$

Comparing the exponents and the coefficients

$$\chi = \frac{\theta - 2}{1-\theta}, \quad \mathcal{C} \sim d^{-\frac{1}{\theta-1}}, \quad t(p) \sim d^{-\frac{1}{\theta-1}} p^{-\frac{\theta-2}{1-\theta}}.$$

Inserting in equation (3.29)

$$\gamma(p) \sim \sum_{[\boldsymbol{k}]} \left(\frac{1}{\lambda_{\boldsymbol{k}}} + \frac{p}{\mathcal{C}p^{-\chi}} \right)^{-2} \tag{3.95}$$

$$\sim \frac{\mathcal{C}^2}{d} \int_0^1 d\lambda\, \lambda^{2-\theta} \frac{1}{(\mathcal{C} + p^{\chi+1}\lambda)^2} \sim d^{-\frac{2}{\theta-1}} p^{-\frac{\theta-3}{1-\theta}}. \tag{3.96}$$

Hence, the average-generalization error reads

$$E_g \sim \sum_{[\boldsymbol{k}]} \frac{1}{\lambda_{\boldsymbol{k}}} \left(\frac{1}{\lambda_{\boldsymbol{k}}} + d^{-\frac{1}{\theta-1}} p^{-\frac{1}{1-\theta}} \right)^{-2} \tag{3.97}$$

$$\sim \frac{1}{d} \sum_{\boldsymbol{k}} \frac{1}{\lambda_{\boldsymbol{k}}} \left(\frac{1}{\lambda_{\boldsymbol{k}}} + d^{-\frac{1}{\theta-1}} p^{-\frac{1}{1-\theta}} \right)^{-2} \tag{3.98}$$

$$\sim \frac{1}{d} \int_0^1 d\lambda\, \lambda^{-\theta+1} (1 + d^{-\frac{1}{\theta-1}} p^{-\frac{1}{1-\theta}} \lambda)^{-2} \tag{3.99}$$

$$\sim d^{-\frac{1}{\theta-1}} p^{-\frac{\theta-2}{1-\theta}}. \tag{3.100}$$

This error decays again with exponent $\frac{1}{d}$ and so it is still affected by the curse of dimensionality. The prefactor is smaller than the one obtained using the Laplacian student kernel when

$$d^{-\frac{1}{\theta-1}} < d^{-1} \tag{3.101}$$

$$\frac{1}{\theta-1} > 1 \tag{3.102}$$

$$\theta < 2 \tag{3.103}$$

and since for these kernels $\theta = 1 + \frac{d}{d+1}$ and so $1 \leq \theta < 2$, it follows that the error of the convolutional student is always the lowest one.
Therefore, our results seem to confirm for translational invariance what Paccolat et al. already showed in [33] for simple invariant task, that is kernel regression is not able to compress invariant dimensions and escape the curse of dimensionality. In spite of this, the presence of invariance for translations reflects in the pre-factors of the typical generalization error, lowering it for student kernels associated to both fully connected and convolutional architectures, which having the correct prior perform best.

Spectral Bias

Bordelon et al. in [28] observed that the modal errors E_ρ are learnt at different rates. In particular, adding a single training example causes a bigger relative reduction in the modal errors corresponding to higher eigenvalues of the student kernel, in the sense that if $\lambda_\rho > \lambda_\gamma$

$$\left| \frac{d}{dp} \log E_\rho \right| > \left| \frac{d}{dp} \log E_\gamma \right| . \tag{3.104}$$

Therefore, larger eigenvalues are learnt faster, and the way in which the spectrum decays controls how much information is learnt. Then, they showed that using p data sampled uniformly on an infinite-dimensional hypersphere, dot-product kernels learn modes with $\rho \leq p$ and don't learn at all modes with $\rho > p$. An equivalent result can be obtained with our formalism (see [32] for an analogous analysis on isotropic kernels). Since

$$p \sim \int_{\lambda_p}^{1} d\lambda\, \lambda^{-\theta} \tag{3.105}$$

$$\sim \lambda_p^{1-\theta} \tag{3.106}$$

for p sufficiently large, the p-th eigenvalue scales as

$$\lambda_p \sim p^{\frac{1}{\theta-1}} . \tag{3.107}$$

Substituting in the equation for the average-case error, together with the scaling forms of $t(p)$ and $\gamma(p)$, we find that

