Contents

1 Introduction **2**

2 Neural Networks **3**

2.1 Notation 3

2.2 Training in the Parameter Space 5

3 Training in the Function Space **13**

3.1 Motivation 13

3.2 Functional Derivative 14

3.3 Reproducing Kernel Hilbert Space 22

3.4 Kernel Gradient Descent 25

3.5 Neural Tangent Kernel 31

4 Training in Low-Dimensional Subspaces **37**

4.1 Motivation 37

4.2 Singular Value Decomposition 38

4.3 Low-Dimensional Trajectory Hypothesis 40

4.4 Dynamic Linear Dimensionality Reduction 44

4.5 DLDR-based Training Algorithms 47

5 Numerical Experiments **50**

5.1 Verification on Various Architectures 51

5.2 Sampling Strategies 55

5.3 Algorithm Design 58

6 Conclusion **64**

A Appendix **65**

1 Introduction

Deep Neural Networks usually contain a tremendous amount of parameters, leading to a lot of computational effort needed to train these models. However, due to their construction through nested layers and the training via error backpropagation, there is likely a lot of dependency and redundancy within the parameters of neural networks. As a consequence, there is a chance that deep neural networks could be trained in low-dimensional subspaces. This circumstance led to the rise of the low-dimensional trajectory hypothesis in [1]:

> *"For a neural network with n parameters, the parameters' trajectory over training can be approximately covered by a d-dimensional space with $d \ll n$."*

If this hypothesis holds, it would provide a new understanding of neural networks, both from a theoretical and practical perspective. According to [1], this includes generalization capability, implicit regularization, efficient learning algorithms, and much more.

The goal of this thesis is to investigate the feasibility of training deep neural networks in low-dimensional subspaces both from a theoretical and practical perspective. For the theoretical investigation, we first study the training of neural networks in function spaces and introduce the Neural Tangent Kernel, which was first raised by Arthur Jacot, Franck Gabriel, and Clément Hongler in "Neural Tangent Kernel: Convergence and Generalization in Neural Networks" [8] from the year 2018. Based on the special properties of this kernel, we can theoretically motivate the existence of low-dimensional optimization trajectories in the infinite-width limit. In practical applications, however, the setting of infinite-width limit never applies. Therefore, the practical investigation is of great importance and will be conducted through various numerical experiments, indicating that the low-dimensional trajectory hypothesis could indeed hold. The practical exploration is predominantly based on the work of Tao Li, Lei Tan, Qinghua Tao, Yipeng Liu, and Xiaolin Huang in "Low Dimensional Trajectory Hypothesis is True: DNNs can be Trained in Tiny Subspaces" [1] from the year 2022. The topic is thus characterized by high topicality and relevance.

In the following, we start by introducing the necessary notations for neural networks and proceed by presenting several algorithms for the training of neural networks in parameter spaces. To circumvent the problem of non-convexity we then analyze the kernel gradient descent algorithm, which is a theoretical approach to training neural networks in function spaces. Naturally, this leads to the definition of the Neural Tangent Kernel, which is used to theoretically motivate the existence of low-dimensional optimization trajectories. After these theoretical considerations, we move on to study the practical feasibility of training deep neural networks in low-dimensional subspaces. We introduce a method to extract appropriate subspaces and adapt widely known training techniques to these subspaces. Finally, these algorithms are extensively tested in various numerical experiments. The code is released on github[1]. Also, detailed results of all numerical experiments are published[2].

[1] https://github.com/04janik/Master_Thesis/Experiments

[2] https://wandb.ai/04janik

2 Neural Networks

This section introduces the basic concepts of neural networks and provides the necessary notation for further sections. We start with the formal definition of neural networks and explain the process of training afterward. Finally, we will introduce several training methods.

2.1 Notation

To start things off, we consider the simple model of a neuron.

Definition 2.1. *Let $n_x \in \mathbb{N}$, $\mathsf{w} \in \mathbb{R}^{n_x}$, $\mathsf{b} \in \mathbb{R}$ and $\sigma : \mathbb{R} \to \mathbb{R}$. A neuron is defined as*

$$\nu : \mathbb{R}^{n_x} \to \mathbb{R} : x \mapsto \sigma\Big(\mathsf{w}^\mathsf{T} x + \mathsf{b}\Big).$$

In this notion σ is called the activation function, w the weights, and b the bias of ν.

As a generalization of this concept, one can use multiple neurons in parallel to form a layer.

Definition 2.2. *Let $n_x, n_y \in \mathbb{N}$, $W = [\mathsf{w}_1, \dots, \mathsf{w}_{n_y}] \in \mathbb{R}^{n_x \times n_y}$, $\mathsf{b} \in \mathbb{R}^{n_y}$, and $\sigma : \mathbb{R} \to \mathbb{R}$. Given neurons ν_i with weights w_i, bias b_i, and activation function σ for $i = 1, \dots, n_y$, we define a layer as*

$$f : \mathbb{R}^{n_x} \to \mathbb{R}^{n_y} : x \mapsto \begin{bmatrix} \nu_1(x) \\ \vdots \\ \nu_{n_y}(x) \end{bmatrix} = \sigma\Big(W^\mathsf{T} x + \mathsf{b}\Big),$$

where the activation function σ is applied element-wise.

To provide a better understanding, these basic concepts can be visualized as graphs.

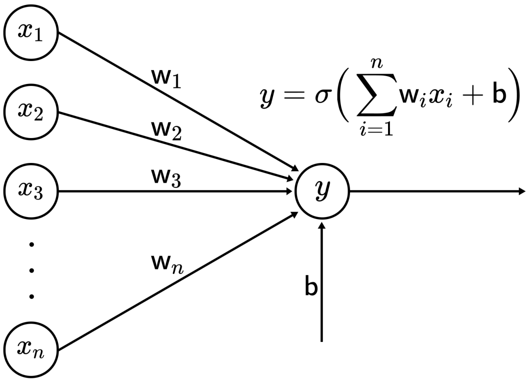


Figure 1: Illustration of a neuron as graph.

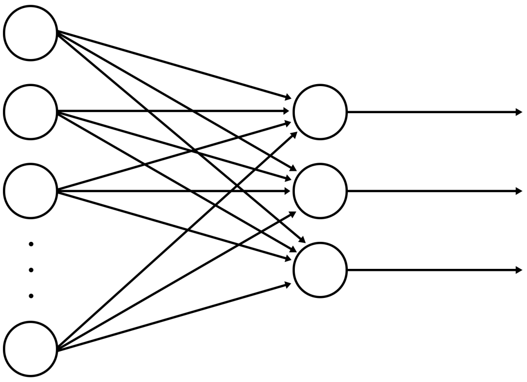


Figure 2: Illustration of a layer as graph.

In the following, we will no longer distinguish between weights and biases and simply refer to them as parameters. Logically one can use the output of a layer as input for another layer. Iteratively doing so will construct specific functions known as neural networks. The following definition formalizes this idea.

Definition 2.3. *Let $l \in \mathbb{N}$ and $n_0, \dots, n_l \in \mathbb{N}$. A neural network is defined as a function*

$$f : \mathbb{R}^{n_0} \to \mathbb{R}^{n_l} : x \mapsto f^{(l)} \circ f^{(l-1)} \circ \cdots \circ f^{(1)}(x),$$

where $f^{(i)}$ for $i = 1, \dots, l$ is a layer with input dimension n_{i-1} and output dimension n_i. Networks with a certain level of complexity, say $l > 2$, are referred to as deep neural networks.

Just like neurons and layers, neural networks can also be visualized as graphs.

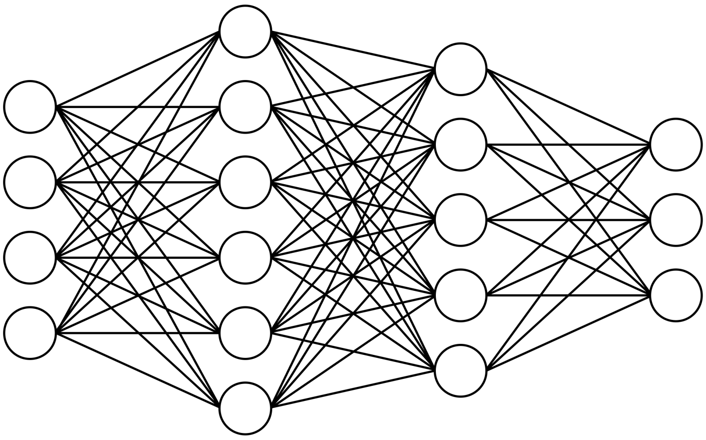


Figure 3: Illustration of a neural network as graph.

Neural networks find massive adoption in machine learning as they can model arbitrary complex functions. For example, one could envision functions that predict the weather at a specific place and time or functions that determine if an image contains a cat or a dog. Clearly, there is no simple mathematical formula to describe such problems.

A common practice to learn those functions is, to fix the number of layers, the number of neurons per layer as well as the activation functions, to choose some initial parameters, and then iteratively adjust the parameters in order to approximate the unknown target function. In the notion of definition 2.3, one can determine the total number of parameters in f as

$$n = \sum_{i=0}^{l-1} (n_i + 1) \cdot n_{i+1}.$$

Denoting these parameters as a vector $w \in \mathbb{R}^n$ enables us to define a function

$$\tilde{f} : \mathbb{R}^{n_0} \times \mathbb{R}^n \to \mathbb{R}^{n_l} : (x, w) \mapsto \tilde{f}(x, w),$$

such that for fixed $w_1, w_2 \in \mathbb{R}^n$ the functions $\tilde{f}(x, w_1)$ and $\tilde{f}(x, w_2)$ are neural networks matching in their number of layers, number of neurons as well as activation functions and differing only in their choice of parameters. One calls $\tilde{f}(x, w_1)$ and $\tilde{f}(x, w_2)$ realizations of the neural network architecture $\tilde{f}$.

In the following, the terms neural network and neural network architecture will be used synonymously if the meaning is clear in the context. Unless otherwise specified, $n_0 \in \mathbb{N}$ will denote the dimension of the input space, $n_l \in \mathbb{N}$ will denote the dimension of the output space and $n \in \mathbb{N}$ will denote the number of adjustable parameters.

2.2 Training in the Parameter Space

Training a neural network refers to finding the optimal parameters, given a fixed neural network architecture $f : \mathbb{R}^{n_0} \times \mathbb{R}^n \to \mathbb{R}^{n_l}$. In other words, finding a function $f_w \in \mathcal{F}|_f$, where

$$\mathcal{F}|_f := \Big\{ f_w : \mathbb{R}^{n_0} \to \mathbb{R}^{n_l} : x \mapsto f(x, w) \mid w \in \mathbb{R}^n \Big\}$$

denotes the set of all neural networks with the given architecture. In order to define a sense of optimality, we need a mechanism for measuring the quality of network outputs. This is done by a loss function

$$\ell : \mathbb{R}^{n_l} \times \mathbb{R}^{n_l} \to \mathbb{R}_+ : \big(f(x, w), y\big) \mapsto \ell\big(f(x, w), y\big),$$

that should be chosen in such a way, that for some input $x \in \mathbb{R}^{n_0}$ with target output $y \in \mathbb{R}^{n_l}$ it associates some cost with the error between the prediction $f(x, w)$, and the true label y.

Under the assumption that, in reality, there exists a probability distribution $\mathcal{D}$ on $\mathbb{R}^{n_0} \times \mathbb{R}^{n_l}$, describing the correlation between inputs $x \in \mathbb{R}^{n_0}$ and target values $y \in \mathbb{R}^{n_l}$, we call a parameter vector $w_* \in \mathbb{R}^n$ to be optimal, if

$$\forall w \in \mathbb{R}^n : R(w_*) \leqslant R(w),$$

where $R(w)$ denotes the so-called risk or generalization error

$$R : \mathbb{R}^n \to \mathbb{R}_+ : w \mapsto \mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}} \Big[\ell\big(f(x, w), y\big) \Big].$$

For later use, we will denote the marginal distribution of the inputs as $\mathcal{D}_x$. At this point, it should be noted, that in general, the underlying probability distribution $\mathcal{D}$ is unknown. Hence we can not evaluate the expectation in the previous expression.

For this reason, the training of neural networks requires a set of $m \in \mathbb{N}$ labeled samples

$$\Big\{ (x_i, y_i) \in \mathbb{R}^{n_0} \times \mathbb{R}^{n_l} \mid i = 1, \dots, m \Big\},$$

which is called training data. Instead of working with the generalization error, the training data enables us to minimize the empirical error function

$$E : \mathbb{R}^n \to \mathbb{R}_+ : w \mapsto \frac{1}{m} \sum_{i=1}^{m} \ell\big(f(x_i, w), y_i\big) + \lambda \|w\|_2^2,$$

where $\lambda \geqslant 0$ is some regularization parameter used to control the complexity of w.

Intuitively, minimization of the empirical error should lead to a minimization of the generalization error as well. Currently, a lot of research is being done on how these measures relate to each other. Nevertheless, this thesis will focus on how to minimize the empirical error given labeled training data.

The problem of finding some parameter vector w that minimizes the empirical error function is usually hard to solve as the error may have highly nonlinear dependencies on the parameters. This is why, in practice, one often aims to compare several local minima and to settle for one that provides a sufficiently small error, regardless of whether it is a global minimum or not.

Definition 2.4. *Let $f : \mathbb{R}^n \to \mathbb{R}$ be differentiable. The gradient of f is defined as*

$$\nabla f : \mathbb{R}^n \to \mathbb{R}^n : x = (x_1, \dots, x_n) \mapsto \left[\frac{\partial}{\partial x_1} f(x), \cdots, \frac{\partial}{\partial x_n} f(x) \right]^{\mathsf{T}}.$$

Due to the complexity of the error function, in general, there is no easy way to find an analytical solution to the problem $\nabla E(w) = 0$. Hence most of the techniques for error minimization are based on iterative approximation. One popular method is gradient descent, which was first raised by Cauchy in [2].

Algorithm 1 Gradient Descent

```
Input: Differentiable $f : \mathbb{R}^n \to \mathbb{R}$, $x_0 \in \mathbb{R}^n$, $\epsilon \geqslant 0$
Initialize $k \leftarrow 0$;
while $\|\nabla f(x_k)\|_2 > \epsilon$ do:
    Choose a step size $\eta_k > 0$;
    Set $x_{k+1} \leftarrow x_k - \eta_k \cdot \nabla f(x_k)$ and $k \leftarrow k + 1$;
end while
```

We will see, that in the case where f is bounded from below and ∇f is Lipschitz continuous, we can guarantee the convergence of gradient descent to a stationary point.

Definition 2.5. *Let $(\mathcal{X}, d_{\mathcal{X}})$ and $(\mathcal{Y}, d_{\mathcal{Y}})$ be two metric spaces. A function $f : \mathcal{X} \to \mathcal{Y}$ is called Lipschitz continuous, if there exists $L \geqslant 0$, such that*

$$\forall x, x' \in \mathcal{X} : d_{\mathcal{Y}}\big(f(x), f(x')\big) \leqslant L \cdot d_{\mathcal{X}}(x, x').$$

The smallest L, that suffices the previous condition is called the Lipschitz constant of f.

Casually speaking, the Lipschitz continuity of ∇f ensures that the gradient does not change heavily for two points that are close to each other.

Definition 2.6. *Let $f : \mathbb{R}^n \to \mathbb{R}$ be differentiable. A stationary point of f is an element of*

$$\big\{ x \in \mathbb{R}^n \mid \nabla f(x) = 0 \big\}.$$

There are three kinds of stationary points: saddle points, local extrema, and global extrema. We are especially interested in global minima as they provide the lowest possible error.

Definition 2.7. *Let $\mathcal{X}$ be a subset of some real vector space and $f : \mathcal{X} \to \mathbb{R}$ be bounded from below. We call $x_* \in \mathcal{X}$ a global minimizer of f, if*

$$\forall x \in \mathcal{X} : f(x_*) \leqslant f(x).$$

To prove convergence of the gradient descent algorithm for bounded f with Lipschitz continuous gradient, we need the following result which is formulated as lemma 1.2.3 in [3].

Lemma 2.8. *Let $f : \mathbb{R}^n \to \mathbb{R}$ be differentiable, such that the gradient ∇f is Lipschitz continuous with Lipschitz constant $L \geqslant 0$, then*

$$\forall x, x' \in \mathbb{R}^n : f(x') \leqslant f(x) + \big\langle \nabla f(x), x' - x \big\rangle + \frac{L}{2} \|x' - x\|_2^2.$$

Proof. Let $x, x' \in \mathbb{R}^n$, define $z_\lambda \coloneqq x + \lambda(x' - x)$ for $\lambda \in \mathbb{R}$ and consider the function

$$\psi : \mathbb{R} \to \mathbb{R} : \lambda \mapsto f(z_\lambda).$$

Since f is differentiable, we can apply the chain rule to derive

$$\psi' : \mathbb{R} \to \mathbb{R} : \lambda \mapsto (x' - x)^\mathsf{T} \cdot \nabla f(z_\lambda).$$

Integration of ψ' from $\lambda = 0$ to $\lambda = 1$ yields

$$\int_0^1 \big\langle \nabla f(z_\lambda), x' - x \big\rangle \, d\lambda = \int_0^1 \psi'(\lambda) \, d\lambda = \psi(1) - \psi(0) = f(x') - f(x).$$

This equality can be rewritten as

$$\begin{aligned} f(x') - f(x) &= \int_0^1 \big\langle \nabla f(z_\lambda), x' - x \big\rangle \, d\lambda \\ &= \int_0^1 \big\langle \nabla f(z_\lambda) - \nabla f(x) + \nabla f(x), x' - x \big\rangle \, d\lambda \\ &= \big\langle \nabla f(x), x' - x \big\rangle + \int_0^1 \big\langle \nabla f(z_\lambda) - \nabla f(x), x' - x \big\rangle \, d\lambda. \end{aligned}$$

Using the Cauchy-Schwarz inequality, we derive

$$\begin{aligned} \Big| \, f(x') - f(x) - \big\langle \nabla f(x), x' - x \big\rangle \, \Big| &= \Big| \, \int_0^1 \big\langle \nabla f(z_\lambda) - \nabla f(x), x' - x \big\rangle \, d\lambda \, \Big| \\ &\leqslant \int_0^1 \Big| \, \big\langle \nabla f(z_\lambda) - \nabla f(x), x' - x \big\rangle \, \Big| \, d\lambda \\ &\leqslant \int_0^1 \big\|\nabla f(z_\lambda) - \nabla f(x)\big\|_2 \cdot \big\|x' - x\big\|_2 \, d\lambda. \end{aligned}$$

Due to the Lipschitz continuity of ∇f, we can use the upper bound

$$\big\|\nabla f(z_\lambda) - \nabla f(x)\big\|_2 \leqslant L \cdot \big\|z_\lambda - x\big\|_2 = L \cdot \big\|\lambda(x' - x)\big\|_2 = L\lambda \cdot \big\|x' - x\big\|_2$$

for $\lambda \in [0, 1]$, to conclude

$$\Big| \, f(x') - f(x) - \big\langle \nabla f(x), x' - x \big\rangle \, \Big| \leqslant \int_0^1 L\lambda \cdot \big\|x' - x\big\|_2^2 \, d\lambda = \frac{L}{2} \|x' - x\|_2^2.$$

Rearranging the inequality finally yields

$$f(x') \leqslant f(x) + \big\langle \nabla f(x), x' - x \big\rangle + \frac{L}{2} \|x' - x\|_2^2.$$

This completes the proof since x and x' were chosen arbitrarily. □

Theorem 2.9. *Let $f : \mathbb{R}^n \to \mathbb{R}$ be differentiable with global minimizer $x_* \in \mathbb{R}^n$, such that the gradient ∇f is Lipschitz continuous with Lipschitz constant $L > 0$. Then, gradient descent with $\epsilon = 0$ and constant step size $\eta_k = 1/L$ produces a sequence $(x_k)_{k=0}^{\infty}$ in $\mathbb{R}^n$, such that*

$$\forall m \in \mathbb{N} : \min_{0 \leqslant k \leqslant m} \big\|\nabla f(x_k)\big\|_2^2 \leqslant \frac{2L\big(f(x_0) - f(x_*)\big)}{m+1}.$$

Proof. Let $k \in \mathbb{N}$ be arbitrary. Since f is differentiable with Lipschitz continuous gradient, we can apply lemma 2.8 to derive

$$f(x_{k+1}) \leqslant f(x_k) + \big\langle \nabla f(x_k), x_{k+1} - x_k \big\rangle + \frac{L}{2}\big\|x_{k+1} - x_k\big\|_2^2.$$

By the definition of gradient descent, we can plug in $x_{k+1} - x_k = -1/L \cdot \nabla f(x_k)$ to obtain

$$\begin{aligned} f(x_{k+1}) &\leqslant f(x_k) + \big\langle \nabla f(x_k), -\frac{1}{L}\nabla f(x_k) \big\rangle + \frac{L}{2}\big\|\frac{1}{L}\nabla f(x_k)\big\|_2^2 \\ &= f(x_k) - \frac{1}{L}\big\|\nabla f(x_k)\big\|_2^2 + \frac{1}{2L}\big\|\nabla f(x_k)\big\|_2^2 \\ &= f(x_k) - \frac{1}{2L}\big\|\nabla f(x_k)\big\|_2^2. \end{aligned}$$

Hence the gradient descent algorithm with constant step size $\eta_k = 1/L$ guarantees to make progress, unless $\nabla f(x_k) \neq 0$. The previous expression is equivalent to

$$\big\|\nabla f(x_k)\big\|_2^2 \leqslant 2L\big(f(x_k) - f(x_{k+1})\big).$$

Summing up both sides from $k = 0$ to some $m \in \mathbb{N}$ and taking the average, results in

$$\frac{1}{m+1}\sum_{k=0}^{m} \big\|\nabla f(x_k)\big\|_2^2 \leqslant \frac{2L}{m+1}\sum_{k=0}^{m} f(x_k) - f(x_{k+1}) = \frac{2L}{m+1}\big(f(x_0) - f(x_{m+1})\big).$$

Using $f(x_*) \leqslant f(x_k)$ for every $k \in \mathbb{N}_0$, we conclude

$$\min_{0 \leqslant k \leqslant m} \big\|\nabla f(x_k)\big\|_2^2 \leqslant \frac{1}{m+1}\sum_{k=0}^{m} \big\|\nabla f(x_k)\big\|_2^2 \leqslant \frac{2L}{m+1}\big(f(x_0) - f(x_*)\big).$$

This completes the proof since m was chosen arbitrarily. □

The theorem can be found in equation (1.2.15) at the beginning of section 1.2.3 of [3]. Note that, in practice, it is inefficient to use a constant step size of $1/L$. Nevertheless, it is useful for theoretical analysis to guarantee convergence.