$$E_g(p) \sim \sum_\rho \frac{\mathbb{E}[|w_\rho^\star|^2]}{\lambda_\rho}\left(\frac{1}{\lambda_\rho} + p^{-\frac{1}{1-\theta}}\right)^{-2} \tag{3.108}$$

$$\sim \sum_\rho \frac{\mathbb{E}[|w_\rho^\star|^2]}{\lambda_\rho}\left(\frac{1}{\lambda_\rho} + \frac{1}{\lambda_p}\right)^{-2} \tag{3.109}$$

$$\sim \sum_{\rho\leq p} \mathbb{E}[|w_\rho^\star|^2]\frac{\lambda_p^2}{\lambda_\rho} + \sum_{\rho>p} \mathbb{E}[|w_\rho^\star|^2]\frac{\lambda_\rho^2}{\lambda_\rho} \tag{3.110}$$

$$\sim \sum_{\rho>p} \mathbb{E}[|w_\rho^\star|^2]\lambda_\rho \tag{3.111}$$

$$\sim \sum_{\rho>p} |\langle f^\star, \phi_\rho\rangle|^2 \,. \tag{3.112}$$

Thus, the main contributions to the error are the squared moduli of the target function's projections on the student kernel eigenfunctions with rank bigger than the number of training samples p. Indeed, the projections of a shift-invariant function on the first eigenfunctions of the convolutional Laplacian are bigger with respect to the ones of the vanilla Laplacian, resulting in a lower magnitude of the error. These results show that the great performance of over-parametrized convolutional architectures on shift-invariant tasks can be due to the fact that their first kernel eigenfunctions are better aligned with the input data distribution. In the next chapter, we verify this hypothesis through numerical experiments on handwritten digits. Furthermore, even if in a very simplified scenario and in the over-parametrized regime only, this analysis justifies recent observation of spectral bias in deep learning models [35, 36], which could be a key point to understand the generalization capabilities of neural networks.

Chapter 4

Numerical Experiments

In this chapter, we test our theoretical results, and we extend them to real data with numerical experiments. In the first section, we compute the learning curves of kernel regression in the teacher-student setting, and we compare them with those of very wide neural networks. In the second section, we estimate the learning curves for kernel regression of handwritten digits computing numerically the projections on the eigenfunctions of the kernels defined in the previous chapter, and we compare with the actual ones. Finally, we analyze the learning curves of our kernels in a classification setting.

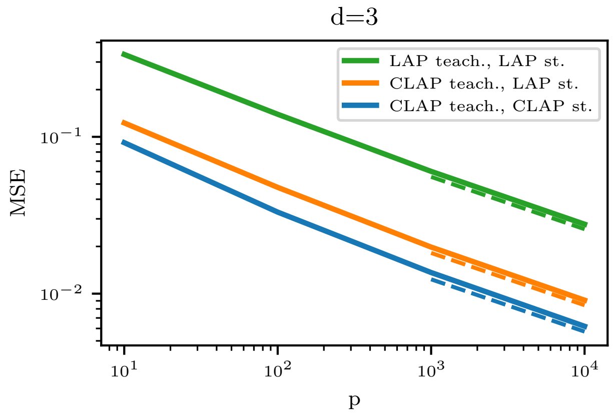
4.1 Synthetic Data

Kernel Regression

We first check if our theory predicts the asymptotic behaviors and the prefactors of the learning curves in the teacher-student setting. We generate $p + p_{\text{test}}$ points sampled from a Gaussian random field with zero mean and covariance given by the teacher kernel. Then, we train a kernel regression model using the student kernel with p points, and we evaluate the performance computing the mean squared error on the p_{test} points that compose the testing set. We use different values of p comprised between 10 and 10000, and $p_{\text{test}} = 1000$. In figure 4.1, we report the learning curves averaged over 100 executions for the various dimensions and combinations of the student and teacher kernels. All the curves follow the predicted power law, and the ratios between them are in good agreement with the predicted dimensional-dependent prefactors, as shown in table 4.1.

Wide Neural Networks

Having established that our theory works with kernels, we test if it approximates well the behaviour of actual neural networks trained in the NTK regime, which is



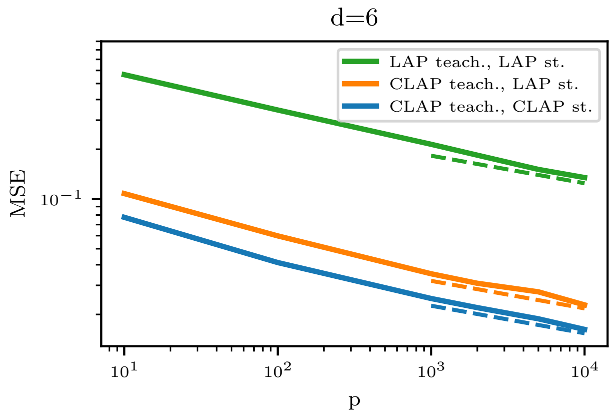


Figure 4.1: Learning curves (mean squared error vs dataset size p) for kernel regression in the teacher-student setting. Data points are sampled uniformly on a d-dimensional hypercube of unitary side. The target function is a Gaussian random field with zero mean and covariance given by the teacher kernel. Regression is performed with the student kernel. Each curve is averaged over 100 runs. The dashed lines are the predicted power laws with exponent $^1/_d$.

	$d = 3$	$d = 6$
$\mathrm{MSE}_{LL}/\mathrm{MSE}_{CL}$	th. 3.00, exp. 3.04	th. 6.00, exp. 6.02
$\mathrm{MSE}_{CL}/\mathrm{MSE}_{CC}$	th. 1.44, exp. 1.45	th. 1.37, exp. 1.40

Table 4.1: Ratios of the mean squared errors for kernel regression in the teacher-student setting. Notation: MSE_{CL} is the generalization error obtained using a convolutional Laplacian teacher (C) with a Laplacian student (L). The first number corresponds to the theoretical prediction (th.), the second one is obtained from the numerical experiments (exp.) averaging the errors of the last decade.

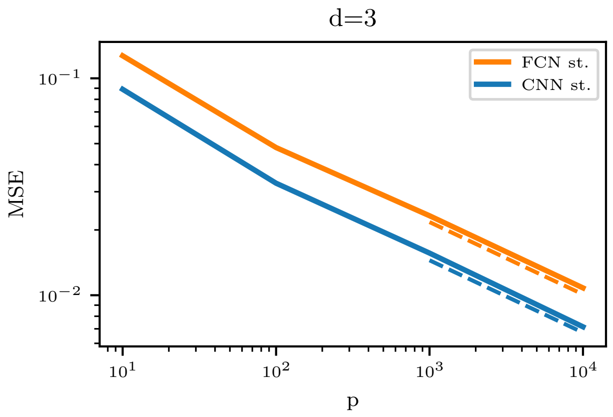


Figure 4.2: Learning curves for NTK regression in the teacher-student setting. Data points are sampled uniformly on a d-dimensional hypercube of unitary side. The target function is a Gaussian random field with zero mean and covariance given by the convolutional Laplacian kernel. Regression is performed by a one hidden layer fully connected network and a minimal convolutional network in the NTK limit. Each curve is averaged over 100 runs. The dashed lines are the predicted power laws with exponent $1/d$.

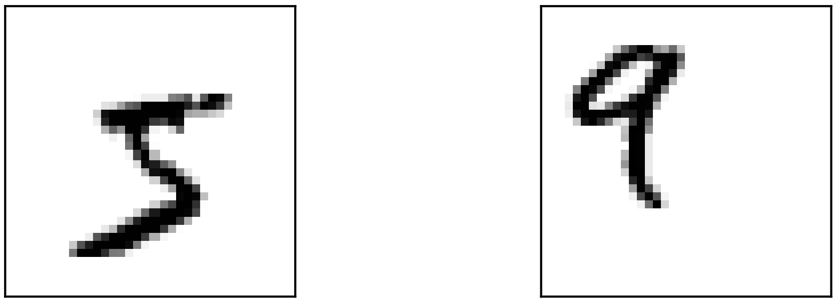


Figure 4.3: Two examples of handwritten digits from the decentred MNIST dataset.