Theorem 2.9 states that in the limit as $n \to \infty$, gradient descent converges to a stationary point $x \in \mathbb{R}^n$ with $\nabla f(x) = 0$. Since we have seen in the proof, that $f(x_k)$ is monotonically decreasing in k, gradient descent converges to the next stationary point in direction of descent, which is either a local minimum, a global minimum, or a saddle point.

Next, we will define a special class of functions, having the nice property that every stationary point is a global minimum, and therefore gradient descent is an excellent tool for minimization.

Definition 2.10. *Let $\mathcal{X}$ be a subset of some real vector space. $\mathcal{X}$ is said to be convex, if*

$$\forall x, x' \in \mathcal{X} : \forall \lambda \in [0,1] : x + \lambda(x' - x) \in \mathcal{X}.$$

Casually speaking, a set is convex if it contains the connection line between any two points in the set. Based on the notion of convex sets we can define convex functions.

Definition 2.11. *Let $\mathcal{X}$ be a convex subset of some real vector space. A function $f : \mathcal{X} \to \mathbb{R}$ is said to be convex, if*

$$\forall x, x' \in \mathcal{X} : \forall \lambda \in [0,1] : f\big(x + \lambda(x' - x)\big) \leqslant f(x) + \lambda\big(f(x') - f(x)\big).$$

For later use, we denote the following property of convex functions.

Lemma 2.12. *Let $\mathcal{X}$ be a convex subset of some real vector space. For convex functions $f : \mathcal{X} \to \mathbb{R}$ and $g : \mathcal{X} \to \mathbb{R}$ the sum $f + g$ is also convex.*

Proof. Let $x, x' \in \mathcal{X}$, and $\lambda \in [0,1]$ be arbitrary, then

$$\begin{aligned}\big(f + g\big)\big(x + \lambda(x' - x)\big) &= f\big(x + \lambda(x' - x)\big) + g\big(x + \lambda(x' - x)\big)\\ &\leqslant f(x) + \lambda\big(f(x') - f(x)\big) + g(x) + \lambda\big(g(x') - g(x)\big)\\ &= (f + g)(x) + \lambda\big((f + g)(x') - (f + g)(x)\big).\end{aligned}$$

This shows the convexity of $f + g$. □

Functions, that are convex and continuously differentiable, have the nice property that every stationary point is a global minimum. Mathematically, this can be formulated as follows.

Lemma 2.13. *Let $f : \mathbb{R}^n \to \mathbb{R}$ be convex and continuously differentiable and $x \in \mathbb{R}^n$, then*

$$\nabla f(x) = 0 \;\Leftrightarrow\; \forall x' \in \mathbb{R}^n : f(x) \leqslant f(x').$$

Proof. Let $x \in \mathbb{R}^n$ with $\nabla f(x) = 0$. For arbitrary $x' \in \mathbb{R}^n$, we define the function

$$\psi : \mathbb{R} \to \mathbb{R} : \lambda \mapsto f(x) + \lambda\big(f(x') - f(x)\big) - f\big(x + \lambda(x' - x)\big),$$

which is non-negative on the interval $[0,1]$ by the convexity of f. Denoting the derivative as

$$\psi' : \mathbb{R} \to \mathbb{R} : \lambda \mapsto f(x') - f(x) - (x' - x)^{\mathsf{T}} \cdot \nabla f\big(x + \lambda(x' - x)\big)$$

and using the non-negativity of ψ together with $\psi(0) = 0$, we observe that

$$\psi'(0) = f(x') - f(x) - (x' - x)^{\mathsf{T}} \cdot \nabla f(x) \geqslant 0.$$

Since x was chosen such that $\nabla f(x) = 0$, we can rearrange terms to conclude

$$\forall x' \in \mathbb{R}^n : f(x) \leqslant f(x').$$

This completes the proof since the backward direction is trivial. □

Thus, for lower bounded functions $f : \mathbb{R}^n \to \mathbb{R}$ that are convex and differentiable with Lipschitz continuous gradient ∇f, we can guarantee the convergence of gradient descent to a global minimum due to theorem 2.9. In general, this does not apply to the error function used for training neural networks, but we will introduce a theoretical approach to prevent this problem in section 3. Next, we will provide further iterative optimization methods.

Another approach to finding local minima of some differentiable function $f : \mathbb{R}^n \to \mathbb{R}$, is to solve the equation $\nabla f(x) = 0$. For this purpose, one can use the widely known Newton method, which is a technique to solve the problem of finding a root of a differentiable function. Hence, to apply this technique for minimization of f, we have to assume that f is twice differentiable. To formally introduce the Newton method, we require the following definition.

Definition 2.14. *Let $f : \mathbb{R}^n \to \mathbb{R}$ be twice differentiable. The Hessian of f is defined as*

$$H_f : \mathbb{R}^n \to \mathbb{R}^{n \times n} : x = (x_1, \dots, x_n) \mapsto \begin{bmatrix} \frac{\partial}{\partial x_1 \partial x_1} f(x) & \cdots & \frac{\partial}{\partial x_1 \partial x_n} f(x) \\ \vdots & \ddots & \vdots \\ \frac{\partial}{\partial x_n \partial x_1} f(x) & \cdots & \frac{\partial}{\partial x_n \partial x_n} f(x) \end{bmatrix}.$$

The basic idea of Newton's method consists of approximating

$$\nabla f(x + \Delta x) \approx \nabla f(x) + H_f(x) \Delta x \overset{!}{=} 0.$$

Based on this approximation, iterative updates $\Delta x = -H_f^{-1}(x) \nabla f(x)$ provide the rule

$$x_{k+1} \leftarrow x_k - H_f^{-1}(x_k) \nabla f(x_k).$$

According to section 1.2.4 of [3], the technique's convergence rate is quadratic, such that convergence in a neighborhood of strict local minima is very fast. However, this method is cursed with two serious drawbacks. Firstly, the Hessian matrices $H(x_k)$ have to be invertible for every $k \in \mathbb{N}$. Secondly, the technique can diverge. To avoid possible divergence, we can slightly modify the update rule and introduce the Newton method as follows.

Algorithm 2 Newton's Method

```
Input: Twice differentiable $f : \mathbb{R}^n \to \mathbb{R}$ with invertible Hessian matrices, $x_0 \in \mathbb{R}^n$, $\epsilon \geqslant 0$
Initialize $k \leftarrow 0$;
while $\|\nabla f(x_k)\|_2 > \epsilon$ do:
    Choose a step size $\eta_k > 0$;
    Set $x_{k+1} \leftarrow x_k - \eta_k H_f^{-1}(x_k) \nabla f(x_k)$ and $k \leftarrow k + 1$;
end while
```

If n is large, the Hessian matrix computation in Newton's method can become computationally very expensive. Since we are only in need of the inverse Hessian matrices $H_f^{-1}(x_k)$, one approach to reduce the computational effort consists of approximating these inverse Hessians. A widely known method that pursues this approach, is the following from [4].

Algorithm 3 Broyden-Fletcher-Goldfarb-Shanno (BFGS)

```
Input: Twice differentiable $f : \mathbb{R}^n \to \mathbb{R}$, $x_0 \in \mathbb{R}^n$, $\epsilon \geqslant 0$
Initialize $k \leftarrow 0$ and $B_0 \leftarrow I_n$;
while $\|\nabla f(x_k)\|_2 > \epsilon$ do:
    Choose a step size $\eta_k > 0$;
    Set $x_{k+1} \leftarrow x_k - \eta_k B_k \nabla f(x_k)$;
    Let $s_k = x_{k+1} - x_k$ and $y_k = \nabla f(x_{k+1}) - \nabla f(x_k)$;
    Let $\rho_k = (y_k^\top s_k)^{-1}$ and $V_k = I_n - \rho_k y_k s_k^\top$;
    Set $B_{k+1} \leftarrow V_k^\top B_k V_k + \rho_k s_k s_k^\top$ and $k \leftarrow k + 1$;
end while
```

With the initialization $B_0 = I_n$, the method starts with a gradient descent update and then iteratively minimizes the objective function by inverse Hessian matrix approximations B_k.

So far, we have seen three different approaches to iteratively minimize a twice differentiable function $f : \mathbb{R}^n \to \mathbb{R}$. Concerning our objective of minimizing the empirical error

$$E : \mathbb{R}^n \to \mathbb{R}_+ : w \mapsto \frac{1}{m} \sum_{i=1}^{m} \ell\big(f(x_i, w), y_i\big) + \lambda \|w\|_2^2,$$

however, we must note that, due to the large number of parameters in deep neural networks in combination with the massive amount of training data needed, the proposed optimization algorithms are impractical. Still, the approach of neural network training is to iteratively optimize the parameters via some update rule of the form

$$w_{k+1} \leftarrow w_k - \eta_k \cdot d_k,$$

where $d_k \in \mathbb{R}^n$ denotes a direction of descent. Note that gradient descent, Newton's method as well as BFGS pursue exactly this approach. For the purpose of minimizing E however, we need to find a computationally cheaper method to determine the descent directions.

Stochastic gradient descent is a variation of gradient descent, that aims to approximate the gradient of the objective function instead of computing it exactly. The basic idea is to compute the gradient on a random fraction of the data set and use this as an estimation of the gradient on the whole data set. In each iteration $k \in \mathbb{N}$, one randomly draws a subset $\mathcal{I}_k \subset \{1, \dots, m\}$ of size $|\mathcal{I}_k| = b \in \mathbb{N}$ and uses $\nabla E(w_k, \mathcal{I}_k)$, defined via

$$E(w_k, \mathcal{I}_k) \coloneqq \frac{1}{b} \sum_{i \in \mathcal{I}_k} \ell\big(f(x_i, w_k), y_i\big) + \lambda \|w_k\|_2^2,$$

as an approximation of $\nabla E(w_k)$. Although only a small amount of data is used for one single parameter update, during training most of the data will be used to fit the model due to the large number of iterative updates. Formally stochastic gradient descent proceeds as follows.

Algorithm 4 Stochastic Gradient Descent (SGD)

```
Input: Differentiable $E : \mathbb{R}^n \to \mathbb{R}$, batch size $b \in \mathbb{N}$, number of epochs $N \in \mathbb{N}$, $w_0 \in \mathbb{R}^n$
for $k = 0, \dots, \frac{Nm}{b}$ do:
    Draw a random subset $\mathcal{I}_k \subset \{1, \dots, m\}$ of size $|\mathcal{I}_k| = b$;
    Choose a step size $\eta_k > 0$;
    Set $w_{k+1} \leftarrow w_k - \eta_k \cdot \nabla E(w_k, \mathcal{I}_k)$;
end for
```

Here, the number of epochs describes the number of complete passes through the training data set. Thus, on average, each training sample will contribute N times to the optimization process. In comparison to this, the classical gradient descent method uses the data of one epoch per iteration. A theoretical convergence analysis of stochastic gradient descent can be found in [5] but would exceed the scope of this thesis.

As of now, we have not discussed the choice of step sizes η_k in any of the four optimization methods proposed so far. In machine learning scenarios, it is a common practice to choose a constant step size $\eta > 0$ for all parameter updates. Of course, there are better choices of step sizes rather than this naive approach. However, these often come with additional computational effort. A widely known technique for the choice of sophisticated step sizes is the backtracking line search, which was first raised in [7].

Algorithm 5 Backtracking Line Search

```
Input: Differentiable $f : \mathbb{R}^n \to \mathbb{R}$, $x \in \mathbb{R}^n$, $\beta \in (0,1)$, $c \in (0,1)$
Initialize $k \leftarrow 0$ and $\eta_0 \leftarrow 1$;
while $f\big(x + \eta_k \cdot \nabla f(x)\big) > f(x) - c \cdot \eta_k \cdot \nabla f(x)^{\mathsf{T}} \nabla f(x)$ do:
    Set $\eta_{k+1} \leftarrow \beta \cdot \eta_k$;
end while
```

This completes the necessary basics of notation and training of neural networks. Next, we will investigate certain approaches on how to improve the training of neural networks, in order to prevent the curse of non-convexity and reduce the computational time effort needed.

3 Training in the Function Space

This section studies the training of neural networks in function space, instead of the parameter space $\mathbb{R}^n$. We will start by motivating this approach and introduce the concept of functional derivatives afterward. Based on the concept of reproducing kernel Hilbert spaces, we can introduce the kernel gradient descent algorithm, which is a generalization of gradient descent for functionals. Finally, this algorithm leads to the definition of a special kernel, the neural tangent kernel, that is of great importance for the further theory. The results here are mainly based on the work of Arthur Jacot, Franck Gabriel, and Clément Hongler in [8].

3.1 Motivation

As seen in section 2.2, the training of neural networks via parameter optimization is usually accomplished by minimizing the empirical error

$$E : \mathbb{R}^n \to \mathbb{R}_+ : w \mapsto \frac{1}{m} \sum_{i=1}^{m} \ell\big(f(x_i, w), y_i\big) + \lambda \|w\|_2^2$$

on labeled training data $\{(x_i, y_i)\}_{i=1}^m \subset \mathbb{R}^{n_0} \times \mathbb{R}^{n_l}$. Due to the non-convexity of the error function E, it is in general hard to find a global minimum. However, for theoretical analysis, this problem can be prevented by moving from the parameter space $\mathbb{R}^n$ to the function space

$$\mathcal{F} := \Big\{ f : \mathbb{R}^{n_0} \to \mathbb{R}^{n_l} \Big\}.$$

That is, we are no longer optimizing the specific parameters of the unknown target function, but rather the function itself. This approach requires the definition of a cost functional

$$C : \mathcal{F} \to \mathbb{R}_+ : f \mapsto \frac{1}{m} \sum_{i=1}^{m} \ell\big(f(x_i), y_i\big),$$

associating some cost to each element of the function space $\mathcal{F}$. The interpretation of ℓ as a convex function on $\mathcal{F}$ provides a convex cost C since by lemma 2.12 the sum of convex functions is again convex. This is a huge advantage in comparison to the parameter optimization approach, where the convexity of ℓ on $\mathcal{F}$ does not translate to a convex error E on $\mathbb{R}^n$, due to the composition of ℓ with f.

In the following, we will introduce the kernel gradient descent algorithm, which is a generalization of gradient descent from the parameter space $\mathbb{R}^n$ to the function space $\mathcal{F}$. Since we can ensure convexity of the cost C in the function space, convergence of kernel gradient descent to a global minimum will be guaranteed.

Note that this approach is great for theoretical analysis due to the advantage of convexity. In practice, however, we are still in need of the explicit parameters $w \in \mathbb{R}^n$, that describe the objective function. Nevertheless, we can use the analysis of kernel gradient descent to get a better understanding of gradient descent in the parameter space. That is why we aim to investigate the relationship between the two optimization approaches.

3.2 Functional Derivative

To start things off, we need the definition of functionals, mapping each element of the function space $\mathcal{F}$ to a real value. The cost C is an example of such a functional.

Definition 3.1. *A functional on the vector space $\mathcal{F}$ is defined as a mapping*

$$\mu : \mathcal{F} \to \mathbb{R} : f \mapsto \mu[f].$$

Minimizing functionals via the same approach in gradient descent requires a translation of the well-known concepts of differentiability and gradients from the parameter space $\mathbb{R}^n$ to the function space $\mathcal{F}$. The following definition is based on Appendix A of [9].

Definition 3.2. *Let $\phi, f_0 \in \mathcal{F}$. A functional $\mu : \mathcal{F} \to \mathbb{R}$ is said to be differentiable at the point f_0 in direction ϕ, if*

$$D_\phi \mu[f_0] \coloneqq \lim_{\epsilon \to 0} \frac{1}{\epsilon} \Big(\mu[f_0 + \epsilon\phi] - \mu[f_0] \Big) = \frac{\partial}{\partial \epsilon} \Big[\mu[f_0 + \epsilon\phi] \Big]_{\epsilon=0}$$

exists. If the limit exists for every $\phi \in \mathcal{F}$ we define the functional derivative of μ at f_0 as

$$\partial_f \mu|_{f_0} : \mathcal{F} \to \mathbb{R} : \phi \mapsto D_\phi \mu[f_0]$$

and call μ differentiable at f_0.

This is similar to the definition of directional derivatives in $\mathbb{R}^{n_0}$. Likewise, we can motivate the functional gradient in $\mathcal{F}$ based on the properties of the classical gradient in $\mathbb{R}^{n_0}$. To clarify this, we let $f \in \mathcal{F}$ and $x, v \in \mathbb{R}^{n_0}$, such that $\|v\|_2 = 1$ and consider the directional derivative of f in x along the direction v, which is given by

$$D_v f(x) \coloneqq \lim_{\epsilon \to 0} \frac{1}{\epsilon} \Big(f(x + \epsilon v) - f(x) \Big).$$

By Taylor's Theorem, we have for any $\epsilon > 0$, that

$$f(x + \epsilon v) = f(x) + \nabla f(x)^\mathsf{T} \big((x + \epsilon v) - x \big) + \mathcal{O}(\epsilon^2),$$

which can be rewritten as

$$f(x + \epsilon v) - f(x) = \epsilon \big\langle \nabla f(x), v \big\rangle + \mathcal{O}(\epsilon^2).$$

Dividing both sides by ϵ and taking the limit $\epsilon \to 0$ yields

$$D_v f(x) = \lim_{\epsilon \to 0} \frac{1}{\epsilon} \Big(f(x + \epsilon v) - f(x) \Big) = \big\langle \nabla f(x), v \big\rangle.$$

This gives us an approach how to defining the functional gradient in $\mathcal{F}$. Roughly speaking, we could translate this concept and think of an element $\nabla \mu|_{f_0} \in \mathcal{F}$ as the functional gradient at $f_0 \in \mathcal{F}$ of some differentiable $\mu : \mathcal{F} \to \mathbb{R}$ if it satisfies

$$\forall \phi \in \mathcal{F} : D_\phi \mu[f_0] = \big\langle \nabla \mu|_{f_0}, \phi \big\rangle_\mathcal{F},$$

for a reasonable inner product $\langle \cdot, \cdot \rangle_\mathcal{F}$. It will turn out, that this is completely possible. We will start with the definition of an inner product and show the existence of functional gradients afterward. The following lemma is based on [8].

Lemma 3.3. *Given a fixed probability distribution $\mathcal{D}_x$ on the input space $\mathbb{R}^{n_0}$, one can equip the function space $\mathcal{F}$ with the seminorm*

$$\|\cdot\|_{\mathcal{D}_x} : \mathcal{F} \to \mathbb{R} : f \mapsto \sqrt{\langle f, f\rangle_{\mathcal{D}_x}}$$

in terms of the symmetric positive semidefinite bilinear form

$$\langle \cdot, \cdot \rangle_{\mathcal{D}_x} : \mathcal{F} \times \mathcal{F} \to \mathbb{R} : (f, g) \mapsto \langle f, g\rangle_{\mathcal{D}_x} := \underset{x \sim \mathcal{D}_x}{\mathbb{E}} \Big[f(x)^\mathsf{T} g(x) \Big].$$

Proof. Let $f_1, f_2, f, g \in \mathcal{F}$ and $\lambda \in \mathbb{R}$. $\langle \cdot, \cdot \rangle_{\mathcal{D}_x}$ is indeed symmetric and bilinear since it suffices

(i) $\langle f_1 + f_2, g\rangle_{\mathcal{D}_x} = \langle f_1, g\rangle_{\mathcal{D}_x} + \langle f_2, g\rangle_{\mathcal{D}_x}$:

$$\begin{aligned}\langle f_1 + f_2, g\rangle_{\mathcal{D}_x} &= \underset{x \sim \mathcal{D}_x}{\mathbb{E}} \Big[\big(f_1(x) + f_2(x)\big)^\mathsf{T} g(x) \Big] \\ &= \underset{x \sim \mathcal{D}_x}{\mathbb{E}} \Big[f_1(x)^\mathsf{T} g(x) + f_2(x)^\mathsf{T} g(x) \Big] \\ &= \underset{x \sim \mathcal{D}_x}{\mathbb{E}} \Big[f_1(x)^\mathsf{T} g(x) \Big] + \underset{x \sim \mathcal{D}_x}{\mathbb{E}} \Big[f_2(x)^\mathsf{T} g(x) \Big] \\ &= \langle f_1, g\rangle_{\mathcal{D}_x} + \langle f_2, g\rangle_{\mathcal{D}_x},\end{aligned}$$

(ii) $\langle \lambda f, g\rangle_{\mathcal{D}_x} = \lambda \langle f, g\rangle_{\mathcal{D}_x}$:

$$\langle \lambda f, g\rangle_{\mathcal{D}_x} = \underset{x \sim \mathcal{D}_x}{\mathbb{E}} \Big[\lambda f(x)^\mathsf{T} g(x) \Big] = \lambda \cdot \underset{x \sim \mathcal{D}_x}{\mathbb{E}} \Big[f(x)^\mathsf{T} g(x) \Big] = \lambda \langle f, g\rangle_{\mathcal{D}_x},$$

(iii) $\langle f, g\rangle_{\mathcal{D}_x} = \langle g, f\rangle_{\mathcal{D}_x}$:

$$\langle f, g\rangle_{\mathcal{D}_x} = \underset{x \sim \mathcal{D}_x}{\mathbb{E}} \Big[f(x)^\mathsf{T} g(x) \Big] = \underset{x \sim \mathcal{D}_x}{\mathbb{E}} \Big[g(x)^\mathsf{T} f(x) \Big] = \langle g, f\rangle_{\mathcal{D}_x}.$$

Furthermore, $\langle \cdot, \cdot \rangle_{\mathcal{D}_x}$ is positive semidefinite, since for any $f \in \mathcal{F}$ it holds, that

$$\langle f, f\rangle_{\mathcal{D}_x} = \underset{x \sim \mathcal{D}_x}{\mathbb{E}} \Big[f(x)^\mathsf{T} f(x) \Big] = \underset{x \sim \mathcal{D}_x}{\mathbb{E}} \Big[\sum_{i=1}^{n_l} f_i(x)^2 \Big] \geqslant 0.$$

Thus $\|\cdot\|_{\mathcal{D}_x} = \sqrt{\langle \cdot, \cdot \rangle_{\mathcal{D}_x}}$ is a seminorm on the function space $\mathcal{F}$. □

Note that $\langle \cdot, \cdot \rangle_{\mathcal{D}_x}$ is not strictly positive definite. For example, in the case where $\mathcal{D}_x$ is a discrete distribution, any $f \in \mathcal{F}$ with $f(\operatorname{supp}(\mathcal{D}_x)) = 0$, satisfies $\langle f, f\rangle_{\mathcal{D}_x} = 0$. Since f could take arbitrary values for $x \notin \operatorname{supp}(\mathcal{D}_x)$, we can not conclude strict positive definiteness. However, in our setting, we only care about data points with a positive probability measure in terms of $\mathcal{D}_x$. That is why we can define an equivalence relation on $\mathcal{F}$, such that

$$\forall f, g \in \mathcal{F} \;:\; f \sim g \;\Leftrightarrow\; \langle f, \cdot \rangle_{\mathcal{D}_x} = \langle g, \cdot \rangle_{\mathcal{D}_x}.$$

Hence, two elements are equivalent in terms of $\sim$, if and only if they are equal on the data. Casually speaking, the restriction of $\langle \cdot, \cdot \rangle_{\mathcal{D}_x}$ to $\mathcal{F}/\sim$ is strictly positive definite, so

$$\langle \cdot, \cdot \rangle_{\mathcal{F}/\sim} : F/\sim \times F/\sim \to \mathbb{R} : \big([f], [g]\big) \mapsto \langle f, g\rangle_{\mathcal{D}_x}$$

is an inner product. We can prove the completeness of $\mathcal{F}$ with respect to $\|\cdot\|_{\mathcal{D}_x}$ and therefore completeness of $\mathcal{F}/\sim$ with respect to $\|\cdot\|_{\mathcal{F}/\sim}$, such that $\big(F/\sim, \langle \cdot, \cdot \rangle_{\mathcal{F}/\sim}\big)$ is a Hilbert space. This is done by the following corollary.

Corollary 3.4. *The seminorm $\|\cdot\|_{\mathcal{D}_x}$ induces the pseudometric*

$$d_{\mathcal{F}} : \mathcal{F} \times \mathcal{F} \to \mathbb{R}_+ : (f, g) \mapsto \big\|f - g\big\|_{\mathcal{D}_x},$$

such that $(\mathcal{F}, d_{\mathcal{F}})$ is a complete pseudometric space.

Proof. Let $f, g, h \in \mathcal{F}$. $d_{\mathcal{F}}$ is indeed a pseudometric since it suffices

(i) $d_{\mathcal{F}}(f, f) = 0$:

$$d_{\mathcal{F}}(f, f) = \big\|f - f\big\|_{\mathcal{D}_x} = 0,$$

(ii) $d_{\mathcal{F}}(f, g) = d_{\mathcal{F}}(g, f)$:

$$d_{\mathcal{F}}(f, g) = \big\|f - g\big\|_{\mathcal{D}_x} = \big\|g - f\big\|_{\mathcal{D}_x} = d_{\mathcal{F}}(g, f),$$

(iii) $d_{\mathcal{F}}(f, h) \leqslant d_{\mathcal{F}}(f, g) + d_{\mathcal{F}}(g, h)$:

$$\begin{aligned} d_{\mathcal{F}}(f, h) &= \big\|f - h\big\|_{\mathcal{D}_x} \\ &= \big\|f - g + g - h\big\|_{\mathcal{D}_x} \\ &\leqslant \big\|f - g\big\|_{\mathcal{D}_x} + \big\|g - h\big\|_{\mathcal{D}_x} = d_{\mathcal{F}}(f, g) + d_{\mathcal{F}}(g, h). \end{aligned}$$

The upper equations follow directly from the properties of the seminorm $\|\cdot\|_{\mathcal{D}_x}$.

It remains to show the completeness of $\mathcal{F}$, which arises from the fact, that every Cauchy sequence in $\mathbb{R}$ converges. To clarify this, we let $(f_n)_{n=1}^{\infty}$ denote a Cauchy sequence in $\mathcal{F}$, so

$$\forall \epsilon > 0 : \exists N \in \mathbb{N} : \forall m, n > N : d_{\mathcal{F}}(f_n, f_m)^2 < \epsilon. \tag{$*$}$$

Recalling the definition of $\|\cdot\|_{\mathcal{D}_x}$, we have

$$d_{\mathcal{F}}(f_n, f_m)^2 = \big\|f_n - f_m\big\|_{\mathcal{D}_x}^2 = \mathop{\mathbb{E}}_{x \sim \mathcal{D}_x}\Big[(f_n - f_m)(x)^{\mathsf{T}}(f_n - f_m)(x)\Big],$$

such that the inequality in $(*)$ becomes

$$d_{\mathcal{F}}(f_n, f_m)^2 = \mathop{\mathbb{E}}_{x \sim \mathcal{D}_x}\Big[\sum_{i=1}^{n_l}(f_n - f_m)_i(x)^2\Big] = \mathop{\mathbb{E}}_{x \sim \mathcal{D}_x}\Big[\sum_{i=1}^{n_l}\big((f_n)_i(x) - (f_m)_i(x)\big)^2\Big] < \epsilon.$$

Since all of the summands are non-negative, we observe for any $x \sim \mathcal{D}_x$, that

$$\forall \epsilon > 0 : \exists N \in \mathbb{N} : \forall m, n > N : \big|(f_n)_i(x) - (f_m)_i(x)\big| < \epsilon.$$

Hence $((f_n)_i(x))_{n=1}^{\infty}$ is a Cauchy sequence in $\mathbb{R}$ and therefore converges to a limit

$$\hat{f}_i(x) \coloneqq \lim_{n \to \infty}(f_n)_i(x) \in \mathbb{R}.$$

Equivalently this can be formulated as

$$\forall \epsilon > 0 : \exists N \in \mathbb{N} : \forall n > N : \big|(f_n)_i(x) - \hat{f}_i(x)\big| < \epsilon.$$

Since $\mathcal{F}$ contains all functions from $\mathbb{R}^{n_0}$ to $\mathbb{R}^{n_l}$, we can choose some $f_{lim} \in \mathcal{F}$, that satisfies

$$\forall x \sim \mathcal{D}_x : f_{lim}(x) = \left[\hat{f}_1(x), \ldots, \hat{f}_{n_l}(x)\right]^{\mathsf{T}}.$$

Again, using the definition of $\| \cdot \|_{\mathcal{D}_x}$, we observe for $n \in \mathbb{N}$, that

$$d_{\mathcal{F}}(f_n, f_{lim})^2 = \mathop{\mathbb{E}}_{x\sim\mathcal{D}_x}\left[\sum_{i=1}^{n_l}(f_n - f_{lim})_i(x)^2\right] = \mathop{\mathbb{E}}_{x\sim\mathcal{D}_x}\left[\sum_{i=1}^{n_l}\left((f_n)_i(x) - (f_{lim})_i(x)\right)^2\right].$$

Since for any $x \sim \mathcal{D}_x$ each sequence $((f_n)_i(x))_{n=1}^{\infty}$ converges, we finally conclude, that

$$\forall \epsilon > 0 : \exists N \in \mathbb{N} : \forall n > N : d_{\mathcal{F}}(f_n, f_{lim})^2 < \epsilon.$$

Hence the Cauchy sequence $(f_n)_{n=1}^{\infty}$ has a limit in $\mathcal{F}$, which makes $(\mathcal{F}, d_{\mathcal{F}})$ complete. □

Note that since $(\mathcal{F}, d_{\mathcal{F}})$ is only a pseudometric space and not a metric space, the limit f_{lim} constructed in the proof is not necessarily unique. Essentially any $f \in \mathcal{F}$, that suffices

$$\forall x \sim \mathcal{D}_x : f(x) = \left[\hat{f}_1(x), \ldots, \hat{f}_{n_l}(x)\right]^{\mathsf{T}}$$

represents a limit function of the Cauchy sequence. This is due to the limit being measured in terms of the seminorm $\| \cdot \|_{\mathcal{D}_x}$, which depends only on the data according to $\mathcal{D}_x$. Namely, it does not matter which values the limit function takes on points $x \notin \operatorname{supp}(\mathcal{D}_x)$. However, in the Hilbert space $\mathcal{F}/\sim$, the limit functions form an equivalence class, such that the limit in $\mathcal{F}/\sim$ is uniquely determined by its equivalence class.

Based on the notion of $\langle \cdot, \cdot \rangle_{\mathcal{D}_x}$, we can think of an element $\nabla\mu|_{f_0} \in \mathcal{F}$ as the functional gradient at $f_0 \in \mathcal{F}$ of some differentiable $\mu : \mathcal{F} \to \mathbb{R}$, if it satisfies

$$\forall \phi \in \mathcal{F} : D_\phi \mu[f_0] = \left\langle \nabla\mu|_{f_0}, \phi \right\rangle_{\mathcal{D}_x}.$$

Note that, due to the definition of $\langle \cdot, \cdot \rangle_{\mathcal{D}_x}$, the functional gradient is not unique and depends only on the support of $\mathcal{D}_x$. Essentially, we could think of any other element $d \in \mathcal{F}$, that is equivalent to $\nabla\mu|_{f_0}$ in terms of $\sim$, as the functional gradient as well. We will discuss and address this problem later on, but first, let us prove the existence of functional gradients.

We will see, that a functional gradient exists if $\partial_f \mu|_{f_0}$ is linear and continuous. For this, we denote the class of linear and continuous functionals on $\mathcal{F}$, namely the dual space of $\mathcal{F}$.

Definition 3.5. *Given a real vector space $\mathcal{F}$, we define the dual space*

$$\mathcal{F}^* := \left\{\mu : \mathcal{F} \to \mathbb{R} \mid \mu \text{ linear and continuous}\right\}.$$

To prove the existence of functional gradients, we will construct a surjective mapping

$$J : \mathcal{F} \to \mathcal{F}^* : d \mapsto \langle d, \cdot \rangle_{\mathcal{D}_x},$$

such that we can conclude the existence based on $\partial_f \mu|_{f_0} \in \mathcal{F}^*$. For the construction of J, we have to translate some well-known results from functional analysis to our setting.

Based on the completeness of $(\mathcal{F}, d_{\mathcal{F}})$, we can introduce the concept of orthogonal projections, which is necessary for our goal of establishing a mapping between $\mathcal{F}$ and its dual space $\mathcal{F}^*$. The following lemma is a translation of Theorem 3.3.1 in [10] to our case, where we have a symmetric positive semidefinite bilinear form $\langle \cdot, \cdot \rangle_{\mathcal{D}_x}$, instead of an inner product.

Lemma 3.6. *Let $\mathcal{A} \subseteq \mathcal{F}$ be non-empty, closed, and convex. For every $f \in \mathcal{F}$ there exists some $a^* \in \mathcal{A}$ such that*

$$\|f - a^*\|_{\mathcal{D}_x}^2 = \inf_{a \in \mathcal{A}} \|f - a\|_{\mathcal{D}_x}^2.$$

Each $a^* \in \mathcal{A}$ that suffices this property, is called an orthogonal projection of f on $\mathcal{A}$.

Proof. By definition of the infimum, there exists a sequence $(a_n)_{n=1}^{\infty}$ in $\mathcal{A}$, such that

$$\delta := \lim_{n \to \infty} \|f - a_n\|_{\mathcal{D}_x}^2 = \inf_{a \in \mathcal{A}} \|f - a\|_{\mathcal{D}_x}^2.$$

We show that $(a_n)_{n=1}^{\infty}$ is a Cauchy sequence. For this, we let $n, m \in \mathbb{N}$ and observe that

$$\begin{aligned} \|a_n - a_m\|_{\mathcal{D}_x}^2 &= \|(f - a_n) - (f - a_m)\|_{\mathcal{D}_x}^2 \\ &= \|f - a_n\|_{\mathcal{D}_x}^2 + \|f - a_m\|_{\mathcal{D}_x}^2 - 2\langle f - a_n, f - a_m \rangle_{\mathcal{D}_x}. \end{aligned}$$

Furthermore, we have

$$\|(f - a_n) + (f - a_m)\|_{\mathcal{D}_x}^2 = \|f - a_n\|_{\mathcal{D}_x}^2 + \|f - a_m\|_{\mathcal{D}_x}^2 + 2\langle f - a_n, f - a_m \rangle_{\mathcal{D}_x}.$$

Combining both expressions yields

$$\begin{aligned} \|a_n - a_m\|_{\mathcal{D}_x}^2 &= 2\|f - a_n\|_{\mathcal{D}_x}^2 + 2\|f - a_m\|_{\mathcal{D}_x}^2 - \|(f - a_n) + (f - a_m)\|_{\mathcal{D}_x}^2 \\ &= 2\|f - a_n\|_{\mathcal{D}_x}^2 + 2\|f - a_m\|_{\mathcal{D}_x}^2 - 4\|f - \frac{1}{2}(a_n + a_m)\|_{\mathcal{D}_x}^2. \end{aligned}$$

By the definition of δ, we observe that

$$\forall \epsilon > 0 : \exists N \in \mathbb{N} : \forall n > N : \|f - a_n\|_{\mathcal{D}_x}^2 < \delta + \epsilon.$$

Furthermore, since $\mathcal{A}$ is convex, $\frac{1}{2}(a_n + a_m) \in \mathcal{A}$ and we derive

$$\|f - \frac{1}{2}(a_n + a_m)\|_{\mathcal{D}_x}^2 \geqslant \inf_{a \in \mathcal{A}} \|f - a\|_{\mathcal{D}_x}^2 = \delta.$$

Using these two observations, we conclude that

$$\forall \epsilon > 0 : \exists N \in \mathbb{N} : \forall n > N : \|a_n - a_m\|_{\mathcal{D}_x}^2 \leqslant 2(\delta + \epsilon) + 2(\delta + \epsilon) - 4\delta = 4\epsilon$$

Thus, $(a_n)_{n=1}^{\infty}$ is a Cauchy sequence. Since $\mathcal{F}$ is complete and $\mathcal{A}$ is closed, $(a_n)_{n=1}^{\infty}$ converges to some $a^* \in \mathcal{A}$ with $\|f - a^*\|_{\mathcal{D}_x}^2 = \delta$, which proves the lemma. □

Note that since $\|\cdot\|_{\mathcal{D}_x}$ is only a seminorm and not a norm, the limit of $(a_n)_{n=1}^{\infty}$ is not unique. Essentially there can exist multiple projections from f onto the subset $\mathcal{A}$. Nevertheless, these orthogonal projections form an equivalence, such that we have uniqueness in $\mathcal{F}/\sim$.

Lemma 3.6 is a weakening of the Hilbert projection theorem stating that, given a Hilbert space, even a unique orthogonal projection exists. In our case, we lose the uniqueness, because $\langle \cdot, \cdot \rangle_{\mathcal{D}_x}$ is just positive semidefinite. However, this is just a side note since the existence of an orthogonal projection suffices for the further theory. Next, we infer a useful corollary, which is denoted as lemma 3.3.2 in [10].

Corollary 3.7. *Let $f \in \mathcal{F}$ and $\mathcal{A} \subseteq \mathcal{F}$ be a non-empty and closed subspace. For every $a \in \mathcal{A}$ it holds, that $\langle f - a^*, a\rangle_{\mathcal{D}_x} = 0$, where $a^* \in \mathcal{A}$ denotes an orthogonal projection of f on $\mathcal{A}$.*

Proof. Let $a \in \mathcal{A}$ and $\lambda \in \mathbb{R}$. Since $\mathcal{A}$ is a subspace, $a^* + \lambda a \in \mathcal{A}$ and we derive

$$\|f - (a^* + \lambda a)\|^2_{\mathcal{D}_x} \geqslant \inf_{a \in \mathcal{A}} \|f - a\|^2_{\mathcal{D}_x} = \|f - a^*\|^2_{\mathcal{D}_x}.$$

Rearranging the inequality yields

$$\|(f - a^*) - \lambda a\|^2_{\mathcal{D}_x} - \|f - a^*\|^2_{\mathcal{D}_x} = 2\big\langle f - a^*, \lambda a\big\rangle_{\mathcal{D}_x} + \|\lambda a\|^2_{\mathcal{D}_x} \geqslant 0,$$

giving rise to the definition of the non-negative function

$$\psi : \mathbb{R} \to \mathbb{R} : \lambda \mapsto 2\lambda\big\langle f - a^*, a\big\rangle_{\mathcal{D}_x} + \lambda^2\|a\|^2_{\mathcal{D}_x}.$$

Under the assumption that $\big\langle f - a^*, a\big\rangle_{\mathcal{D}_x} \neq 0$ and $\|a\|^2_{\mathcal{D}_x} \neq 0$, we derive

$$\psi\Big(- \frac{\langle f - a^*, a\rangle_{\mathcal{D}_x}}{\|a\|^2_{\mathcal{D}_x}}\Big) = -2 \cdot \frac{\big\langle f - a^*, a\big\rangle^2_{\mathcal{D}_x}}{\|a\|^2_{\mathcal{D}_x}} + \frac{\big\langle f - a^*, a\big\rangle^2_{\mathcal{D}_x}}{\|a\|^2_{\mathcal{D}_x}} = -\frac{\big\langle f - a^*, a\big\rangle^2_{\mathcal{D}_x}}{\|a\|^2_{\mathcal{D}_x}} < 0$$

and under the assumption that $\big\langle f - a^*, a\big\rangle_{\mathcal{D}_x} \neq 0$ and $\|a\|^2_{\mathcal{D}_x} = 0$, we derive

$$\psi\Big(- \big\langle f - a^*, a\big\rangle_{\mathcal{D}_x}\Big) = -2 \cdot \big\langle f - a^*, a\big\rangle^2_{\mathcal{D}_x} < 0,$$

which is a contradiction to $\psi(\lambda) \geqslant 0$ for every $\lambda \in \mathbb{R}$. Thus $\big\langle f - a^*, a\big\rangle_{\mathcal{D}_x} = 0$. □

Finally, the concept of orthogonal projections enables us to construct any $\mu \in \mathcal{F}^*$ based on a representative $d \in \mathcal{F}$. The next lemma is a translation of theorem 3.8.1 in [10] to our setting.

Theorem 3.8. *The map*

$$J : \mathcal{F} \to \mathcal{F}^* : d \mapsto \langle d, \cdot\rangle_{\mathcal{D}_x}$$

is linear and surjective.

Proof. The linearity follows directly from the bilinearity of $\langle \cdot, \cdot\rangle_{\mathcal{D}_x}$. To prove that J is indeed surjective, we let $\mu \in \mathcal{F}^*$ and need to find some $d \in \mathcal{F}$ such that $\mu = J(d)$.

Case $\mu = 0$: We can choose $d = 0$, with $\mu = 0 = J(d)$.

Case $\mu \neq 0$: Due to the linearity of μ, we can normalize and find some $e \in \mathcal{F}$ with $\mu[e] = 1$. Let $N \coloneqq \mu^{-1}(0)$ denote the kernel of μ, which is non-empty, closed, and convex since μ is linear and continuous. Thus, by Lemma 3.6, there exists some $p_e \in N$, such that p_e is an orthogonal projection of e on N. Define $f \coloneqq e - p_e$, then $f \notin N$ since

$$\mu[f] = \mu[e] - \mu[p_e] = \mu[e] - 0 = 1.$$

Thus, we can choose any $g \in \mathcal{F}$, define $h \coloneqq g - \mu[g]f$ and use the linearity of μ to derive

$$\mu[h] = \mu\big[g - \mu[g]f\big] = \mu[g] - \mu[g]\mu[f] = \mu[g] - \mu[g] \cdot 1 = 0.$$

This implies $h \in N$, so that $\langle f, h\rangle_{\mathcal{D}_x} = \langle e - p_e, h\rangle_{\mathcal{D}_x} = 0$ by corollary 3.7 and we conclude

$$\langle f, g\rangle_{\mathcal{D}_x} = \langle f, h\rangle_{\mathcal{D}_x} + \langle f, \mu[g]f\rangle_{\mathcal{D}_x} = 0 + \mu[g] \cdot \langle f, f\rangle_{\mathcal{D}_x} = \mu[g] \cdot \|f\|^2_{\mathcal{D}_x}.$$

Now we can assume that $\|f\|^2_{\mathcal{D}_x} \neq 0$. Otherwise, we would have $\langle f, g\rangle_{\mathcal{D}_x} = 0$ for every $g \in \mathcal{F}$ and therefore $f = 0$ which is a contradiction to $f \notin N$. Rearranging terms results in

$$\mu[g] = \left\langle f/\|f\|^2_{\mathcal{D}_x}, g\right\rangle_{\mathcal{D}_x} = J\left(f/\|f\|^2_{\mathcal{D}_x}\right)[g].$$

Hence we have found $d = f/\|f\|^2_{\mathcal{D}_x}$ with $\mu = J(d)$. This shows surjectivity. □

Thus, for each functional $\mu \in \mathcal{F}^*$, there exists $d \in \mathcal{F}$ such that $\mu = \langle d, \cdot\rangle_{\mathcal{D}_x}$. In other words

$$\mathcal{F}^* = \left\{\mu : \mathcal{F} \to \mathbb{R} : f \mapsto \langle d, f\rangle_{\mathcal{D}_x} \mid d \in \mathcal{F}\right\} \cong F/\sim.$$

Hence we can conclude the existence of functional gradients, as long as the functional derivative $\partial_f \mu|_{f_0}$ is an element of $\mathcal{F}^*$. In other words, $\partial_f \mu|_{f_0}$ is linear and continuous. To guarantee this, for the rest of the thesis we make the restriction, that the marginal distribution $\mathcal{D}_x$ on the input space is given by the empirical distribution on a finite subset of $\mathbb{R}^{n_0}$. This means there exist $m \in \mathbb{N}$ and $x_1, \ldots, x_m \in \mathbb{R}^{n_0}$, such that the distribution of inputs is given by

$$\mathcal{D}_x = \frac{1}{m}\sum_{i=1}^{m} \delta_{x_i},$$

where δ_{x_i} denotes the Dirac measure in x_i. Under this assumption, $\langle\cdot,\cdot\rangle_{\mathcal{D}_x}$ depends only on the values of f at a finite data set. This implies the following result from [8].