also called lazy regime for very wide but still finite-size networks. In particular, we use the networks we defined in equations (3.1) and (3.10) with $h = 20000$, and we force them to work in the lazy regime with a trick first developed by Chizat et al. in [17]. Instead of the network function $f(\boldsymbol{x};\boldsymbol{\theta})$, we train a model defined as

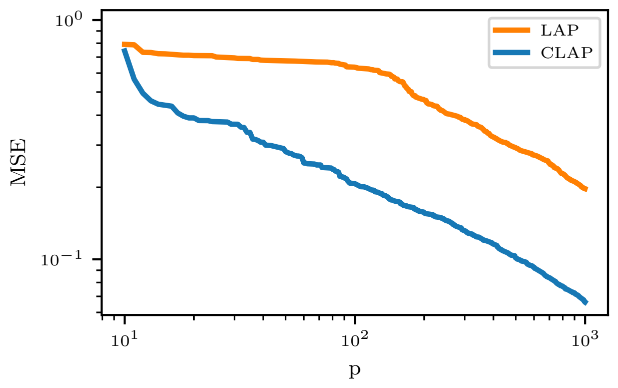
$$F(\boldsymbol{x};\boldsymbol{\theta},\boldsymbol{\theta}_0) = \alpha(f(\boldsymbol{x};\boldsymbol{\theta}) - f(\boldsymbol{x};\boldsymbol{\theta}_0)) \tag{4.1}$$

with $\boldsymbol{\theta}_0$ the parameters at initialization. The network function at initialization is not affected by the optimization algorithm, and for big values of α the weights barely need to move to change significantly the predictor $F(\boldsymbol{x};\boldsymbol{\theta},\boldsymbol{\theta}_0)$, keeping the training dynamics in the lazy regime. In our simulations we proceed similarly as before. We generate the training and the test set sampling a Gaussian random field with the teacher kernel, and we train our model $F(\boldsymbol{x};\boldsymbol{\theta},\boldsymbol{\theta}_0)$ using a mean squared loss divided by α^2, optimizing by SGD with batch size 20 until the error on the test set stabilizes or starts increasing. Figure 4.2 shows the averaged learning curves obtained using a shift-invariant teacher kernel in $d = 3$ and the two different architectures as students. Compared with the learning curves in figure 4.1, they have the same power-law behaviour and magnitude, confirming the validity of the adopted approximations.

4.2 Real Data

Kernel Regression

In this section, we switch from Gaussian to real data. Specifically, we consider the problem of inferring if a handwritten digit is an even or odd number. We use the MNIST database, which consists of 70000, 28×28, greyscale images of digits. Since the images are centered, we pad with 4 empty pixels on all the sides, and we apply a random translation. In figure 4.4 we show two examples of images taken from our modified database. The training set consists in p tuples $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^{p}$, where $\boldsymbol{x}_i$ is



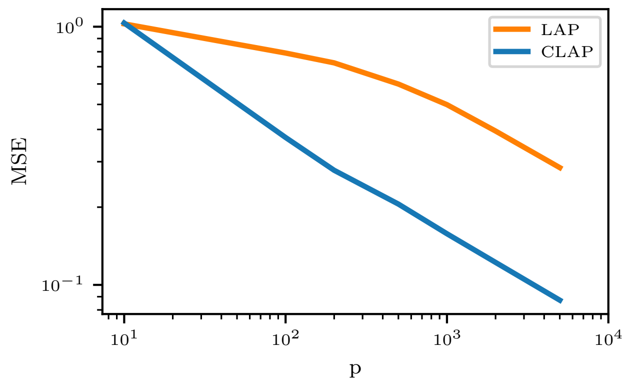


Figure 4.4: Learning curves for kernel regression on the translated MNIST dataset using the Laplacian and convolutional Laplacian kernels. Labels are assigned according to the parity of the number. The curves in plot (a) are obtained computing the projections of the labels on the eigenfunctions of the kernel, which are computed numerically. The curves in plot (b) are obtained averaging 100 runs of kernel regression.