Lemma 3.9. *Let $\mu : \mathcal{F} \to \mathbb{R}$ be differentiable in $f_0 \in \mathcal{F}$, then we have $\partial_f \mu|_{f_0} \in \mathcal{F}^*$. In other words, the functional derivative $\partial_f \mu|_{f_0}$ is linear and continuous.*

Proof. The proof is based on the linear operator

$$T : \mathcal{F}/\sim \to \mathbb{R} : [\phi] \mapsto \partial_f \mu|_{f_0}[\phi].$$

To prove that T is indeed linear, we need to show that $\partial_f \mu|_{f_0}$ is linear. Based on the linearity of the total derivative, we observe for any $\phi_1, \phi_2 \in \mathcal{F}$, that

$$\begin{aligned}\partial_f \mu|_{f_0}[\phi_1 + \phi_2] &= \frac{\partial}{\partial\epsilon}\Big[\mu[f_0 + \epsilon\phi_1 + \epsilon\phi_2]\Big]_{\epsilon=0} \\ &= \frac{\partial}{\partial\epsilon}\Big[\mu[f_0 + \epsilon\phi_1]\Big]_{\epsilon=0} + \frac{\partial}{\partial\epsilon}\Big[\mu[f_0 + \epsilon\phi_2]\Big]_{\epsilon=0} \\ &= \partial_f \mu|_{f_0}[\phi_1] + \partial_f \mu|_{f_0}[\phi_2].\end{aligned}$$

Furthermore, we have for $\phi \in \mathcal{F}$ and $\lambda \in \mathbb{R}$ that

$$\partial_f \mu|_{f_0}[\lambda\phi] = \frac{\partial}{\partial\epsilon}\Big[\mu[f_0 + \epsilon\lambda\phi]\Big]_{\epsilon=0} = \lambda \cdot \frac{\partial}{\partial\epsilon}\Big[\mu[f_0 + \epsilon\phi]\Big]_{\epsilon=0} = \lambda \cdot \partial_f \mu|_{f_0}[\phi].$$

Hence T is linear. Using the assumption, that $\mathcal{D}_x$ is the empirical distribution on a finite subset of $\mathbb{R}^{n_0}$, we conclude, that $\mathcal{F}/\sim$ is finite-dimensional. Thus, there exists $m := \dim(F/\sim)$ and a basis $[e_1], \ldots, [e_m] \in \mathcal{F}/\sim$, such that for any $[f] = \sum_{i=1}^m \alpha_i [e_i] \in \mathcal{F}/\sim$ we observe, that

$$\big|T[f]\big| = \Big|\sum_{i=1}^{m} \alpha_i T[e_i]\Big| \leqslant \sum_{i=1}^{m} |\alpha_i| \cdot \big|T[e_i]\big| \leqslant \max_{j=1,\ldots,m} |\alpha_j| \cdot \sum_{i=1}^{m} \big|T[e_i]\big|.$$

Since $\max_{j=1,\dots,m} |\alpha_j|$ defines a norm on $\mathcal{F}/\sim$, based on the equivalence of norms in finite-dimensional spaces, we conclude the existence of a constant $c_1 \geqslant 0$, such that

$$\forall \alpha_1, \dots, \alpha_m \in \mathbb{R} : \max_{j=1,\dots,m} |\alpha_j| \leqslant c_1 \Big\| \sum_{i=1}^{m} \alpha_i [e_i] \Big\|_{\mathcal{F}/\sim}.$$

Using this and another constant $c_2 \coloneqq c_1 \sum_{i=1}^{m} \big|T[e_i]\big|$, we derive

$$\forall [f] \in F/\sim : \big|T[f]\big| \leqslant \max_{j=1,\dots,m} |\alpha_j| \cdot \sum_{i=1}^{m} \big|T[e_i]\big| \leqslant c_2 \Big\| \sum_{i=1}^{m} \alpha_i [e_i] \Big\|_{\mathcal{F}/\sim} = c_2 \big\|[f]\big\|_{F/\sim}.$$

Thus, the operator T is bounded. In combination with the linearity of T, we conclude

$$\forall [f], [h] \in \mathcal{F}/\sim : \Big| T\big([f] + [h]\big) - T[f] \Big| = \Big| T[h] \Big| \leqslant c \big\|[h]\big\|_{F/\sim}$$

for some constant $c \geqslant 0$. Taking the limit $[h] \to 0$ yields the continuity of T at $[f]$. Thus, T is linear and continuous, which implies that $\partial_f \mu|_{f_0}$ is indeed linear and continuous. □

Thus, the functional derivative $\partial_f \mu|_{f_0}$ can be viewed as an element of $\mathcal{F}^*$. According to theorem 3.8, we conclude the existence of a functional gradient $\nabla \mu|_{f_0}$, such that

$$\forall \phi \in \mathcal{F} : \big\langle \nabla \mu|_{f_0}, \phi \big\rangle_{\mathcal{D}_x}.$$

Theorem 3.8 is a weakening of the Riesz representation theorem, stating that any Hilbert space $\mathcal{H}$ is isometric and isomorphic to its dual space $\mathcal{H}^*$ via $J : \mathcal{H} \to \mathcal{H}^* : x \mapsto \langle x, \cdot \rangle_{\mathcal{H}}$. However, since $\langle \cdot, \cdot \rangle_{\mathcal{D}_x}$ is only positive semidefinite and not positive definite we lose the property of J being isometric and therefore injective. This is a severe problem for our approach to minimizing a functional via gradient descent because the functional gradient is not uniquely determined. Namely any descent direction from the corresponding equivalence class according to $\sim$ could be chosen for the iterative updates, but we have no criterium to select one of them.

In terms of machine learning scenarios, this problem can be explained as follows. We assume that there exists an unknown probability distribution $\mathcal{D}_x$ and approximate it by the empirical distribution corresponding to our data set. Then, we could minimize the functional cost C via the same iterative approach in gradient descent by choosing any element from the corresponding equivalence class of functional gradients. This would perfectly minimize C on the training data set. However, we have no criterium to optimize our target function on unseen data points, which is the main objective of our machine learning task. Thus, we have to find a solution that generalizes the functional derivative to values outside the data set.

Kernel gradient descent is a method, that addresses this problem by minimizing the cost C in a special subspace of $\mathcal{F}$, namely a reproducing kernel Hilbert space, in which a unique descent direction can be constructed. This idea will be explained in detail in section 3.4, but first, we need some background knowledge on kernels and reproducing kernel Hilbert spaces.

3.3 Reproducing Kernel Hilbert Space

Reproducing kernel Hilbert spaces are widely used in machine learning scenarios. In the scalar case, their main application is to reduce the computation effort needed or to simplify infinite-dimensional optimization problems to finite-dimensional ones via the kernel trick. Here, we will study a generalization of this concept to vector-valued reproducing kernel Hilbert spaces. These are based on the definition of multi-dimensional kernels from [8].

Definition 3.10. *A multi-dimensional kernel over $\mathbb{R}^{n_0}$ is a measurable map*

$$K : \mathbb{R}^{n_0} \times \mathbb{R}^{n_0} \to \mathbb{R}^{n_l \times n_l} : (x, x') \mapsto K(x, x'),$$

such that $K(x, x') = K(x', x) = K(x, x')^\mathsf{T}$ for any $x, x' \in \mathbb{R}^{n_0}$.

Equivalently, one could define a kernel K as a symmetric tensor in $\mathcal{F} \otimes \mathcal{F}$. Further on, we will always work with multi-dimensional kernels $K : \mathbb{R}^{n_0} \times \mathbb{R}^{n_0} \to \mathbb{R}^{n_l \times n_l}$ and for reasons of simplicity just call them kernels. We are especially interested in positive semidefinite kernels, as they naturally induce vector-valued reproducing kernel Hilbert spaces.

Definition 3.11. *We say that a kernel K is positive semidefinite if it suffices*

$$\forall m \in \mathbb{N} : \forall x_1, \ldots, x_m \in \mathbb{R}^{n_0} : \forall y_1, \ldots, y_m \in \mathbb{R}^{n_l} : \sum_{i=1}^{m} \sum_{j=1}^{m} y_i^\mathsf{T} K(x_i, x_j) y_j \geqslant 0.$$

Based on this definition, we can formulate the following theorem according to [11].

Theorem 3.12. *Let K be a positive semidefinite kernel and define*

$$\mathcal{H}_0 \coloneqq span\Big\{K_x y : \mathbb{R}^{n_0} \to \mathbb{R}^{n_l} : z \mapsto K(z, x)y \mid x \in \mathbb{R}^{n_0}, y \in \mathbb{R}^{n_l}\Big\},$$

then there exists an inner product $\langle \cdot, \cdot \rangle_{\mathcal{H}_K}$, such that the closure $\mathcal{H}_K \coloneqq \overline{\mathcal{H}_0}$ is a Hilbert space. We call $\big(\mathcal{H}_K, \langle \cdot, \cdot \rangle_{\mathcal{H}_K}\big)$ the vector-valued reproducing kernel Hilbert space associated with K.

Proof. The idea of the proof is to construct an inner product on $\mathcal{H}_0$, and afterward complete the space by adding all limits of Cauchy sequences in $\mathcal{H}_0$. We will denote $f, g, h \in \mathcal{H}_K$ as

$$f = \sum_i K_{x_i} y_i \quad \wedge \quad g = \sum_j K_{\hat{x}_j} \hat{y}_j \quad \wedge \quad h = \sum_k K_{\bar{x}_k} \bar{y}_k.$$

Based on this notation, we can define the inner product

$$\langle \cdot, \cdot \rangle_{\mathcal{H}_0} : \mathcal{H}_0 \times \mathcal{H}_0 \to \mathbb{R} : (f, g) \mapsto \sum_{i,j} y_i^\mathsf{T} K(x_i, \hat{x}_j) \hat{y}_j.$$

To prove that $\langle \cdot, \cdot \rangle_{\mathcal{H}_0}$ is indeed an inner product, we start with linearity and symmetry.

(i) $\langle f + g, h \rangle_{\mathcal{H}_0} = \langle f, h \rangle_{\mathcal{H}_0} + \langle g, h \rangle_{\mathcal{H}_0}$:

$$\langle f + g, h \rangle_{\mathcal{H}_0} = \sum_{i,k} y_i^\mathsf{T} K(x_i, \bar{x}_k) \bar{y}_k + \sum_{j,k} \hat{y}_j^\mathsf{T} K(\hat{x}_j, \bar{x}_k) \bar{y}_k = \langle f, h \rangle_{\mathcal{H}_0} + \langle g, h \rangle_{\mathcal{H}_0},$$

(ii) $\langle \lambda f, g \rangle_{\mathcal{H}_0} = \lambda \langle f, g \rangle_{\mathcal{H}_0}$:

$$\langle \lambda f, g \rangle_{\mathcal{H}_0} = \sum_{i,j} (\lambda y_i)^\mathsf{T} K(x_i, \hat{x}_j)\hat{y}_j = \lambda \cdot \sum_{i,j} y_i^\mathsf{T} K(x_i, \hat{x}_j)\hat{y}_j = \lambda \langle f, g \rangle_{\mathcal{H}_0},$$

(iii) $\langle f, g \rangle_{\mathcal{H}_0} = \langle g, f \rangle_{\mathcal{H}_0}$:

$$\langle f, g \rangle_{\mathcal{H}_0} = \sum_{i,j} y_i^\mathsf{T} K(x_i, \hat{x}_j)\hat{y}_j = \sum_{i,j} y_i^\mathsf{T} K(\hat{x}_j, x_i)^\mathsf{T}\hat{y}_j = \sum_{i,j} \hat{y}_j^\mathsf{T} K(\hat{x}_j, x_i) y_i = \langle g, f \rangle_{\mathcal{H}_0}.$$

Next, we need to show that $\langle \cdot, \cdot \rangle_{\mathcal{H}_0}$ is strictly positive definite. To do this, it should be noted that, for any $x \in \mathbb{R}^{n_0}, y \in \mathbb{R}^{n_l}$ and $f \in \mathcal{H}_0$ the so-called reproducing property holds. That is

$$\langle f(x), y \rangle = \sum_i \big(K(x, x_i) y_i\big)^\mathsf{T} y = \sum_{i,j} y_i^\mathsf{T} K(x_i, x) y = \langle f, K_x y \rangle_{\mathcal{H}_0}.$$

Moreover, since the kernel K is assumed to be positive semidefinite, we have

$$\forall f \in \mathcal{H}_0 : \langle f, f \rangle_{\mathcal{H}_0} = \sum_{i,j} y_i^\mathsf{T} K(x_i, x_j) y_j \geqslant 0.$$

To prove strict definiteness, we can apply the Cauchy-Schwarz inequality and derive

$$\langle f(x), y \rangle^2 = \langle f, K_x y \rangle_{\mathcal{H}_0}^2 \leqslant \langle f, f \rangle_{\mathcal{H}_0} \cdot \langle K_x y, K_x y \rangle_{\mathcal{H}_0}.$$

Again, using the Cauchy-Schwarz inequality, we observe that

$$\langle K_x y, K_x y \rangle_{\mathcal{H}_0} = y^T K(x,x) y = \langle y, K(x,x) y \rangle \leqslant \|y\|_2 \cdot \|K(x,x) y\|_2 \leqslant \|y\|_2^2 \cdot \|K(x,x)\|_2,$$

where $\|K(x,x)\|_2$ denotes the spectral norm of $K(x,x)$. In total, we can conclude

$$\langle f(x), y \rangle^2 \leqslant \langle f, f \rangle_{\mathcal{H}_0} \cdot \|y\|_2^2 \cdot \|K(x,x)\|_2.$$

Since x, y and f were chosen arbitrarily, we conclude that

$$\forall f \in \mathcal{H}_0 \ : \ \langle f, f \rangle_{\mathcal{H}_0} = 0 \ \Leftrightarrow \ f \equiv 0.$$

In other words, $\langle \cdot, \cdot \rangle_{\mathcal{H}_0}$ is strictly positive definite and therefore an inner product.

It remains to prove that the completion of $\mathcal{H}_0$ is a Hilbert space. Let $(f_n)_{n=1}^\infty$ be a Cauchy sequence in $\mathcal{H}_0$. For any $x \in \mathbb{R}^{n_0}$, we can apply the same argumentation as before, so that

$$\langle f_n(x) - f_m(x), y \rangle^2 \leqslant \|f_n - f_m\|_{\mathcal{H}_0} \cdot \|y\|_2^2 \cdot \|K(x,x)\|_2 \leqslant c \cdot \|f_n - f_m\|_{\mathcal{H}_0}$$

for a constant $c > 0$. Since $(f_n)_{n=1}^\infty$ was assumed to be Cauchy with respect to $\| \cdot \|_{\mathcal{H}_0}$, we conclude that $(f_n(x))_{n=1}^\infty$ is a Cauchy sequence in $\mathbb{R}^{n_l}$. Hence the limit

$$f_{lim} : \mathbb{R}^{n_0} \to \mathbb{R}^{n_l} : x \mapsto \lim_{n\to\infty} f_n(x)$$

is well-defined. Now let $\mathcal{H}_K$ be the completion of $\mathcal{H}_0$ by adding all these limits of Cauchy sequences. Thus for any $f, g \in \mathcal{H}_K$ there exist sequences $(f_n)_{n=1}^\infty$ and $(g_n)_{n=1}^\infty$ in $\mathcal{H}_0$ with

$$f = \lim_{n\to\infty} f_n \quad \wedge \quad g = \lim_{n\to\infty} g_n.$$

Based on the continuity of the inner product $\langle\cdot,\cdot\rangle_{\mathcal{H}_0}$, we are able to define the inner product

$$\langle\cdot,\cdot\rangle_{\mathcal{H}_K} : \mathcal{H}_K \times \mathcal{H}_K \to \mathbb{R} : (f,g) \mapsto \lim_{n\to\infty} \langle f_n, g_n\rangle_{\mathcal{H}_0}.$$

To ensure that $\langle\cdot,\cdot\rangle_{\mathcal{H}_K}$ is well-defined, we have to prove the existence of the limit. For this, we let $n, m \in \mathbb{N}$ be arbitrary and apply the Cauchy-Schwarz inequality to derive

$$\begin{aligned}\Big|\langle f_n, g_n\rangle_{\mathcal{H}_0} - \langle f_m, g_m\rangle_{\mathcal{H}_0}\Big| &= \Big|\langle f_n, g_n\rangle_{\mathcal{H}_0} - \langle f_n, g_m\rangle_{\mathcal{H}_0} + \langle f_n, g_m\rangle_{\mathcal{H}_0} - \langle f_m, g_m\rangle_{\mathcal{H}_0}\Big| \\ &= \Big|\langle f_n, g_n - g_m\rangle_{\mathcal{H}_0} + \langle f_n - f_m, g_m\rangle_{\mathcal{H}_0}\Big| \\ &\leqslant \Big|\langle f_n, g_n - g_m\rangle_{\mathcal{H}_0}\Big| + \Big|\langle f_n - f_m, g_m\rangle_{\mathcal{H}_0}\Big| \\ &\leqslant \|f_n\|_{\mathcal{H}_0}\|g_n - g_m\|_{\mathcal{H}_0} + \|f_n - f_m\|_{\mathcal{H}_0}\|g_m\|_{\mathcal{H}_0}.\end{aligned}$$

Furthermore, we observe that

$$\Big|\|f_n\|_{\mathcal{H}_0} - \|f_m\|_{\mathcal{H}_0}\Big| \leqslant \|f_n - f_m\|_{\mathcal{H}_0}.$$

Since $(f_n)_{n=1}^{\infty}$ is a Cauchy sequence in $\mathcal{H}_0$, there exists a constant $c_f \geqslant 0$, such that

$$\lim_{n\to\infty} \|f_n\|_{\mathcal{H}_0} = c_f.$$

Using the same argumentation for $(g_n)_{n=1}^{\infty}$, we conclude the existence of $c_g \geqslant 0$, such that

$$\lim_{n\to\infty} \|g_n\|_{\mathcal{H}_0} = c_g.$$

Since $(f_n)_{n=1}^{\infty}$ and $(g_n)_{n=1}^{\infty}$ are Cauchy sequences in $\mathcal{H}_0$ we conclude, that $(\langle f_n, g_n\rangle_{\mathcal{H}_0})_{n=1}^{\infty}$ is a Cauchy sequence in $\mathbb{R}$. This implies well-definiteness of $\langle\cdot,\cdot\rangle_{\mathcal{H}_K}$.

It remains to prove that $\langle\cdot,\cdot\rangle_{\mathcal{H}_K}$ is indeed an inner product. We verify

(i) $\langle f+g, h\rangle_{\mathcal{H}_K} = \langle f, h\rangle_{\mathcal{H}_K} + \langle g, h\rangle_{\mathcal{H}_K}$:

$$\langle f+g, h\rangle_{\mathcal{H}_K} = \lim_{n\to\infty}\langle f_n, h_n\rangle_{\mathcal{H}_0} + \lim_{n\to\infty}\langle g_n, h_n\rangle_{\mathcal{H}_0} = \langle f, h\rangle_{\mathcal{H}_K} + \langle g, h\rangle_{\mathcal{H}_K},$$

(ii) $\langle \lambda f, g\rangle_{\mathcal{H}_K} = \lambda\langle f, g\rangle_{\mathcal{H}_K}$:

$$\langle \lambda f, g\rangle_{\mathcal{H}_K} = \lim_{n\to\infty}\langle \lambda f_n, g_n\rangle_{\mathcal{H}_0} = \lambda \cdot \lim_{n\to\infty}\langle f_n, g_n\rangle_{\mathcal{H}_0} = \lambda\langle f, g\rangle_{\mathcal{H}_K},$$

(iii) $\langle f, g\rangle_{\mathcal{H}_K} = \langle g, f\rangle_{\mathcal{H}_K}$:

$$\langle f, g\rangle_{\mathcal{H}_K} = \lim_{n\to\infty}\langle f_n, g_n\rangle_{\mathcal{H}_0} = \lim_{n\to\infty}\langle g_n, f_n\rangle_{\mathcal{H}_0} = \langle g, f\rangle_{\mathcal{H}_K}.$$

Again, by Cauchy-Schwarz we have for any $x \in \mathbb{R}^{n_0}$, $y \in \mathbb{R}^{n_l}$ and $f \in \mathcal{H}_K$, that

$$\big\langle f(x), y\big\rangle^2 = \lim_{n\to\infty} \big\langle f_n(x), y\big\rangle^2 \leqslant \lim_{n\to\infty} \big\langle f_n, f_n\big\rangle_{\mathcal{H}_0} \cdot \|y\|_2^2 \cdot \|K(x,x)\|_2 = c \cdot \big\langle f, f\big\rangle_{\mathcal{H}_K}$$

for a constant $c \geqslant 0$. Hence $\langle\cdot,\cdot\rangle_{\mathcal{H}_K}$ is positive definite and therefore an inner product. □

The naming arises from the fact that the following reproducing property holds.

Corollary 3.13. *For any $f \in \mathcal{H}_K$ the reproducing property holds, that is*

$$\forall x \in \mathbb{R}^{n_0}, y \in \mathbb{R}^{n_l} : \big\langle f(x), y \big\rangle = \big\langle f, K_x y \big\rangle_{\mathcal{H}_K}.$$

Proof. Let $x \in \mathbb{R}^{n_0}, y \in \mathbb{R}^{n_l}$ and $f \in \mathcal{H}_K$ be arbitrary. We investigate the following two cases.

Case $f \in \mathcal{H}_0$: We have already seen in the proof of theorem 3.12, that

$$\big\langle f(x), y \big\rangle = \sum_i \big(K(x, x_i) y_i\big)^\intercal y = \sum_{i,j} y_i^\intercal K(x_i, x) y = \big\langle f, K_x y \big\rangle_{\mathcal{H}_K}.$$

Case $f \notin \mathcal{H}_0$: By construction of $\mathcal{H}_K$, there exists a sequence $(f_n)_{n=1}^\infty$ in $\mathcal{H}_0$, such that

$$\big\langle f(x), y \big\rangle = \lim_{n \to \infty} \big\langle f_n(x), y \big\rangle = \lim_{n \to \infty} \big\langle f_n, K_x y \big\rangle_{\mathcal{H}_K} = \big\langle f, K_x y \big\rangle_{\mathcal{H}_K}. \qquad \square$$

This completes the necessary background knowledge on vector-valued reproducing kernel Hilbert spaces. Finally, we can introduce the kernel gradient descent method.

3.4 Kernel Gradient Descent

Kernel gradient descent is a generalization of gradient descent to functionals. The idea is to minimize a differentiable cost C in a special subspace of $\mathcal{F}$, namely a vector-valued reproducing kernel Hilbert space, in which the descent direction is unique. This prevents the problem of choosing a suitable functional gradient. The theory in here is based on [8].

Definition 3.14. *Let $K : \mathbb{R}^{n_0} \times \mathbb{R}^{n_0} \to \mathbb{R}^{n_l \times n_l}$ be a kernel. For $j = 1, \ldots, n_l$, one defines the partial application of K as the function*

$$K_j : \mathbb{R}^{n_0} \times \mathbb{R}^{n_0} \to \mathbb{R}^{n_l} : (x, x') \mapsto K(x, x')_{\cdot,j},$$

where $K(x, x')_{\cdot,j}$ denotes the jth column of the $n_l \times n_l$ matrix $K(x, x')$.

Fixing one argument of K_j results in a function in $\mathcal{F}$. Namely, for any $x \in \mathbb{R}^{n_0}$, we have

$$K_j(x, \cdot) : \mathbb{R}^{n_0} \to \mathbb{R}^{n_l} : x' \mapsto K_j(x, x'),$$

which is an element of the function space $\mathcal{F}$. This enables the following definition.

Definition 3.15. *Given a kernel $K : \mathbb{R}^{n_0} \times \mathbb{R}^{n_0} \to \mathbb{R}^{n_l \times n_l}$, we define the map*

$$\Phi_K : \mathcal{F}^* \to \mathcal{F} : \mu \mapsto f_\mu,$$

mapping a dual element $\mu \in \mathcal{F}^*$ to the function

$$f_\mu : \mathbb{R}^{n_0} \to \mathbb{R}^{n_l} : x \mapsto \begin{bmatrix} \mu\big[K_1(x, \cdot)\big] \\ \vdots \\ \mu\big[K_{n_l}(x, \cdot)\big] \end{bmatrix}.$$

The mapping Φ_K will be used to restrict our minimization problem to a vector-valued reproducing kernel Hilbert space. To explain this in detail we have to mention the following characterization of Φ_K, which is based on the representation theorem 3.8 and the restriction, that the marginal distribution $\mathcal{D}_x$ is given by the empirical distribution on a finite data set.

Lemma 3.16. *Let $\mu = \langle d, \cdot \rangle_{\mathcal{D}_x} \in \mathcal{F}^*$ and K be a kernel, then*

$$\Phi_K(\mu) = \frac{1}{m} \sum_{i=1}^{m} K(\cdot, x_i) d(x_i),$$

where $x_1, \dots, x_m \in \mathbb{R}^{n_0}$ denotes the finite support of the empirical distribution $\mathcal{D}_x$.

Proof. Let $x \in \mathbb{R}^{n_0}$ be arbitrary. By the definition of Φ_K, we have

$$\Phi_K\big(\mu\big)(x) = \Phi_K\Big(\langle d, \cdot \rangle_{\mathcal{D}_x}\Big)(x) = \begin{bmatrix} \mu\big[K_1(x,\cdot)\big] \\ \vdots \\ \mu\big[K_{n_l}(x,\cdot)\big] \end{bmatrix} = \begin{bmatrix} \big\langle d, K_1(x,\cdot)\big\rangle_{\mathcal{D}_x} \\ \vdots \\ \big\langle d, K_{n_l}(x,\cdot)\big\rangle_{\mathcal{D}_x} \end{bmatrix}.$$

Recalling the definition of $\langle \cdot, \cdot \rangle_{\mathcal{D}_x}$, it holds that

$$\big\langle d, K_j(x,\cdot)\big\rangle_{\mathcal{D}_x} = \mathop{\mathbb{E}}_{x' \sim \mathcal{D}_x}\Big[d(x')^\mathsf{T} K_j(x,x')\Big]$$

for $j = 1, \dots, n_l$. Thus we derive

$$\Phi_K\big(\mu\big)(x) = \begin{bmatrix} \big\langle d, K_1(x,\cdot)\big\rangle_{\mathcal{D}_x} \\ \vdots \\ \big\langle d, K_{n_l}(x,\cdot)\big\rangle_{\mathcal{D}_x} \end{bmatrix} = \mathop{\mathbb{E}}_{x' \sim \mathcal{D}_x} \begin{bmatrix} d(x')^\mathsf{T} K_1(x,x') \\ \vdots \\ d(x')^\mathsf{T} K_{n_l}(x,x') \end{bmatrix} = \mathop{\mathbb{E}}_{x' \sim \mathcal{D}_x}\Big[d(x')^\mathsf{T} K(x,x')\Big]^\mathsf{T}.$$

With $K(x,x')^\mathsf{T} = K(x,x')$ for every $x, x' \in \mathbb{R}^{n_0}$ by definition, we conclude

$$\Phi_K\big(\mu\big)(x) = \mathop{\mathbb{E}}_{x' \sim \mathcal{D}_x}\Big[d(x')^\mathsf{T} K(x,x')\Big]^\mathsf{T} = \mathop{\mathbb{E}}_{x' \sim \mathcal{D}_x}\Big[K(x,x')^\mathsf{T} d(x')\Big] = \mathop{\mathbb{E}}_{x' \sim \mathcal{D}_x}\Big[K(x,x') d(x')\Big].$$

The last step consists of using the assumption, that $\mathcal{D}_x$ is the empirical distribution on a finite data set $x_1, \dots, x_m \in \mathbb{R}^{n_0}$. This enables us to derive

$$\Phi_K\big(\mu\big)(x) = \mathop{\mathbb{E}}_{x' \sim \mathcal{D}_x}\Big[K(x,x') d(x')\Big] = \frac{1}{m} \sum_{i=1}^{m} K(x, x_i) d(x_i),$$

which proves the statement of the lemma. □

Note that the definition of Φ_K does not require definiteness of the kernel K. However, in the case where K is positive semidefinite, we have seen in theorem 3.12 that K induces a vector-valued reproducing kernel Hilbert space. Thus, Φ_K can be understood as a mapping

$$\Phi_K : \mathcal{F}^* \to \mathcal{H}_K : \mu \mapsto \frac{1}{m} \sum_{i=1}^{m} K_{x_i} y_i,$$

where $y_i := d(x_i)$. Based on the notion of Φ_K, we can define the kernel gradient of a functional.

Definition 3.17. *Let K be a kernel and the functional $\mu : \mathcal{F} \to \mathbb{R}$ be differentiable at $f_0 \in \mathcal{F}$. The kernel gradient of μ at f_0 with respect to K is defined as*

$$\nabla_K \mu|_{f_0} : \mathbb{R}^{n_0} \to \mathbb{R}^{n_l} : x \mapsto \Phi_K\Big(\partial_f \mu|_{f_0}\Big)(x).$$

The kernel gradient is well-defined based on the restriction, that $\mathcal{D}_x$ is given by the empirical distribution on a finite data set. In lemma 3.9, we have shown that $\partial_f \mu|_{f_0} \in \mathcal{F}^*$ for any $f_0 \in \mathcal{F}$. In the case where the kernel K is positive semidefinite, we can think of the kernel gradient as a descent direction in the vector-valued reproducing kernel Hilbert space associated with K.

In terms of minimizing a differentiable cost $C : \mathcal{F} \to \mathbb{R}$, the notation translates as follows. By theorem 3.8 there exists $d|_{f_0} \in \mathcal{F}$, such that the functional derivative of C is given by

$$\partial_f C|_{f_0} = \big\langle d|_{f_0}, \cdot \big\rangle_{\mathcal{D}_x}.$$

The element $d|_{f_0} \in \mathcal{F}$ can be thought of as a functional gradient. Using lemma 3.16, the kernel gradient of C at f_0 with respect to any kernel K can be written as

$$\nabla_K C|_{f_0}(x) = \Phi_K\Big(\big\langle d|_{f_0}, \cdot \big\rangle_{\mathcal{D}_x}\Big)(x) = \frac{1}{m}\sum_{i=1}^{m} K(x, x_i) d|_{f_0}(x_i).$$

In contrast to a functional gradient $d|_{f_0} \in \mathcal{F}$, the kernel gradient is in general not the direction of steepest descent. Instead, it computes a linear combination of the values of a functional gradient $d|_{f_0}$ at the data set, using the kernel K. This construction is independent of the functional gradient chosen since they are equal on the data. Thanks to the kernel K, the kernel gradient generalizes to values outside the data set, which prevents the problem of choosing a suitable descent direction from the set of functional gradients.

Based on the uniqueness of the kernel gradient, we can finally generalize the gradient descent algorithm to kernel gradient descent, which is formalized as follows.

Algorithm 6 Kernel Gradient Descent

```
Input: Kernel $K$, differentiable cost $C : \mathcal{F} \to \mathbb{R}$, $f_0 \in \mathcal{F}$, $\epsilon \geqslant 0$
1: Initialize $k \leftarrow 0$;
2: while $\big\|\nabla_K C|_{f_k}\big\|_{\mathcal{D}_x} > \epsilon$ do:
3:     Choose a step size $\eta_k > 0$;
4:     Set $f_{k+1} \leftarrow f_k - \eta_k \cdot \nabla_K C|_{f_k}$ and $k \leftarrow k + 1$;
5: end while
```

Again, this is based on the assumption, that $\mathcal{D}_x$ is given by the empirical distribution on a finite data set. Otherwise, we could not guarantee, that the sequence of kernel gradients produced by the algorithm is well-defined.

In the convergence analysis of gradient descent, with the choice of an appropriate step size, the empirical error E was strictly decreasing along the parameter sequence $(w_k)_{k=0}^{\infty}$ in $\mathbb{R}^n$. This guaranteed the convergence to a critical point, as long as the objective function was bounded from below. We will analyze kernel gradient descent in a similar manner. However, in contrast to the analysis of gradient descent, we will investigate the change in C during kernel gradient descent along a continuous curve in the function space $\mathcal{F}$.

Definition 3.18. *Let K be a kernel. A continuous time-dependent function*

$$f : [0, \infty) \to \mathcal{F} : t \mapsto f(t)$$

is said to follow the kernel gradient descent on C with respect to K if it satisfies

$$\partial_t f(t) = -\nabla_K C|_{f(t)}.$$

Roughly speaking, we can think of such a time-dependent function as the continuous extension to the sequence $(f_k)_{k=0}^{\infty}$ in $\mathcal{F}$, produced by kernel gradient descent on C, when using an infinitely small step size. Hence $f(0) = f_0$ denotes the initialization of kernel gradient descent. Furthermore, the convergence analysis is based on the observation, that kernels induce symmetric bilinear forms with respect to $\mathcal{D}_x$. This is formulated in the following lemma.

Lemma 3.19. *A kernel K induces the symmetric bilinear form*

$$\langle \cdot, \cdot \rangle_K : \mathcal{F} \times \mathcal{F} \to \mathbb{R} : (f, g) \mapsto \langle f, g \rangle_K \coloneqq \mathop{\mathbb{E}}_{x,x' \sim \mathcal{D}_x} \Big[f(x)^{\mathsf{T}} K(x, x') g(x') \Big],$$

where $x, x' \in \mathbb{R}^{n_0}$ are drawn independently according to $\mathcal{D}_x$.

Proof. Let $f_1, f_2, f, g \in \mathcal{F}$ and $\lambda \in \mathbb{R}$. We observe that

(i) $\langle f_1 + f_2, g \rangle_K = \langle f_1, g \rangle_K + \langle f_2, g \rangle_K$:

$$\begin{aligned} \langle f_1 + f_2, g \rangle_K &= \mathop{\mathbb{E}}_{x,x' \sim \mathcal{D}_x} \Big[\big(f_1(x) + f_2(x)\big)^{\mathsf{T}} K(x, x') g(x') \Big] \\ &= \mathop{\mathbb{E}}_{x,x' \sim \mathcal{D}_x} \Big[f_1(x)^{\mathsf{T}} K(x, x') g(x') + f_2(x)^{\mathsf{T}} K(x, x') g(x') \Big] \\ &= \mathop{\mathbb{E}}_{x,x' \sim \mathcal{D}_x} \Big[f_1(x)^{\mathsf{T}} K(x, x') g(x') \Big] + \mathop{\mathbb{E}}_{x,x' \sim \mathcal{D}_x} \Big[f_2(x)^{\mathsf{T}} K(x, x') g(x') \Big] \\ &= \langle f_1, g \rangle_K + \langle f_2, g \rangle_K, \end{aligned}$$

(ii) $\langle \lambda f, g \rangle_K = \lambda \langle f, g \rangle_K$:

$$\begin{aligned} \langle \lambda f, g \rangle_K &= \mathop{\mathbb{E}}_{x,x' \sim \mathcal{D}_x} \Big[\lambda f(x)^{\mathsf{T}} K(x, x') g(x') \Big] \\ &= \lambda \cdot \mathop{\mathbb{E}}_{x,x' \sim \mathcal{D}_x} \Big[f(x)^{\mathsf{T}} K(x, x') g(x') \Big] = \lambda \langle f, g \rangle_K, \end{aligned}$$

(iii) $\langle f, g \rangle_K = \langle g, f \rangle_K$:

$$\begin{aligned} \langle f, g \rangle_K &= \mathop{\mathbb{E}}_{x,x' \sim \mathcal{D}_x} \Big[f(x)^{\mathsf{T}} K(x, x') g(x') \Big] \\ &= \mathop{\mathbb{E}}_{x,x' \sim \mathcal{D}_x} \Big[f(x)^{\mathsf{T}} K(x', x)^{\mathsf{T}} g(x') \Big] \\ &= \mathop{\mathbb{E}}_{x,x' \sim \mathcal{D}_x} \Big[g(x')^{\mathsf{T}} K(x', x) f(x) \Big] = \langle g, f \rangle_K. \end{aligned}$$

Thus $\langle \cdot, \cdot \rangle_K$ is indeed symmetric and bilinear. □

The approach to proving convergence of kernel gradient descent is to show that the cost C is strictly decreasing along the curve, defined by a continuous time-dependent function f, that follows the kernel gradient descent on C with respect to a kernel K. In other words

$$\forall t \in [0, \infty) : \partial_t C|_{f(t)} \coloneqq \partial_t \big(C \circ f\big)(t) \leqslant 0,$$

such that the inequality is strict, as long as kernel gradient descent has not converged. More precisely, convergence along the curve $f(t)$ refers to finding some $t \in [0, \infty)$, such that

$$\big\|\nabla_K C|_{f(t)}\big\|_{\mathcal{D}_x} = 0.$$

Based on this, we provide the following result, to prove convergence of kernel gradient descent.

Lemma 3.20. *Let K be a kernel and f be a time-dependent function, that follows the kernel gradient descent on C with respect to K. During kernel gradient descent, $C \circ f$ evolves as*

$$\partial_t C|_{f(t)} = -\big\langle d|_{f(t)}, d|_{f(t)} \big\rangle_K,$$

where $d|_{f(t)} \in \mathcal{F}$ denotes a representative of $\partial_f C|_{f(t)} = \big\langle d|_{f(t)}, \cdot \big\rangle_{\mathcal{D}_x} \in \mathcal{F}^*$ for $t \in \mathbb{R}$.

Proof. According to the chain rule for functionals in Appendix A of [9], we have

$$\partial_t C|_{f(t)} = \big\langle d|_{f(t)}, \partial_t f(t) \big\rangle_{\mathcal{D}_x} = \big\langle d|_{f(t)}, -\nabla_K C|_{f(t)} \big\rangle_{\mathcal{D}_x},$$

where we have used the assumption, that f follows the kernel gradient descent on C with respect to the kernel K. Recalling the definition of $\nabla_K C|_{f(t)}$, which was given by

$$\nabla_K C|_{f(t)} = \frac{1}{m} \sum_{i=1}^{m} K(\cdot, x_i) d|_{f(t)}(x_i),$$

we derive by the linearity of $\langle \cdot, \cdot \rangle_{\mathcal{D}_x}$, that

$$\begin{aligned}
\big\langle d|_{f(t)}, -\nabla_K C|_{f(t)} \big\rangle_{\mathcal{D}_x} &= -\frac{1}{m} \sum_{i=1}^{m} \big\langle d|_{f(t)}, K(\cdot, x_i) d|_{f(t)}(x_i) \big\rangle_{\mathcal{D}_x} \\
&= -\frac{1}{m} \sum_{i=1}^{m} \mathop{\mathbb{E}}_{x \sim \mathcal{D}_x} \Big[d|_{f(t)}(x)^\intercal K(x, x_i) d|_{f(t)}(x_i) \Big] \\
&= - \mathop{\mathbb{E}}_{x' \sim \mathcal{D}_x} \mathop{\mathbb{E}}_{x \sim \mathcal{D}_x} \Big[d|_{f(t)}(x)^\intercal K(x, x') d|_{f(t)}(x') \Big] \\
&= -\big\langle d|_{f(t)}, d|_{f(t)} \big\rangle_K.
\end{aligned}$$

Finally, we conclude that

$$\partial_t C|_{f(t)} = \big\langle d|_{f(t)}, -\nabla_K C|_{f(t)} \big\rangle_{\mathcal{D}_x} = -\big\langle d|_{f(t)}, d|_{f(t)} \big\rangle_K.$$

This completes the proof. □

Thus, to guarantee that the cost C is strictly decreasing, we have to ensure that

$$\forall t \in [0, \infty) : \left\langle d|_{f(t)}, d|_{f(t)} \right\rangle_K \geqslant 0,$$

such that the inequality is strict, except at points $t \in [0, \infty)$, where the kernel gradient vanishes, that is $\left\| \nabla_K C|_{f(t)} \right\|_{\mathcal{D}_x} = 0$. In contrast to $\langle \cdot, \cdot \rangle_{\mathcal{D}_x}$, the bilinear map $\langle \cdot, \cdot \rangle_K$ is in general not positive semidefinite. This can be seen from a simple counterexample. Consider

$$K : \mathbb{R}^{n_0} \times \mathbb{R}^{n_0} \to \mathbb{R}^{n_l} : x \mapsto -I_{n_l},$$

then for $f \equiv (1, \ldots, 1)^T \in \mathcal{F}$, we observe that

$$\langle f, f \rangle_K = \mathop{\mathbb{E}}_{x,x' \sim \mathcal{D}_x} \left[f(x)^\mathsf{T} K(x, x') f(x') \right] = -n_l < 0.$$

However, the semidefiniteness of $\langle \cdot, \cdot \rangle_K$ can be realized with the choice of a suitable kernel.

Definition 3.21. *We say that a kernel K is positive definite with respect to $\mathcal{D}_x$, if*

$$\forall f \in \mathcal{F} : \|f\|_{\mathcal{D}_x} > 0 \implies \langle f, f \rangle_K > 0.$$

Given a positive definite kernel K, the bilinear form $\langle \cdot, \cdot \rangle_K$ is positive semidefinite. This follows directly from definition 3.21 and the fact that for $f \in \mathcal{F}$ with $\|f\|_{\mathcal{D}_x} = 0$, we have

$$\|f\|^2_{\mathcal{D}_x} = \langle f, f \rangle_{\mathcal{D}_x} = \mathop{\mathbb{E}}_{x \sim \mathcal{D}_x} \left[f(x)^\mathsf{T} f(x) \right] = \mathop{\mathbb{E}}_{x \sim \mathcal{D}_x} \left[\sum_{i=1}^{n_l} f_i(x)^2 \right] = 0.$$

Therefore, for every $x \sim \mathcal{D}_x$ it holds, that $f(x) = 0$ and we conclude

$$\langle f, f \rangle_K = \mathop{\mathbb{E}}_{x,x' \sim \mathcal{D}_x} \left[f(x)^\mathsf{T} K(x, x') f(x') \right] = 0,$$

so $\langle \cdot, \cdot \rangle_K$ is positive semidefinite. Moreover, a positive definite kernel with respect to $\mathcal{D}_x$, ensures that C is strictly decreasing during kernel gradient descent. By definition 3.21, we just have to investigate points $t \in \mathbb{R}$, such that $\left\| d|_{f(t)} \right\|_{\mathcal{D}_x} = 0$. In this case, we have

$$\forall x \sim \mathcal{D}_x : d|_{f(t)}(x) = 0,$$

so that we can conclude $\nabla_K C|_{f(t)} = 0$ based on the construction via the data points, namely

$$\nabla_K C|_{f(t)} = \frac{1}{m} \sum_{i=1}^{m} K(\cdot, x_i) d|_{f(t)}(x_i) = 0.$$

Finally, if the cost is convex and bounded from below and the kernel K is positive definite with respect to $\mathcal{D}_x$, we conclude the convergence of $f(t)$ to a global minimum as $t \to \infty$.