an image from the database, and $y_i = f^\star(\boldsymbol{x}_i) = \pm 1$ depending on the parity of the number. We try to infer the label $f(\boldsymbol{x})$ of a digit $\boldsymbol{x}$ which does not belong to the training set doing regression with the student kernel K_S.
First, we attempt to get the learning curves using equation (3.112). Consider a dataset built as the previous one with $\tilde{p} > p$ elements. The eigenfunctions with respect to the discrete measure which associates equal mass to all the $\tilde{p}$ points are the $\tilde{p}$-dimensional eigenvectors $\tilde{\boldsymbol{\phi}}_\rho$ of the $\tilde{p} \times \tilde{p}$ Gram matrix $\mathbb{K}_S$

$$\frac{1}{\tilde{p}} \mathbb{K}_S \tilde{\boldsymbol{\phi}}_\rho = \tilde{\lambda}_\rho \tilde{\boldsymbol{\phi}}_\rho. \tag{4.2}$$

Therefore, we can approximate equation (3.112) with

$$E_g(p) \sim \sum_{\rho=p}^{\tilde{p}} \left(\boldsymbol{y}^\top \tilde{\boldsymbol{\phi}}_\rho \right)^2 \tag{4.3}$$

where $\boldsymbol{y}$ is the vector with all the labels $(\boldsymbol{y})_i = y_i$. Figure X shows the plots of the learning curves computed with $\tilde{p} = 5000$, compared to the actual learning curves averaged over 100 runs. Except for the region in which $\tilde{p} \simeq p$ where finite size effects are present, the curves follow the same power law. As for Gaussian data, the learning curves obtained with the two student kernels decay with the same exponent for large p, and the performance with the convolutional kernel is higher. Indeed, the first eigenfunctions of this kernel are better aligned with our dataset than those of the standard Laplacian (figure 4.5). We point out that the fitted value for the exponent β is 0.37, which is notably larger than $\frac{1}{d} = \frac{1}{36 \times 36} = 7,71 \times 10^{-4}$. Surprisingly, the curse of dimensionality seems to be absent. However, the distribution of MNIST images is highly anisotropic, and therefore the effective dimension of the manifold in which the data live is much lower than the number of pixels of the pictures, explaining why β is significantly larger than $\frac{1}{d}$ [32].

Kernel Classification

Finally, we study the previous problem of inferring the parity of a handwritten number in a classification setting using soft-margin kernel SVMs. This algorithm seeks for an estimator $f(\boldsymbol{x})$ which is linear in the feature space $\boldsymbol{\phi}(\boldsymbol{x})$ related to the student kernel by $K_S(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{\phi}(\boldsymbol{x})^\top \boldsymbol{\phi}(\boldsymbol{x}')$, and whose sign predicts the label $f^\star(\boldsymbol{x})$ correctly. Moreover, for every training point $\boldsymbol{x}_i$ we impose a margin $f^\star(\boldsymbol{x}_i) f(\boldsymbol{x}_i) > 1 - \xi_i$, and we penalize large values of ξ_i. Mathematically, we write our estimator as

$$f(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{\phi}(\boldsymbol{x}) + b \tag{4.4}$$

and we want to solve

$$\min_{\boldsymbol{w}, b, \boldsymbol{\xi}} \left(\boldsymbol{w}^\top \boldsymbol{w} + C \sum_i \xi_i \right) \tag{4.5}$$

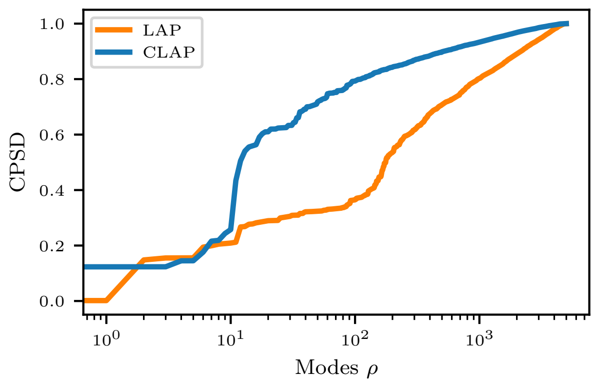


Figure 4.5: Cumulative power spectral density (CPSD) of the projections of the translated MNIST dataset on the eigenfunctions of the Laplacian and convolutional Laplacian kernels. Faster increments correspond to larger projections on the first eigenfunctions.

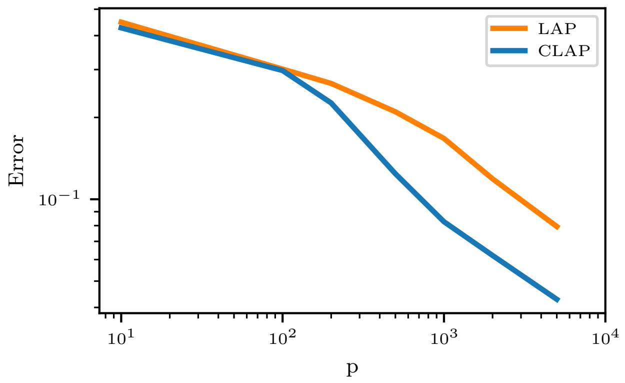


Figure 4.6: Learning curves for kernel classification on the translated MNIST dataset using soft-margin SVMs with the Laplacian and convolutional Laplacian kernels. Labels are assigned according to the parity of the number. Each curve is averaged over 100 runs.

subject to the constraints

$$f^\star(\boldsymbol{x}_i)\left(\boldsymbol{w}^\top\boldsymbol{\phi}(\boldsymbol{x}_i)+b\right) > 1-\xi_i, \quad \xi_i \geq 0. \tag{4.6}$$

The constant C controls the trade-off between minimizing the error on the training set and maximizing the margins (the limit $C \to \infty$ corresponds to the usual hard-margin SVMs). This problem is equivalent to the following dual Lagrangian problem

$$\max_{\boldsymbol{\alpha}}\left(\sum_{i=1}^{p}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{p}\alpha_i\alpha_j f^\star(\boldsymbol{x}_i)f^\star(\boldsymbol{x}_j)K_S(\boldsymbol{x}_i,\boldsymbol{x}_j)\right) \tag{4.7}$$

subject to the constraints

$$\sum_{i=1}^{p}\alpha_i f^\star(\boldsymbol{x}_i) = 0 \tag{4.8}$$

$$0 \leq \alpha_i \leq C. \tag{4.9}$$

Let $\boldsymbol{\alpha}_*$ be the maximum,

$$f(\boldsymbol{x}) = \sum_{i=1}^{p}\alpha_{*i}f^\star(\boldsymbol{x}_i)K_S(\boldsymbol{x}_i,\boldsymbol{x}) + b_* \tag{4.10}$$

with

$$b_* = f(\boldsymbol{x}_i) - \sum_{j=1}^{p}\alpha_{*j}f^\star(\boldsymbol{x}_j)K_S(\boldsymbol{x}_i,\boldsymbol{x}_j) \tag{4.11}$$

for any i such that $\alpha_i < C$.
In figure 4.6, we report the learning curves obtained using $C = 10^4$ and 1000 test points to compute the generalization error, which we define as the fraction of misclassifications. Again, the convolutional Laplacian kernel performs best, but the exponents are the same in the two cases. In this setting, we measure $\beta = 0.42$, similarly to [32].