As explained before, it is easy to ensure convexity of the cost C in the function space, such that we have nice convergence properties. This is great for theoretical analysis. In practice, however, we are still in need of the explicit parameters $w \in \mathbb{R}^n$, that describe the objective function. Since, in general, any function can be parametrized in several ways, kernel gradient descent does not prevent us from training neural networks via some sort of gradient descent. Nevertheless, in the next section, a relationship between the two minimization approaches will be established, such that we get a better understanding of gradient descent in the parameter space, based on the properties of kernel gradient descent in the function space.

3.5 Neural Tangent Kernel

The parameter training of neural networks via gradient descent is strongly connected to kernel gradient descent with respect to a special kernel, the neural tangent kernel. To motivate this connection, we start with a simple example, inspired by the approach of [12], in which the objective function depends linearly on its parameters. The example will be generalized, leading to the definition of the neural tangent kernel. The theory here is based on [8].

At first, we let $\mathcal{D}_{\mathcal{F}}$ be any probability distribution on the function space $\mathcal{F}$, whose support is given by a subset of the continuous functions on $\mathbb{R}^{n_0}$. Next, we choose an $n \in \mathbb{N}$ and draw random functions $f^{(i)} : \mathbb{R}^{n_0} \to \mathbb{R}^{n_l}$ for $i = 1, \dots, n$ independently from the distribution $\mathcal{D}_{\mathcal{F}}$. Using these, we can define an objective function, which is parametrized via

$$F^{lin} : \mathbb{R}^n \to \mathcal{F} : w \mapsto f_w \coloneqq \frac{1}{\sqrt{n}} \sum_{i=1}^{n} w_i f^{(i)}.$$

We observe for any $w, w' \in \mathbb{R}^n$, that

$$f_{w+w'} = \frac{1}{\sqrt{n}} \sum_{i=1}^{n} (w + w')_i f^{(i)} = \frac{1}{\sqrt{n}} \sum_{i=1}^{n} w_i f^{(i)} + \frac{1}{\sqrt{n}} \sum_{i=1}^{n} w'_i f^{(i)} = f_w + f_{w'}.$$

Moreover, we have for any $w \in \mathbb{R}^n$ and $\lambda \in \mathbb{R}$ that

$$f_{\lambda w} = \frac{1}{\sqrt{n}} \sum_{i=1}^{n} \lambda w_i f^{(i)} = \lambda \cdot \frac{1}{\sqrt{n}} \sum_{i=1}^{n} w_i f^{(i)} = \lambda f_w,$$

such that F^{lin} is linear in the parameters $w \in \mathbb{R}^n$. Given labeled training data, the objective function can be learned via gradient descent on the empirical error

$$E \coloneqq C \circ F^{lin} : \mathbb{R}^n \to \mathbb{R},$$

where $C : \mathcal{F} \to \mathbb{R}$ denotes a cost functional. Our goal is to construct a kernel K, such that gradient descent on $C \circ F^{lin}$ is equivalent to kernel gradient descent on C with respect to K.

Similar to the approach of analyzing the convergence of kernel gradient descent along a continuous curve in the function space $\mathcal{F}$, we can analyze gradient descent along a continuous curve in the parameter space $\mathbb{R}^n$. For this, we require the following definition.

Definition 3.22. *A continuous time-dependent function*

$$w : [0, \infty) \to \mathbb{R}^n : t \mapsto w(t)$$

is said to follow the gradient descent on $C \circ F^{lin}$, if it satisfies the differential equation

$$\partial_t w(t) = -\nabla\big(C \circ F^{lin}\big)(w(t)) = -\nabla C|_{f_{w(t)}},$$

where $-\nabla C|_{f_{w(t)}}$ denotes the gradient of $C \circ F^{lin}$ at $w(t)$.

Roughly speaking, we can think of such a time-dependent function as the continuous extension to the sequence $(w_k)_{k=0}^{\infty}$ in $\mathbb{R}^n$, produced by gradient descent on $C \circ F^{lin}$, when using an infinitely small step size. Hence $w(0) = w_0$ denotes the initialization of gradient descent.

To establish a relationship between gradient descent and kernel gradient descent, we have to construct a kernel K such that, given a continuous time-dependent function $w : [0, \infty) \to \mathbb{R}^n$ that follows the gradient descent on $C \circ F^{lin}$, the continuous time-dependent function

$$F^{lin} \circ w : [0, \infty) \to \mathcal{F} : t \mapsto f_{w(t)}$$

follows the kernel gradient descent on C with respect to the kernel K. In other words, the evolution of C during kernel gradient descent and the evolution of $C \circ F^{lin}$ during gradient descent are governed by the same parameter curve $w(t)$.

To simplify the further notation, we introduce the following definition.

Definition 3.23. *For $y, \hat{y} \in \mathbb{R}^{n_l}$ we define the outer product $y \otimes \hat{y}$ as the $n_l \times n_l$ matrix*

$$y \otimes \hat{y} \coloneqq y\hat{y}^{\mathsf{T}}.$$

For $i = 1, \ldots, n$, the partial derivatives $\partial_{w_i} F^{lin}$ of the parametrization are given by

$$\partial_{w_i} F^{lin} : \mathbb{R}^n \to \mathcal{F} : w \mapsto \frac{1}{\sqrt{n}} f^{(i)}.$$

Using these, we can introduce the so-called tangent kernel $\tilde{K}$, which is defined as follows.

Definition 3.24. *Given n random functions $f^{(i)} : \mathbb{R}^{n_0} \to \mathbb{R}^{n_l}$ for $i = 1, \ldots, n$, drawn i.i.d. according to some distribution $\mathcal{D}_{\mathcal{F}}$ on the function space $\mathcal{F}$, the tangent kernel is defined as*

$$\tilde{K} \coloneqq \sum_{i=1}^{n} \partial_{w_i} F^{lin}(w) \otimes \partial_{w_i} F^{lin}(w) = \frac{1}{n} \sum_{i=1}^{n} f^{(i)} \otimes f^{(i)}.$$

In other words, $\tilde{K}$ maps a tuple $(x, x') \in \mathbb{R}^{n_0} \times \mathbb{R}^{n_0}$ to the matrix $\tilde{K} \in \mathbb{R}^{n_l \times n_l}$ with entries

$$\tilde{K}_{i,j}(x, x') = \frac{1}{n} \sum_{k=1}^{n} f_i^{(k)}(x) \cdot f_j^{(k)}(x'), \quad i, j = 1, \ldots, n_l.$$

We will prove, that gradient descent on $C \circ F^{lin}$ is equivalent to kernel gradient descent on the cost C with respect to the tangent kernel $\tilde{K}$, which is based on the following result.

Lemma 3.25. *Let $w : [0, \infty) \to \mathbb{R}^n$ be a time-dependent function, that follows the gradient descent on $C \circ F^{lin}$. For $i = 1, \ldots, n$, the change in w_i during gradient descent is given by*

$$\partial_t w_i(t) = -\frac{1}{\sqrt{n}} \langle d|_{f_{w(t)}}, f^{(i)} \rangle_{\mathcal{D}_x},$$

where $d|_{f_{w(t)}} \in \mathcal{F}$ denotes the representative of $\partial_f C|_{f_{w(t)}} = \langle d|_{f_{w(t)}}, \cdot \rangle_{\mathcal{D}_x} \in \mathcal{F}^*$ for $t \in [0, \infty)$.

Proof. According to the chain rule for functionals in Appendix A of [9], we have

$$\partial_t w_i(t) = -\partial_{w_i} C|_{f_{w(t)}} = -\langle d|_{f_{w(t)}}, \partial_{w_i} f_{w(t)} \rangle_{\mathcal{D}_x} = -\langle d|_{f_{w(t)}}, \frac{1}{\sqrt{n}} f^{(i)} \rangle_{\mathcal{D}_x}$$

for $i = 1, \ldots, n$. Using the linearity of $\langle \cdot, \cdot \rangle_{\mathcal{D}_x}$ completes the proof. □

Based on this, we can prove that the dynamics of gradient descent on $C \circ F^{lin}$ and kernel gradient descent on C with respect to the tangent kernel $\tilde{K}$ are governed by the same parameter curve $w(t)$ in $\mathbb{R}^n$. The corresponding result is formulated as follows.

Lemma 3.26. *Let $w : [0, \infty) \to \mathbb{R}^n$ be a time-dependent function, that follows the gradient descent on $C \circ F^{lin}$, then the time-dependent function $F^{lin} \circ w : [0, \infty) \to \mathcal{F} : t \mapsto f_{w(t)}$ follows the kernel gradient descent on C with respect to the tangent kernel $\tilde{K}$. Formally this is*

$$\partial_t \big(F^{lin} \circ w\big)(t) = \partial_t f_{w(t)} = -\nabla_{\tilde{K}} C|_{f_{w(t)}}.$$

Proof. On one hand, we can use the definition of F^{lin} and apply lemma 3.25 to derive

$$\partial_t f_{w(t)} = \frac{1}{\sqrt{n}} \sum_{i=1}^{n} \partial_t w_i(t) \cdot f^{(i)} = -\frac{1}{n} \sum_{i=1}^{n} \langle d|_{f_{w(t)}}, f^{(i)} \rangle_{\mathcal{D}_x} \cdot f^{(i)}.$$

On the other hand, using the equation

$$\tilde{K}(x, x_j) = \frac{1}{n} \sum_{i=1}^{n} f^{(i)}(x) \otimes f^{(i)}(x_j) = \frac{1}{n} \sum_{i=1}^{n} f^{(i)}(x) f^{(i)}(x_j)^{\mathsf{T}},$$

we can write the kernel gradient of C at $f_{w(t)}$ with respect to $\tilde{K}$ as

$$\begin{aligned}
\nabla_{\tilde{K}} C|_{f_{w(t)}}(x) &= \frac{1}{m} \sum_{j=1}^{m} \tilde{K}(x, x_j) d|_{f_{w(t)}}(x_j) \\
&= \frac{1}{m} \sum_{j=1}^{m} \frac{1}{n} \sum_{i=1}^{n} f^{(i)}(x) f^{(i)}(x_j)^{\mathsf{T}} d|_{f_{w(t)}}(x_j) \\
&= \frac{1}{n} \sum_{i=1}^{n} f^{(i)}(x) \cdot \frac{1}{m} \sum_{j=1}^{m} f^{(i)}(x_j)^{\mathsf{T}} d|_{f_{w(t)}}(x_j) \\
&= \frac{1}{n} \sum_{i=1}^{n} f^{(i)}(x) \cdot \mathop{\mathbb{E}}_{x' \sim \mathcal{D}_x} \Big[f^{(i)}(x')^{\mathsf{T}} d|_{f_{w(t)}}(x') \Big] \\
&= \frac{1}{n} \sum_{i=1}^{n} f^{(i)}(x) \cdot \langle f^{(i)}, d|_{f_{w(t)}} \rangle_{\mathcal{D}_x},
\end{aligned}$$

A comparison of the two expressions results in

$$\partial_t f_{w(t)} = -\frac{1}{n} \sum_{i=1}^{n} \langle d|_{f_{w(t)}}, f^{(i)} \rangle_{\mathcal{D}_x} \cdot f^{(i)} = -\nabla_{\tilde{K}} C|_{f_{w(t)}}.$$

In other words, $F^{lin} \circ w$ follows the kernel gradient descent on C with respect to $\tilde{K}$. □

Thus, kernel gradient descent on C with respect to the tangent kernel $\tilde{K}$ is equivalent to gradient descent on $C \circ F^{lin}$. For $i, j = 1, \dots, n_l$ the law of large numbers yields

$$\forall x, x' \in \mathbb{R}^{n_0} : \lim_{n \to \infty} \tilde{K}_{i,j}(x, x') = \lim_{n \to \infty} \frac{1}{n} \sum_{k=1}^{n} f_i^{(k)}(x) \cdot f_j^{(k)}(x') = \mathop{\mathbb{E}}_{f \sim \mathcal{D}_{\mathcal{F}}} \Big[f_i(x) \cdot f_j(x') \Big].$$

This motivates the definition of a constant limiting kernel K, such that for $x, x' \in \mathbb{R}^{n_0}$, the image $K(x, x')$ is given by the $n_l \times n_l$ matrix defined as

$$K_{i,j}(x, x') \coloneqq \mathop{\mathbb{E}}_{f \sim \mathcal{D}_{\mathcal{F}}} \Big[f_i(x) \cdot f_j(x') \Big], \quad i, j = 1, \dots, n_l.$$

This definition is well-defined since we assumed any $f \sim \mathcal{D}_{\mathcal{F}}$ to be continuous on $\mathbb{R}^{n_0}$. The random tangent kernel $\tilde{K}$ is therefore an approximation of the constant limiting kernel K.

The training of neural networks behaves quite similarly to the kernel gradient descent with respect to the tangent kernel. We consider the realization function

$$F : \mathbb{R}^n \to \mathcal{F}|_{\tilde{f}} : w \mapsto f_w \coloneqq \tilde{f}(\cdot, w),$$

mapping a parameter vector $w \in \mathbb{R}^n$ to its corresponding network function, with regard to the architecture $\tilde{f}$. Similarly to the definition of the tangent kernel, we can introduce the neural tangent kernel. Formally, it is given as follows.

Definition 3.27. *Given parameters $w \in \mathbb{R}^n$, the neural tangent kernel is defined as*

$$\Theta_w \coloneqq \sum_{i=1}^{n} \partial_{w_i} F(w) \otimes \partial_{w_i} F(w).$$

In contrast to the previous example, the realization function F is not linear, due to the nested construction of neural networks, so the partial derivatives $\partial_{w_i} F$ are no longer independent of the parameters w. Thus, in contrast to the tangent kernel, which is constant during training due to its independency of the parameters w, the neural tangent kernel is random at initialization and varies during training. As a result, the relationship between gradient descent and kernel gradient descent with respect to the neural tangent kernel is more complex.

Let us clarify this problem. For the same reasons as in lemma 3.26, during training via gradient descent on $C \circ F$, the network function f_w evolves along the differential equation

$$\partial_t f_{w(t)} = -\nabla_{\Theta_w} C|_{f_{w(t)}}.$$

However, in contrast to the situation in lemma 3.26, the kernel Θ_w varies during training, such that there is no direct connection to kernel gradient descent.

In the previous example, the random kernel $\tilde{K}$ converged to the fixed kernel K, as the number of parameters n tended to infinity. This motivates studying the training of neural networks in the infinite width limit. More precisely, the setting where the width of each hidden layer increases to infinity sequentially over the total number of hidden layers. This is denoted as

$$n_1, \dots, n_{l-1} \to \infty \;\equiv\; \lim_{n_{l-1} \to \infty} \cdots \lim_{n_1 \to \infty}.$$

In the following, we assume that the network parameters $w \in \mathbb{R}^n$ are initialized as i.i.d. Gaussian random variables. In other words, for $i = 1, \dots, n$ it holds, that

$$w_i \sim \mathcal{N}(0, 1).$$

Given this initialization, one of the main results in [8] is, that in the infinite width limit, the neural tangent kernel converges in probability to an explicit deterministic limiting kernel.

Theorem 3.28. *Let $f : \mathbb{R}^{n_0} \to \mathbb{R}^{n_l}$ be a neural network of depth l at initialization, with a Lipschitz continuous non-linear activation function $\sigma : \mathbb{R} \to \mathbb{R}$ at each layer. Then, in the limit as the layer widths $n_1, \dots, n_{l-1} \to \infty$ sequentially, the initial neural tangent kernel Θ_{w_0} converges in probability to a deterministic limiting kernel, that is*

$$\Theta_{w_0} \to \Theta_\infty \otimes I_{n_l}.$$

The scalar kernel $\Theta_\infty : \mathbb{R}^{n_0} \times \mathbb{R}^{n_0} \to \mathbb{R}$ is defined recursively as $\Theta_\infty := \Theta_\infty^{(l)}$ by

$$\begin{aligned} \Theta_\infty^{(1)}(x, x') &= \Sigma^{(1)}(x, x') \\ \Theta_\infty^{(i+1)}(x, x') &= \Theta_\infty^{(i)}(x, x')\dot{\Sigma}^{(i+1)}(x, x') + \Sigma^{(i+1)}(x, x'), \end{aligned}$$

where

$$\dot{\Sigma}^{(i+1)}(x, x') := \mathop{\mathbb{E}}_{f \sim \mathcal{N}(0, \Sigma^{(l)})} \Big[\sigma'\big(f(x)\big)\sigma'\big(f(x')\big)\Big],$$

taking the expectation with respect to a centered Gaussian process f of covariance $\Sigma^{(i)}$.

A proof via induction over the number of layers is given in [8]. Note that Θ_∞ depends only on the choice of σ, the number of layers, and the variance of parameters at initialization. As a consequence, the limiting kernel Θ_∞ is deterministic, despite the random initialization of w.

The theorem states that, for any initialization $w_0 \in \mathbb{R}^n$ and any $x, x' \in \mathbb{R}^{n_0}$, it holds that

$$\forall \epsilon > 0 : \lim_{n_{l-1} \to \infty} \cdots \lim_{n_1 \to \infty} \mathbb{P}\Big(\big|(\Theta_{w_0})_{i,i}(x, x') - \Theta_\infty(x, x')\big| > \epsilon\Big) = 0, \quad i = 1, \dots, n_l.$$

The second key result in [8] is that the neural tangent kernel stays asymptotically constant during training. To formalize this, we have to provide a slightly more general definition of training. More precisely, in the infinite width limit, the parameter space is infinite-dimensional, such that speaking of parameter training is nonsense. Thus, instead of denoting objects in dependence of the parameters $w(t)$, we use a notation in dependence of time t. In contrast to lemma 3.25, where a descent direction was denoted as $d|_{f_{w(t)}} \in \mathcal{F}$, such that

$$\partial_t w_i(t) = -\big\langle d|_{f_{w(t)}}, \partial_{w_i} f_{w(t)} \big\rangle_{\mathcal{D}_x},$$

we denote a descent direction as $d|_{f_t} \in \mathcal{F}$, such that

$$\partial_t w_i(t) = -\big\langle d|_{f_t}, \partial_{w_k} f_t \big\rangle_{\mathcal{D}_x}.$$

Furthermore, we denote the neural tangent kernel at time $t \in [0, \infty)$ as Θ_t, instead of $\Theta_{w(t)}$. Based on this, we can formalize the asymptotic behavior of the neural tangent kernel.

Theorem 3.29. *Let $f : \mathbb{R}^{n_0} \to \mathbb{R}^{n_l}$ be a neural network of depth l, with a twice differentiable non-linear activation function $\sigma : \mathbb{R} \to \mathbb{R}$ at each layer, whose second derivative is bounded. Then, for any $T > 0$, such that in the limit as the layer widths $n_1, \ldots, n_{l-1} \to \infty$ sequentially, the integral $\int_0^T \|d_t\|_{\mathcal{D}_x} dt$ stays stochastically bounded, we have uniformly for $t \in [0, T]$, that*

$$\Theta_t \to \Theta_\infty \otimes I_{n_l}.$$

As a consequence, in this limit, the dynamics of f_t are described by the differential equation

$$\partial_t f_t = \Phi_{\Theta_\infty \otimes I_{n_l}} \Big(\big\langle d|_{f_t}, \cdot \big\rangle_{\mathcal{D}_x} \Big).$$

A proof via induction over the number of layers is given in [8].

The theorem states that, for any time $t \in [0, T]$ and any $x, x' \in \mathbb{R}^{n_0}$ it holds

$$\lim_{n_{l-1} \to \infty} \cdots \lim_{n_1 \to \infty} \Big| \big(\Theta_{w(t)}\big)_{i,j}(x, x') - \big(\Theta_{w(0)}\big)_{i,j}(x, x') \Big| = 0, \quad i, j = 1, \ldots, n_l.$$

Thus, in the infinite width limit, the neural tangent kernel stays asymptotically constant during training. As discussed in section 3.4, for a lower bounded and convex cost C, the convergence of kernel gradient descent with respect to the neural tangent kernel to a global minimum of C is guaranteed, if $\Theta_\infty \otimes I_{n_l}$ is positive definite with respect to $\mathcal{D}_x$.

This concludes the necessary theory on the neural tangent kernel. Next, we will use its properties to motivate our approach of training deep neural networks in tiny subspaces.

4 Training in Low-Dimensional Subspaces

This section analyzes the training dynamics of deep neural networks, especially the occurring parameter training sequences. If not otherwise specified, the following work is based on [1], in which the hypothesis that deep neural networks can be trained in low-dimensional subspaces is raised. We start by giving an illustration of this hypothesis, followed by theoretical considerations on the existence of low-dimensional optimization trajectories. Finally, this leads to an algorithm for subspace extraction, which offers the possibility to train deep neural networks in subspaces so small, that second-order methods become applicable.

4.1 Motivation

In the following, we denote a deep neural network's parameter training sequence after $t \in \mathbb{N}$ iterations as $(w_k)_{k=0}^t$ in $\mathbb{R}^n$. Such a sequence naturally arises during training of the neural network via some reasonable iterative optimization method, such as stochastic gradient descent.

In [1], it is argued that based on the nested construction of deep neural networks and the training of networks via error backpropagation, there has to be a lot of dependence and redundancy within the parameters of a deep neural network. This leads to the low-dimensional trajectory hypothesis, claiming that there exists a low-dimensional affine set that approximately covers the optimization trajectory $(w_k)_{k=0}^t$. An illustration of this hypothesis is given in figure 4, which was originally published in [1] as figure 1.