Chapter 5

Conclusion

In this thesis, we studied the performance of the NTK induced by convolutional architectures, motivated by the remarkable empirical results obtained by these kernels on image datasets. We introduced a minimal model of a one-dimensional CNN with a single hidden convolutional layer, average pooling, and a final dense layer. We derived the correspondent NTK analytically, and we found a simple relation with the NTK of a one hidden layer fully connected network. The convolutional NTK is the average of the vanilla NTK over translations of one of the two arguments. In particular, this convolutional NTK inherits the invariance to translations of the convolutional architecture. Then, we studied the learning curves of these two kernels in the presence of a translational invariant task. We used a teacher-student setting for kernels in which data are modeled as Gaussian random fields, and we computed the learning curves using an approximate theory of the typical generalization performance based on the RKHS spectrum. We proved that the exponents β of the learning curves of these two kernels behave as $O(1/d)$, and thus suffer from the curse of dimensionality. This fact suggests that kernel methods cannot exploit translational invariance to reduce the dimensionality of the problem, similarly to what is observed in [33] for other simple symmetries. However, we showed that the eigenfunctions of the convolutional NTK are better aligned to the translational-invariant Gaussian random field with respect to the vanilla NTK's ones. Since the generalization error is governed by the power of the target function on the first kernel eigenfunctions, the error of the convolutional NTK is lowered by a dimensional-dependent prefactor. The learning curves obtained numerically are consistent with our predictions. Further numerical experiments on MNIST suggest that our analysis is also valid beyond Gaussian random fields. Indeed, both in a regression and a classification setting, the learning curves display the same exponent β in the limit of a large number of training points p, and the convolutional NTK achieves the best performance. Furthermore, computing the data projections on the numerical kernel eigenfunctions results in reasonable estimates of the real learning curves. A potential improvement to this analysis is to consider convolutional networks with smaller filters to implement locality, or with increased depth, to add hierarchy to our model. It would also be interesting to develop similar simple models of other modern architectures. Indeed, the computation of the RKHS spectrum of the corresponding induced kernels can shred light on the inductive biases of these models and its effect on sample complexity. Beyond the NTK regime, there is empirical evidence that infinitely wide neural networks are able to compress irrelevant dimensions [37]. However, a rigorous understanding of what symmetries in data can guarantee such an improvement is still missing. The minimal model of a convolutional network that we introduced, together with simple data models, can prove useful to analytical analyses aimed at answering these questions and explaining the improved performance of convolutional architectures compared to fully connected ones.

Bibliography

[1] CD Freeman and J Bruna. "Topology and geometry of deep rectified network optimization landscapes Int". In: *Conf. on Learning Representations.* 2017.

[2] Elad Hoffer, Itay Hubara, and Daniel Soudry. "Train longer, generalize better: closing the generalization gap in large batch training of neural networks". In: *Advances in Neural Information Processing Systems.* 2017, pp. 1731–1741.

[3] Levent Sagun, Léon Bottou, and Yann LeCun. "Singularity of the hessian in deep learning". In: *arXiv preprint arXiv:1611.07476* (2016).

[4] Levent Sagun et al. "Empirical analysis of the hessian of over-parametrized neural networks". In: *arXiv preprint arXiv:1706.04454* (2017).

[5] Marco Baity-Jesi et al. "Comparing dynamics: Deep neural networks versus glassy systems". In: *International Conference on Machine Learning.* 2018, pp. 314–323.

[6] Behnam Neyshabur et al. "Geometry of optimization and implicit regularization in deep learning". In: *arXiv preprint arXiv:1705.03071* (2017).

[7] Behnam Neyshabur et al. "Towards understanding the role of over-parametrization in generalization of neural networks". In: *arXiv preprint arXiv:1805.12076* (2018).

[8] Madhu S Advani, Andrew M Saxe, and Haim Sompolinsky. "High-dimensional dynamics of generalization error in neural networks". In: *Neural Networks* (2020).

[9] Stefano Spigler et al. "A jamming transition from under-to over-parametrization affects loss landscape and generalization". In: *arXiv preprint arXiv:1810.09665* (2018).

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

[11] Simon S Du et al. "Gradient descent provably optimizes over-parameterized neural networks". In: *arXiv preprint arXiv:1810.02054* (2018).

[12] Jaehoon Lee et al. “Wide neural networks of any depth evolve as linear models under gradient descent”. In: *Advances in neural information processing systems*. 2019, pp. 8572–8583.

[13] Song Mei, Andrea Montanari, and Phan-Minh Nguyen. “A mean field view of the landscape of two-layer neural networks”. In: *Proceedings of the National Academy of Sciences* 115.33 (2018), E7665–E7671.