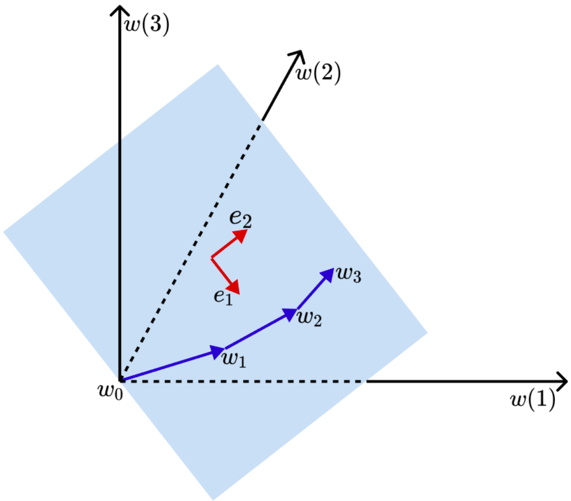


Figure 4: Low-dimensional optimization trajectory

Figure 4 visualizes a three-dimensional space containing three variables $w(1)$, $w(2)$, $w(3)$. These refer to the optimizable parameters of a very basic neural network. By training this network for three iterations, one achieves an optimization trajectory $(w_k)_{k=0}^3$, that could be embedded in a two-dimensional hyperplane, spanned by the orthogonal vectors e_1 and e_2.

Theoretically, this basic idea could translate to higher-dimensional models and more complex affine sets as well. Note that none of the optimizable variables could be left out to reduce the dimension, but rather one has to construct new independent variables to obtain the lower-dimensional affine set that approximately contains the optimization trajectory.

If the low-dimensional trajectory hypothesis holds true, it would provide a new understanding of neural networks, both from a theoretical and practical perspective. According to [1], this includes generalization capability, implicit regularization, efficient learning algorithms, and much more. Furthermore, there would be benefits like the possibility to use second-order methods for optimization in case the optimization trajectory's dimension is small enough.

We start by investigating the low-dimensional trajectory hypothesis from a theoretical perspective. This is based on the following background knowledge on singular value decomposition (SVD) and low-rank approximations.

4.2 Singular Value Decomposition

The following result is denoted and proven as theorem 2.5.2 in [13].

Theorem 4.1 (Singular Value Decomposition)**.** *For every matrix $A \in \mathbb{R}^{m\times n}$, there exist orthogonal matrices $U \in \mathbb{R}^{m\times m}$ and $V \in \mathbb{R}^{n\times n}$ and a diagonal matrix*

$$\Sigma \coloneqq diag(\sigma_1, \dots, \sigma_p) \in \mathbb{R}^{m\times n}$$

with $p \coloneqq min\{m, n\}$, and singular values $\sigma_1 \geqslant \sigma_2 \geqslant \dots \geqslant \sigma_p \geqslant 0$, such that

$$A = U\Sigma V^\mathsf{T}.$$

Note that orthogonal matrices refer to real square matrices, whose columns are orthogonal unit vectors. Some authors call these kinds of matrices orthonormal and do not demand the unity of columns for a matrix to be orthogonal. However, we will stick to the notation in [13]. In other words, $Q \in \mathbb{R}^{n\times n}$ is called an orthogonal matrix, if and only if

$$Q^\mathsf{T}Q = I_n = QQ^\mathsf{T}.$$

The SVD decomposes any real-valued matrix into a product of three special matrices. This representation yields the singular values which, like eigenvalues, characterize important properties of a matrix. For the further course, we need the following definition.

Definition 4.2. *The columns of the matrix $U = [u_1, \dots, u_m]$ are referred to as left singular vectors and the columns of the matrix $V = [v_1, \dots, v_n]$ are referred to as right singular vectors.*

Theorem 4.1 shows that the sequence of singular values is unique since they are sorted in descending order. However, the matrices U and V do not have to be unique. For example, the corresponding singular vectors can be interchanged if there are two identical singular values, such that a matrix can have several singular value decompositions.

As a simple implication of theorem 4.1, we can denote the following corollary to establish a useful connection between the left and right singular vectors.

Corollary 4.3. *Let $A \in \mathbb{R}^{m \times n}$ be a matrix with singular value decomposition $A = U\Sigma V^\mathsf{T}$, such that $p \coloneqq min\{m, n\}$ and $\sigma_1 \geqslant \cdots \geqslant \sigma_r > \sigma_{r+1} = \cdots = \sigma_p = 0$ for $r \leqslant p$.*

1. *Denoting the columns of U and V with u_i and v_i, we have*

$$Av_i = \sigma_i u_i, \quad A^\mathsf{T} u_i = \sigma_i v_i, \quad i = 1, \ldots, p.$$

2. *The squares of the singular values $\sigma_1^2, \ldots, \sigma_r^2$ are the eigenvalues of $A^\mathsf{T} A$ to the corresponding eigenvectors $v_1, \ldots, v_r$.*

Proof.

1. Using $A = U\Sigma V^\mathsf{T}$ and the orthogonality of V, we derive

$$Av_i = U\Sigma V^\mathsf{T} v_i = U\Sigma e_i = U\sigma_i e_i = \sigma_i u_i$$

for $i = 1, \ldots, p$. Analogously, we conclude the second statement

$$A^\mathsf{T} u_i = V\Sigma U^\mathsf{T} u_i = V\Sigma e_i = V\sigma_i e_i = \sigma_i v_i.$$

2. Since $A^\mathsf{T} A$ is symmetric, its spectral decomposition is given by

$$A^\mathsf{T} A = V\Sigma^\mathsf{T} U^\mathsf{T} U\Sigma V^\mathsf{T} = V\Sigma^\mathsf{T}\Sigma V^\mathsf{T} = V\Sigma^\mathsf{T}\Sigma V^{-1}.$$

Using $\Sigma^\mathsf{T}\Sigma = \operatorname{diag}(\sigma_1^2, \ldots, \sigma_r^2, 0, \ldots, 0) \in \mathbb{R}^{n \times n}$, we conclude the corollary. □

The SVD is an excellent tool to find low-rank approximations of a matrix $A \in \mathbb{R}^{m \times n}$, that is to find a matrix $B \in \mathbb{R}^{m \times n}$ with $\operatorname{rank}(B) < \operatorname{rank}(A)$ such that $\|A - B\|$ is small for some given matrix norm. We can generate such a low-rank approximation by simply reducing the number of singular values in the singular value decomposition.

Definition 4.4. *Let $A \in \mathbb{R}^{m \times n}$ be a matrix with singular value decomposition $A = U\Sigma V^\mathsf{T}$. For $d < r = rank(A)$ we define an approximation of A as*

$$A_d \coloneqq \sum_{i=1}^{d} u_i \sigma_i v_i^\mathsf{T} = U_d \Sigma_d V_d^\mathsf{T},$$

where $U_d \coloneqq [u_1, \ldots, u_d]$, $V_d \coloneqq [v_1, \ldots, v_d]$, and $\Sigma_d \coloneqq diag(\sigma_1, \ldots, \sigma_d)$.

In terms of the spectral norm $\| \cdot \|_2$, the matrix A_d is the best possible rank-d approximation. The underlying result is the following, which is denoted and proven as theorem 2.5.3 in [13].

Theorem 4.5 (Eckart-Young-Mirsky)**.** *Let $A \in \mathbb{R}^{m \times n}$ be a matrix with singular value decomposition $A = U\Sigma V^\mathsf{T}$. For every $d < r = rank(A)$ it holds, that*

$$\min_{rank(B)=d} \|A - B\|_2 = \|A - A_d\|_2 = \sigma_{d+1}.$$

This concludes the necessary background knowledge on singular value decompositions and low-rank approximations, which will play an important in subspace extraction.

4.3 Low-Dimensional Trajectory Hypothesis

To provide a mathematical formulation of the low-dimensional trajectory hypothesis, we need the concept of orthogonal projections. The following result is based on section 2.6.1 in [13].

Lemma 4.6. *Let $S \subseteq \mathbb{R}^n$ be a subspace. There exists a unique matrix $P \in \mathbb{R}^{n\times n}$, such that*

$$\mathit{Im}(P) = S \ \wedge \ P^2 = P = P^\mathsf{T}.$$

The linear map $\tilde{P} : \mathbb{R}^n \to \mathbb{R}^n : x \mapsto Px$ is called the orthogonal projection onto S.

Proof. To prove the existence of an orthogonal projection, we let $d := \dim(S)$ denote the dimension of S and $Q := [q_1, \ldots, q_d] \in \mathbb{R}^{n\times d}$ denote an orthonormal basis of S. We define

$$P := QQ^\mathsf{T} \in \mathbb{R}^{n\times n}$$

and observe by the orthonormality of Q, that $Q^\mathsf{T} Q = I_d$. Thus P is idempotent, this is

$$P^2 = QQ^\mathsf{T} QQ^\mathsf{T} = QI_dQ^\mathsf{T} = QQ^\mathsf{T} = P.$$

Furthermore, P is symmetric since

$$P^\mathsf{T} = (QQ^\mathsf{T})^\mathsf{T} = QQ^\mathsf{T} = P.$$

By the orthonormality of Q, for arbitrary $s \in S$ there exists $\alpha = (\alpha_1, \ldots, \alpha_d)^\mathsf{T} \in \mathbb{R}^d$ with

$$s = \alpha_1 q_1 + \cdots + \alpha_d q_d = Q\alpha.$$

Based on this, we derive $S \subseteq \mathrm{Im}(P)$ since

$$Ps = QQ^\mathsf{T} s = QQ^\mathsf{T} Q\alpha = QI_d\alpha = Q\alpha = s.$$

Additionally, $\mathrm{Im}(P) \subseteq S$ due to the fact that, for $x \in \mathbb{R}^n$, we have $\tilde{\alpha} := Q^\mathsf{T} x \in \mathbb{R}^d$, so

$$Px = QQ^\mathsf{T} x = Q\tilde{\alpha} \in S.$$

Thus $\mathrm{Im}(P) = S$. This proves the existence of an orthogonal projection P onto S. To prove the uniqueness of P, we should first note that based on $P^2 = P = P^\mathsf{T}$ we have

$$\forall x, y \in \mathbb{R}^n : (Px)^\mathsf{T}(y - Py) = x^\mathsf{T} P^\mathsf{T} y - x^\mathsf{T} P^\mathsf{T} Py = x^\mathsf{T} Py - x^\mathsf{T} Py = 0.$$

Using $Px \in S$ for every $x \in \mathbb{R}^n$, we can conclude that

$$\forall y \in \mathbb{R}^n : y - Py \in S^\perp := \{z \in \mathbb{R}^n \mid \forall s \in S : z^\mathsf{T} s = 0\}.$$

Next, we let P_1 and P_2 denote orthogonal projections onto S, then for any $z \in \mathbb{R}^n$ we derive

$$\begin{aligned}\|(P_1 - P_2)z\|_2^2 &= \|P_1 z - P_2 z\|_2^2 \\ &= z^\mathsf{T} P_1^\mathsf{T} P_1 z - z^\mathsf{T} P_2^\mathsf{T} P_1 z - z^\mathsf{T} P_1^\mathsf{T} P_2 z + z^\mathsf{T} P_2^\mathsf{T} P_2 z \\ &= z^\mathsf{T} P_1 z - z^\mathsf{T} P_2^\mathsf{T} P_1 z - z^\mathsf{T} P_1^\mathsf{T} P_2 z + z^\mathsf{T} P_2 z,\end{aligned}$$

where we have used the property $P_i^2 = P_i = P_i^\mathsf{T}$ for $i = 1, 2$. Since each term on the right-hand side is a real number, we can transpose each term to derive

$$\begin{aligned}\left\|(P_1 - P_2)z\right\|_2^2 &= z^\mathsf{T} P_1 z - z^\mathsf{T} P_2^\mathsf{T} P_1 z - z^\mathsf{T} P_1^\mathsf{T} P_2 z + z^\mathsf{T} P_2 z\\ &= (P_1 z)^\mathsf{T} z - (P_1 z)^\mathsf{T} P_2 z - (P_2 z)^\mathsf{T} P_1 z + (P_2 z)^\mathsf{T} z\\ &= (P_1 z)^\mathsf{T}(z - P_2 z) + (P_2 z)^\mathsf{T}(z - P_1 z).\end{aligned}$$

Using $\mathrm{Im}(P_1) = S = \mathrm{Im}(P_2)$, we observe that

$$P_1 z \in S \ \wedge\ z - P_2 z \in S^\perp \ \wedge\ P_2 z \in S \ \wedge\ z - P_1 z \in S^\perp.$$

Thus the right-hand side of the equation is zero and by the positive definiteness of the norm, we conclude $P_1 - P_2 = 0$. This implies $P_1 = P_2$, showing the uniqueness of P. □

Based on this we can formulate the low-dimensional trajectory hypothesis mathematically.

Low-Dimensional Trajectory Hypothesis. *Given a training trajectory $(w_k)_{k=0}^t$ in $\mathbb{R}^n$, there exists $d \ll n$ and a d-dimensional subspace $S \subset \mathbb{R}^n$ with an orthonormal basis $Q \in \mathbb{R}^{n\times d}$, such that there is a low-dimensional trajectory $(\alpha_k)_{k=0}^t$ in $\mathbb{R}^d$, satisfying*

$$w_{k+1} - w_k \approx Q(\alpha_{k+1} - \alpha_k), \quad k = 0, \ldots, t-1.$$

The hypothesis can be explained as follows. Recalling the illustration in figure 4, the low-dimensional trajectory hypothesis claims the existence of a d-dimensional affine set, that approximately covers the parameter trajectory $(w_k)_{k=0}^t$ in $\mathbb{R}^n$. More precisely, it claims the existence of some d-dimensional subspace $S \subset \mathbb{R}^n$ and some $x \in \mathbb{R}^n$, defining the affine set

$$x + S \coloneqq \left\{y \in \mathbb{R}^n \mid x - y \in S\right\},$$

such that the distance between $x + S$ and any of the parameters w_k is very small, that is

$$\exists s_k \in x + S : \left\|w_k - s_k\right\|_2 \approx 0, \quad k = 0, \ldots, t.$$

By lemma 4.6, there exists a unique orthogonal projection $P \in \mathbb{R}^{n\times n}$ onto the subspace, so

$$w_{k+1} - w_k \approx P(w_{k+1} - w_k), \quad k = 0, \ldots, t-1.$$

Thus, we can define the projected low-dimensional trajectory $(s_k)_{k=0}^t \coloneqq (Pw_k)_{k=0}^t$, satisfying

$$w_{k+1} - w_k \approx P(w_{k+1} - w_k) = s_{k+1} - s_k, \quad k = 0, \ldots, t-1.$$

Given an orthonormal basis $Q \in \mathbb{R}^{n\times d}$ of S, each element of the low-dimensional trajectory can be expressed in terms of its coefficients with regard to the basis Q, that is

$$\exists \alpha_k \in \mathbb{R}^d : s_k = V\alpha_k, \quad k = 0, \ldots, t.$$

As a consequence, the low-dimensional trajectory hypothesis claims, that

$$w_{k+1} - w_k \approx s_{k+1} - s_k = Q(\alpha_{k+1} - \alpha_k), \quad k = 0, \ldots, t-1.$$

To motivate the low-dimensional trajectory hypothesis from a theoretical perspective, we investigate a single output neural network

$$f : \mathbb{R}^{n_0} \times \mathbb{R}^n \to \mathbb{R} : (x, w) \mapsto f(x, w).$$

As seen in section 2.2 the neural network architecture f induces a function space

$$\mathcal{F}|_f := \Big\{ f_w : \mathbb{R}^{n_0} \to \mathbb{R} : x \mapsto f(x, w) \mid w \in \mathbb{R}^n \Big\}.$$

In order to train the neural network, we let

$$\mathcal{X} = \big\{ (x_i, y_i) \mid i = 1, \ldots, m \big\} \subset \mathbb{R}^{n_0} \times \mathbb{R}$$

denote a training set of size $m \in \mathbb{N}$. Recall that training of f refers to finding an optimal parameter vector w, which is accomplished by minimizing the empirical error function

$$E : \mathbb{R}^n \to \mathbb{R}_+ : w \mapsto \sum_{i=1}^{m} \ell\big(f(x_i, w), y_i\big),$$

where ℓ is a loss function of the form

$$\ell : \mathbb{R} \times \mathbb{R} \to \mathbb{R}_+ : (x, y) \mapsto \ell(x, y).$$

Note that in comparison to the formulation given in section 2.2, we removed the constant factor $1/m$ from the front of the sum, which has no influence on the minimum. Furthermore, we chose $\lambda = 0$, which means we do not penalize the complexity of w.

Under the assumption that all activation functions in f are differentiable, thus f is differentiable itself, the choice of a differentiable loss function ℓ enables us to minimize E via gradient descent. Recall that, based on definition 3.22, we can analyze the training dynamics with a continuous function $w : [0, \infty) \to \mathbb{R}^n : t \mapsto w_t$, that follows the gradient descent on E. To clarify this, since gradient descent performs parameter updates with regard to the rule

$$w_{k+1} \leftarrow w_k - \eta \cdot \nabla E(w_k),$$

we can let $\eta \to 0$, to derive the equation

$$\lim_{\eta \to 0} \frac{w_{k+1} - w_k}{\eta} = -\nabla E(w_k).$$

As a consequence, the training dynamics are governed by the differential equation

$$\partial_t w_t = -\nabla E(w_t).$$

A basic application of the chain rule yields

$$\partial_t w_t = -\nabla E(w_t) = \sum_{i=1}^{m} \nabla_w f(x_i, w_t) \cdot \partial_x \ell\big(f(x_i, w_t), y_i\big),$$

where $\partial_x \ell$ denotes the partial derivative of the loss function with respect to the first argument. Note that we are still in the one-dimensional setting, such that $\partial_x \ell$ can equivalently be interpreted as the gradient $\nabla_x \ell$. To explain the main idea of dimensionality reduction, we have to present this gradient flow in terms of vectors and matrices. Therefore, we let

$$\nabla_w f(\mathcal{X}, w_t)^\mathsf{T} \coloneqq \Big[\nabla_w f(x_1, w_t), \ldots, \nabla_w f(x_m, w_t)\Big] \in \mathbb{R}^{n \times m}$$

denote the gradients of f at w_t on the training data set $\mathcal{X}$ and

$$\nabla_{f(\mathcal{X}, w_t)} E \coloneqq \Big[\nabla_x \ell\big(f(x_1, w_t), y_1\big), \ldots, \nabla_x \ell\big(f(x_m, w_t), y_m\big)\Big]^\mathsf{T} \in \mathbb{R}^m$$

denote the gradients of ℓ with respect to the first argument. This yields the gradient flow

$$\partial_t w_t = -\nabla_w f(\mathcal{X}, w_t)^\mathsf{T} \cdot \nabla_{f(\mathcal{X}, w_t)} E.$$

Based on this differential equation we can use the previously seen properties of the neural tangent kernel to motivate the existence of low-dimensional optimization trajectories and to explain the approach of dimensionality reduction. For this, we denote the parallelized outputs

$$\tilde{f} : \mathbb{R}^n \to \mathbb{R}^m : w \mapsto \Big[f(x_1, w), \ldots, f(x_m, w)\Big]^\mathsf{T},$$

such that for given $w \in \mathbb{R}^n$, by definition 3.27 the neural tangent kernel translates to

$$\Theta_w = \sum_{i=1}^{n} \partial_{w_i} \tilde{f}(w) \otimes \partial_{w_i} \tilde{f}(w) = \nabla_w f(\mathcal{X}, w) \cdot \nabla_w f(\mathcal{X}, w)^\mathsf{T}.$$

In the setting of theorem 3.29, we have seen that, in the infinite width limit, the neural tangent kernel converges to a deterministic limiting kernel. Thus, in this limit, we have

$$\nabla_w f(\mathcal{X}, w_t) \cdot \nabla_w f(\mathcal{X}, w_t)^\mathsf{T} = \Theta_t = \Theta_0 = \nabla_w f(\mathcal{X}, w_0) \cdot \nabla_w f(\mathcal{X}, w_0)^\mathsf{T},$$

such that, ideally, the dynamics of gradient flow are governed by

$$\partial_t w_t = -\nabla_w f(\mathcal{X}, w_0)^\mathsf{T} \cdot \nabla_{f(\mathcal{X}, w_t)} E,$$

where the first factor is a constant matrix and the second factor changes over time t. Thus, if the neural tangent kernel Θ_0 is low-rank, $\nabla_w f(\mathcal{X}, w_0)$ could be approximated by a low-rank matrix, such that the optimization trajectory would be embedded in a low-dimensional affine set. To clarify this, we apply singular value decomposition on $\nabla_w f(\mathcal{X}, w_0)$, such that

$$\nabla_w f(\mathcal{X}, w_0) = U_0 \Sigma_0 V_0^\mathsf{T},$$

where $U_0 \in \mathbb{R}^{m \times m}$ and $V_0 \in \mathbb{R}^{n \times n}$ are orthogonal and $\Sigma_0 = \operatorname{diag}(\sigma_1, \ldots, \sigma_m)$ is positive semidefinite. Based on this, the neural tangent kernel can be rewritten as

$$\Theta_0 = \nabla_w f(\mathcal{X}, w_0) \cdot \nabla_w f(\mathcal{X}, w_0)^\mathsf{T} = U_0 \Sigma_0 \Sigma_0^\mathsf{T} U_0^\mathsf{T},$$

which gives the spectral decomposition of Θ_0. More precisely, we could define

$$\Sigma^{\text{NTK}} \coloneqq \Sigma_0 \Sigma_0^\mathsf{T} = \operatorname{diag}(\sigma_1^2, \ldots, \sigma_m^2),$$

such that the eigenvalues of Θ_0 are given by $\lambda_i = \sigma_i^2$ for $i = 1, \ldots, m$.