[14] Grant M Rotskoff and Eric Vanden-Eijnden. “Trainability and accuracy of neural networks: An interacting particle system approach”. In: *arXiv preprint arXiv:1805.00915* (2018).

[15] Song Mei, Theodor Misiakiewicz, and Andrea Montanari. “Mean-field theory of two-layers neural networks: dimension-free bounds and kernel limit”. In: *arXiv preprint arXiv:1902.06015* (2019).

[16] Phan-Minh Nguyen and Huy Tuan Pham. “A rigorous framework for the mean field limit of multilayer neural networks”. In: *arXiv preprint arXiv:2001.11443* (2020).

[17] Lenaic Chizat, Edouard Oyallon, and Francis Bach. “On lazy training in differentiable programming”. In: *Advances in Neural Information Processing Systems*. 2019, pp. 2937–2947.

[18] Sanjeev Arora et al. “On exact computation with an infinitely wide neural net”. In: *Advances in Neural Information Processing Systems*. 2019, pp. 8141–8150.

[19] Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

[20] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. `http://www.deeplearningbook.org`. MIT Press, 2016.

[21] Bernhard Schölkopf and A.J. Smola. *Smola, A.: Learning with Kernels - Support Vector Machines, Regularization, Optimization and Beyond. MIT Press, Cambridge, MA*. Vol. 98. Jan. 2001.

[22] Ulrike von Luxburg and Olivier Bousquet. “Distance-based classification with Lipschitz functions”. In: *Journal of Machine Learning Research* 5.Jun (2004), pp. 669–695.

[23] George Cybenko. “Approximation by superpositions of a sigmoidal function”. In: *Mathematics of control, signals and systems* 2.4 (1989), pp. 303–314.

[24] Kurt Hornik, Maxwell Stinchcombe, Halbert White, et al. “Multilayer feedforward networks are universal approximators.” In: *Neural networks* 2.5 (1989), pp. 359–366.

[25] Ronen Eldan and Ohad Shamir. “The power of depth for feedforward neural networks”. In: *Conference on learning theory*. 2016, pp. 907–940.

[26] Yann LeCun et al. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[27] James Mercer. “Xvi. functions of positive and negative type, and their connection the theory of integral equations”. In: *Philosophical transactions of the royal society of London. Series A, containing papers of a mathematical or physical character* 209.441-458 (1909), pp. 415–446.

[28] Blake Bordelon, Abdulkadir Canatar, and Cengiz Pehlevan. “Spectrum Dependent Learning Curves in Kernel Regression and Wide Neural Networks”. In: (2020). arXiv: 2002.02561 [cs.LG].

[29] Youngmin Cho and Lawrence K Saul. “Kernel methods for deep learning”. In: *Advances in neural information processing systems*. 2009, pp. 342–350.

[30] Peter Sollich. “Learning curves for Gaussian processes”. In: *Advances in neural information processing systems*. 1999, pp. 344–350.

[31] Peter Sollich. “Gaussian process regression with mismatched models”. In: *Advances in Neural Information Processing Systems*. 2002, pp. 519–526.

[32] Stefano Spigler, Mario Geiger, and Matthieu Wyart. “Asymptotic learning curves of kernel methods: empirical data vs Teacher-Student paradigm”. In: *arXiv preprint arXiv:1905.10843* (2019).

[33] Jonas Paccolat, Stefano Spigler, and Matthieu Wyart. “How isotropic kernels learn simple invariants”. In: *arXiv preprint arXiv:2006.09754* (2020).

[34] Amnon Geifman et al. “On the Similarity between the Laplace and Neural Tangent Kernels”. In: *arXiv preprint arXiv:2007.01580* (2020).

[35] Nasim Rahaman et al. “On the spectral bias of neural networks”. In: *International Conference on Machine Learning*. PMLR. 2019, pp. 5301–5310.

[36] Zhi-Qin John Xu, Yaoyu Zhang, and Yanyang Xiao. “Training behavior of deep neural network in frequency domain”. In: *International Conference on Neural Information Processing*. Springer. 2019, pp. 264–274.

[37] Jonas Paccolat et al. “Geometric compression of invariant manifolds in neural nets”. In: (2020). arXiv: 2007.11471 [cs.LG].