If Θ_0 is low-rank, we could approximate Σ_0 by a low-rank matrix $\tilde{\Sigma}_0 = \operatorname{diag}(\sigma_1, \ldots, \sigma_d)$, so

$$\Sigma_0 \approx \tilde{U}_0 \tilde{\Sigma}_0 \tilde{V}_0^\mathsf{T},$$

where $\tilde{U}_0 \in \mathbb{R}^{m \times d}$ contains the first d columns of an $m \times m$ identity matrix and $\tilde{V}_0 \in \mathbb{R}^{n \times d}$ contains the first d columns of an $n \times n$ identity matrix. Using this approximation, we have

$$\nabla_w f(\mathcal{X}, w_0) = U_0 \Sigma_0 V_0^\mathsf{T} \approx U_0 \tilde{U}_0 \tilde{\Sigma}_0 \tilde{V}_0^\mathsf{T} V_0^\mathsf{T}.$$

This is exactly the rank-d approximation from definition 4.4. As a consequence, the dynamics of gradient flow are approximately governed by

$$\partial_t w_t = -\nabla_w f(\mathcal{X}, w_0)^\mathsf{T} \cdot \nabla_{f(\mathcal{X}, w_t)} E \approx -V_0 \tilde{V}_0 \Big(\tilde{\Sigma}_0 \tilde{U}_0^\mathsf{T} U_0^\mathsf{T} \cdot \nabla_{f(\mathcal{X}, w_t)} E \Big). \qquad (*)$$

This gives rise to the low-dimensional trajectory hypothesis. The term $V_0 \tilde{V}_0 \in \mathbb{R}^{n \times d}$ refers to the orthonormal basis Q of the d-dimensional subspace S and the remaining term in parentheses describes the projected gradient in terms of its coefficients with regard to the basis Q of S.

By the Eckart-Young-Mirsky theorem 4.5, the used rank-d approximation in $(*)$ is the best possible and the approximation error in terms of the spectral norm is given by

$$\left\| \nabla_w f(\mathcal{X}, w_0) - U_0 \tilde{U}_0 \tilde{\Sigma}_0 \tilde{V}_0^\mathsf{T} V_0^\mathsf{T} \right\|_2 = \sigma_{d+1}.$$

The approximation error in $(*)$ is therefore determined by the singular value decay of the matrix $\nabla_w f(\mathcal{X}, w_0)$, or equivalently, the eigenvalue decay of the neural tangent kernel Θ_0. An extensive analysis of this eigenvalue decay can be found in [14] and [15]. These papers show that in the infinite width limit, the NTK is indeed low-rank, which theoretically motivates the claim of the low-dimensional trajectory hypothesis.

From a practical perspective, however, the previous discussion is too theoretical, since it holds true only in the infinite width limit. Of course, this pre-condition is never given in practical applications. This is why section 5 contains various numerical experiments, that investigate the low-dimensional trajectory with regard to deep neural networks of finite width.

4.4 Dynamic Linear Dimensionality Reduction

For the practical investigation of the low-dimensional trajectory hypothesis, we have to provide a method to efficiently extract the low-dimensional optimization trajectory. The key issue here consists in finding a suitable dimension $d \in \mathbb{N}$ and an orthonormal basis Q of the d-dimensional subspace $S \subset \mathbb{R}^n$ whose translation approximately covers the optimization trajectory. This problem will be addressed via principal component analysis (PCA), which operates based on the following definition.

Definition 4.7. *For $d \in \mathbb{N}$, we define the set of rank-d orthogonal projection matrices*

$$\mathcal{P}_d \coloneqq \Big\{ P \in \mathbb{R}^{n \times n} \mid \mathit{rank}(P) = d \ \wedge \ P^2 = P = P^\mathsf{T} \Big\}.$$

In the following, we fix $d \leqslant n$ and let $W = [w_1, \dots, w_t] \in \mathbb{R}^{n \times t}$ be a mean-centered data matrix, that is $\sum_{k=1}^{t} w_k = 0$. Principal component analysis (PCA) consists of finding the orthogonal projection matrix $P^* \in \mathcal{P}_d$, that minimizes the reconstruction error

$$\mathcal{P}_d \to \mathbb{R} : P \mapsto \sum_{k=1}^{t} \|Pw_k - w_k\|_2^2.$$

Equivalently, PCA can be formulated as finding the solution $P^* \in \mathcal{P}_d$, such that

$$P^* := \operatorname*{arg\,min}_{P \in \mathcal{P}_d} \|PW - W\|_F^2,$$

where $\|\cdot\|_F$ denotes the Frobenius norm. Casually speaking, the principal component analysis aims to project the original data to a lower-dimensional space while preserving as much information as possible. The following theorem provides a method to solve the PCA problem via singular value decomposition. It can be found as theorem 15.1 in [16].

Theorem 4.8. *The PCA solution $P^* \in \mathcal{P}_d$ can be decomposed as*

$$P^* = U_d U_d^\mathsf{T},$$

where $U_d \in \mathbb{R}^{n \times d}$ is the matrix formed by the first d left singular vectors of W.

Proof. Let $P \in \mathcal{P}_d$ be arbitrary. By the linearity of the trace, we observe that

$$\begin{aligned} \|PW - W\|_F^2 &= \operatorname{Tr}\big[(PW - W)^\mathsf{T}(PW - W)\big] \\ &= \operatorname{Tr}\big[W^\mathsf{T} P^2 W - 2W^\mathsf{T} PW + W^\mathsf{T} W\big] \\ &= \operatorname{Tr}\big[W^\mathsf{T} W\big] - \operatorname{Tr}\big[W^\mathsf{T} PW\big], \end{aligned}$$

since $P^\mathsf{T} = P = P^2$ by definition. Using that $\operatorname{Tr}\big[W^\mathsf{T} W\big]$ is independent of P, we derive

$$\operatorname*{arg\,min}_{P \in \mathcal{P}_d} = \|PW - W\|_F^2 = \operatorname*{arg\,max}_{P \in \mathcal{P}_d} \operatorname{Tr}\big[W^\mathsf{T} PW\big].$$

By the proof of lemma 4.6, there exists an orthonormal basis $Q = [q_1, \dots, q_d] \in \mathbb{R}^{n \times d}$ of $\mathbb{R}^d$, such that P can be decomposed as $P = QQ^\mathsf{T}$. Using the invariance of the trace under cyclic permutation and the orthogonality of Q, we observe that

$$\operatorname{Tr}\big[W^\mathsf{T} PW\big] = \operatorname{Tr}\big[W^\mathsf{T} QQ^\mathsf{T} W\big] = \operatorname{Tr}\big[Q^\mathsf{T} WW^\mathsf{T} Q\big] = \sum_{i=1}^{d} q_i^\mathsf{T} WW^\mathsf{T} q_i.$$

To maximize the rightmost expression, we perform SVD on $W = U\Sigma V^\mathsf{T}$, such that

$$WW^\mathsf{T} = U\Sigma V^\mathsf{T} V \Sigma^\mathsf{T} U^\mathsf{T} = U\Sigma\Sigma^\mathsf{T} U^\mathsf{T}.$$

Using the notation $Z = U^\mathsf{T} QQ^\mathsf{T} U$, we derive by the invariance under cyclic permutation, that

$$\operatorname{Tr}\big[Q^\mathsf{T} WW^\mathsf{T} Q\big] = \operatorname{Tr}\big[Q^\mathsf{T} U\Sigma\Sigma^\mathsf{T} U^\mathsf{T} Q\big] = \operatorname{Tr}\big[U^\mathsf{T} QQ^\mathsf{T} U\Sigma\Sigma^\mathsf{T}\big] = \operatorname{Tr}\big[Z\Sigma\Sigma^\mathsf{T}\big].$$

Furthermore, we can conclude that $|Z_{i,j}| \leqslant 1$ for $i, j = 1, \dots, n$, since Z is orthogonal as a product of four orthogonal matrices. This implies the inequality

$$\mathrm{Tr}\big[W^\mathsf{T} P W\big] = \mathrm{Tr}\big[Z\Sigma\Sigma^\mathsf{T}\big] = \sum_{i=1}^{d} z_{i,i}\sigma_i^2 \leqslant \sum_{i=1}^{d} \sigma_i^2.$$

The upper bound is attained in the case where $Q = U_d = [u_1, \dots, u_d]$ contains the first d columns of U. This follows from corollary 4.3, namely the identity $W^\mathsf{T} u_i = \sigma_i v_i$, providing

$$\mathrm{Tr}\big[W^\mathsf{T} P W\big] = \sum_{i=1}^{d} q_i^\mathsf{T} W W^\mathsf{T} q_i = \sum_{i=1}^{d} u_i^\mathsf{T} W W^\mathsf{T} u_i = \sum_{i=1}^{d} u_i^\mathsf{T} W v_i \sigma_i = \sum_{i=1}^{d} \sigma_i^2.$$

As a consequence, the PCA solution is given by $P^* = U_d U_d^\mathsf{T}$. □

In order to extract the low-dimensional optimization trajectory, it is sufficient to determine the d-dimensional subspace $S \subset \mathbb{R}^n$. This can be effectively achieved by the procedure provided in the proof of theorem 4.8. The idea is to sample $t \geqslant d$ parameter vectors along the optimization trajectory and to determine the orthogonal projection P onto S by PCA. By lemma 4.6 this also provides an orthonormal basis of the lower-dimensional subspace.

Sampling the parameters $\{w_0, \dots, w_t\}$ can be achieved by training the network via some reasonable iterative optimization method such as SGD. To be precise, in the following we understand these $t + 1$ parameter vectors as t samples only since the initialization w_0 is chosen randomly. The application of PCA to this data requires centralization. Hence we let

$$\overline{w} := \frac{1}{t+1} \sum_{i=0}^{t} w_i$$

denote the mean of the samples and define the mean-centered data matrix

$$W := \big[w_0 - \overline{w}, \dots, w_t - \overline{w}\big] \in \mathbb{R}^{n \times (t+1)}.$$

Recall that PCA consists of finding the orthogonal projection matrix $P^* \in \mathcal{P}_d$ defined as

$$P^* := \operatorname*{arg\,min}_{P \in \mathcal{P}_d} \big\|PW - W\big\|_F^2.$$

The construction of P^* in the proof of theorem 4.8 is straight-forward and also provides an orthonormal basis of the d-dimensional subspace S. At first, we need to determine the right singular vectors $v_1, \dots, v_d$ of W. Since $t \ll n$ we apply the second result of corollary 4.3 to efficiently compute $v_1, \dots, v_d$ by the spectral decomposition of $W^\mathsf{T} W \in \mathbb{R}^{(t+1)\times(t+1)}$. According to the first statement of corollary 4.3, these can then be used to derive

$$u_i = \frac{1}{\sigma_i} W v_i, \quad i = 1, \dots, d.$$

By theorem 4.8, the PCA solution is then given by $P^* = U_d U_d^\mathsf{T}$, where $U_d = [u_1, \dots, u_d]$ is an orthonormal basis of S, according to the proof of lemma 4.6. Furthermore, the corresponding orthogonal projection onto the d-dimensional subspace S is given by the map

$$P : \mathbb{R}^n \to \mathbb{R}^n : w \mapsto U_d U_d^\mathsf{T} w.$$

With regard to the illustration in figure 4, one could reverse the centralization to define

$$\tilde{P} : \mathbb{R}^n \to \mathbb{R}^n : w \mapsto \overline{w} + U_d U_d^\intercal w,$$

projecting the optimization trajectory onto the lower-dimensional affine set $\overline{w}+S$. Speaking in terms of the low-dimensional trajectory hypothesis, it claims that the optimization trajectory is nearly invariant under transformation with $\tilde{P}$. That is

$$\forall t > 0 : \tilde{P} w_t \approx w_t.$$

In summary, we denote the following algorithm for dimensionality reduction from [1].

Algorithm 7 Dynamic Linear Dimensionality Reduction (DLDR)

```
Input: Dimensionality of the subspace $d$, number of samples $t \geqslant d$
1: Sample parameter trajectory $(w_k)_{k=0}^t$ along the optimization trajectory;
2: Compute the mean $\overline{w} \coloneqq \frac{1}{t+1} \sum_{k=0}^t w_k$;
3: Centralize the samples as $W = [w_0 - \overline{w}, \dots, w_t - \overline{w}]$;
4: Perform spectral decomposition, such that $W^\intercal W = V\Sigma^2 V^{-1}$;
5: Obtain the $d$ largest eigenvalues $\left[\sigma_1^2, \dots, \sigma_d^2\right]$ with eigenvectors $\left[v_1, \dots, v_d\right]$;
6: Determine the singular vectors $u_i = \frac{1}{\sigma_i} W v_i$ for $i = 1, \dots, d$;
7: Return the orthonormal basis $\left[u_1, \dots, u_d\right]$ of the subspace;
```

Besides the parameter sampling in line 1, the complexity of the algorithm is determined by lines 5 and 7. In line 5 we calculate the matrix product $W^\intercal W$ which is of complexity $\mathcal{O}(t^2 n)$ and the complexity of the spectral decomposition is $\mathcal{O}(t^3)$, since $W^\intercal W \in \mathbb{R}^{(t+1)\times(t+1)}$. Furthermore, line 7 contains d products Wv_i which is of complexity $\mathcal{O}(dtn)$. Using $d \leqslant t$ the total complexity of lines 2 to 7 is $\mathcal{O}(t^3 + 2t^2 n)$. Since $t \ll n$ the time consumption is negligible in comparison to the complexity of training in line 1.

4.5 DLDR-based Training Algorithms

Assuming that the low-dimensional trajectory hypothesis holds true, not only in the infinite width limit but also for finite wide deep neural networks, we could adopt the training algorithms from section 2.2 to perform parameter updates in the lower-dimensional subspace.

Recall that all of the proposed training algorithms operate in a similar manner, which consists of iterative parameter updates. This is achieved by computing a descent direction $d_k \in \mathbb{R}^n$, choosing a learning rate $\eta > 0$, and updating the parameters according to the rule

$$w_{k+1} \leftarrow w_k - \eta \cdot d_k.$$

Note that the training algorithms differ only in determining the descent directions. The basic idea of a parameter update in the lower-dimensional subspace $S \subset \mathbb{R}^n$ at step $k \in \mathbb{N}$, is to project the parameters w_k and the descent direction d_k onto S and apply the update rule in the subspace S. Given an orthonormal basis Q of S, the update rule translates to

$$w_{k+1} \leftarrow Q\big(Q^\mathsf{T} w_k - \eta \cdot Q^\mathsf{T} d_k\big).$$

Using the fact, that S is a subspace, this procedure results in a parameter vector $w_{k+1} \in S$, such that an iterative application can be simplified to the update rule

$$w_{k+1} \leftarrow w_k - \eta \cdot QQ^\mathsf{T} d_k.$$

If the dimension of the subspace is small enough, training should happen a lot faster than in the original space. Also, since there are only a few variables to optimize, second-order methods become applicable. Another benefit is that training in the lower-dimensional space should prevent overfitting and improve robustness to label noises.

Based on the stochastic gradient descent algorithm, we can introduce its projected version, performing the parameter updates in a lower-dimensional space as follows.

Algorithm 8 Projected Stochastic Gradient Descent (P-SGD)

```
Input: Differentiable $E : \mathbb{R}^n \to \mathbb{R}$, batch size $b \in \mathbb{N}$, number of epochs $N \in \mathbb{N}$, $w_0 \in \mathbb{R}^n$,
       orthonormal basis $Q \in \mathbb{R}^{n \times d}$ of the $d$-dimensional subspace
Project $w_0 \leftarrow QQ^\mathsf{T} w_0$;
for $k = 0, \dots, \frac{Nm}{b}$ do:
    Draw a random subset $\mathcal{I}_k \subset \{1, \dots, m\}$ of size $|\mathcal{I}_k| = b$;
    Choose a step size $\eta_k > 0$;
    Set $w_{k+1} \leftarrow w_k - \eta_k \cdot QQ^\mathsf{T} \nabla E(w_k, \mathcal{I}_k)$;
end for
```

The convergence of projected stochastic gradient descent will be investigated in section 5.

If the dimension of the subspace covering the optimization trajectory is small enough, we could probably use second-order information for the training of deep neural networks. To clarify this, we have to recall that in Newton's method, the descent direction was defined as

$$d_k = -H_E^{-1}(x_k) \nabla E(x_k).$$

As discussed, for large n it is computationally impractical to determine the inverse Hessian. However, based on the low-dimensional trajectory hypothesis, for any x_k there could exist some lower-dimensional version of the Hessian, call it $H_0(x_k) \in \mathbb{R}^{d \times d}$, such that

$$H_E(x_k) \approx QH_0(x_k)Q^\mathsf{T}.$$

For small d we could effectively compute the inverse Hessian as

$$H_E^{-1}(x_k) \approx (QH_0(x_k)Q^\mathsf{T})^\dagger = QH_0^{-1}(x_k)Q^\mathsf{T},$$

where † denotes the Moore-Penrose inverse. Doing so would enable us to perform the Newton update in the lower-dimensional subspace via the update rule

$$w_{k+1} \leftarrow w_k - \eta_k \cdot Q\big(H_0^{-1}(x_k) Q^{\mathsf{T}} \nabla E(x_k)\big).$$

The problem with that approach, however, is that we are still confronted with the determination of H_0. To clarify this, although we only need to optimize d independent variables, this optimization is based on the gradients of parameters in the n-dimensional space. As a consequence, we still have to approximate the lower-dimensional inverse Hessians H_0^{-1}. One method for this purpose, namely BFGS was already introduced in section 2.2. Finally, we can denote the following algorithm, which is a slight modification of algorithm 2 in [1].

Algorithm 9 Projected Broyden-Fletcher-Goldfarb-Shanno (P-BFGS)

```
Input: Twice differentiable $E : \mathbb{R}^n \to \mathbb{R}$, batch size $b \in \mathbb{N}$, number of epochs $N \in \mathbb{N}$,
       $w_0 \in \mathbb{R}^n$, orthonormal basis $Q \in \mathbb{R}^{n \times d}$ of the $d$-dimensional subspace
Initialize $B_0 \leftarrow I_n$, $\mathcal{I}_0 \leftarrow \{1, \dots, b\}$ and project $w_0 \leftarrow QQ^{\mathsf{T}} w_0$;
for $k = 0, \dots, \frac{Nm}{b}$ do:
    Compute a step size $\eta_k > 0$ using backtracking line search;
    Set $w_{k+1} \leftarrow w_k - \eta_k \cdot QB_kQ^T \nabla E(w_k, \mathcal{I}_k)$;
    Draw a random subset $\mathcal{I}_{k+1} \subset \{1, \dots, m\}$ of size $|\mathcal{I}_k| = b$;
    Let $s_k = w_{k+1} - w_k$ and $y_k = \nabla E(w_{k+1}, \mathcal{I}_{k+1}) - \nabla E(w_k, \mathcal{I}_k)$;
    Let $\rho_k = (y_k^{\mathsf{T}} s_k)^{-1}$ and $V_k = I_n - \rho_k y_k s_k^{\mathsf{T}}$;
    Set $B_{k+1} \leftarrow V_k^{\mathsf{T}} B_k V_k + \rho_k s_k s_k^{\mathsf{T}}$;
end for
```

Note that the backtracking line search in here has to be performed with respect to the gradient in the lower-dimensional subspace. Denoting this lower-dimensional descent direction as

$$d_k \coloneqq Q^T \nabla E\big(w_k, \mathcal{I}_k\big),$$

the adopted version of backtracking line search iteratively searches for some $\eta_k > 0$ satisfying

$$E\big(w_k - \eta_k \cdot QB_k d_k, \mathcal{I}_k\big) \leqslant E\big(w_k, \mathcal{I}_k\big) - c \cdot \eta_k \cdot d_k^{\mathsf{T}} B_k d_k.$$

In this equation, we used the minor definition from section 2.2, namely

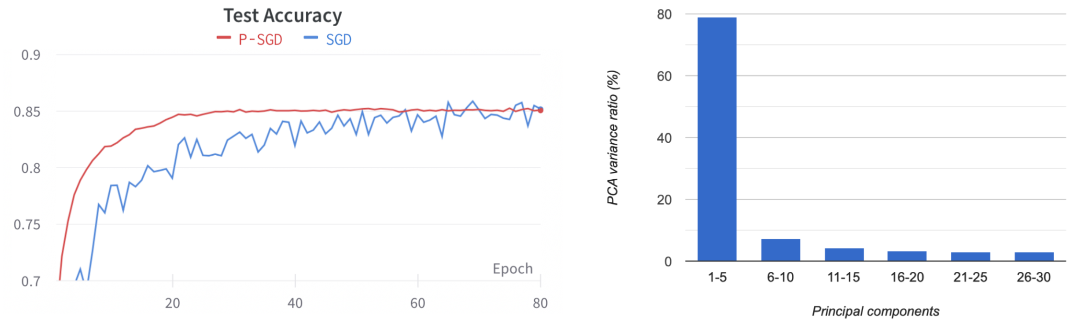
$$E(w_k, \mathcal{I}_k) \coloneqq \frac{1}{b} \sum_{i \in \mathcal{I}_k} \ell\big(f(x_i, w_k), y_i\big) + \lambda \|w_k\|_2^2.$$

According to [1], the step size choice via this adopted version of backtracking linesearch ensures $y_k^{\mathsf{T}} s_k > 0$, which guarantees the positive definiteness of the inverse Hessian approximation B_{k+1}. As a consequence, the proposed P-BFGS algorithm is well-defined. Similar to the choice in [1], we always use $c = 0.4$ and $\beta = 0.55$ to perform the backtracking line search.

5 Numerical Experiments

This section investigates the low-dimensional trajectory hypothesis from a practical perspective. We conduct extensive numerical experiments to study the performance of DLDR-based training algorithms, namely P-SGD and P-BFGS, on various architectures and data sets. In this context, we also analyze the computational complexity of the subspace extraction via DLDR. The experiments include a comparison of different sampling strategies and subspace dimensions. Finally, we propose a technique to effectively find an appropriate space size and use this to introduce a new training algorithm based on DLDR. All of the experiments are run on a NVIDIA A100 GPU using PyTorch.

As a warm-up, we start with a motivational experiment, indicating that the low-dimensional trajectory hypothesis could indeed hold true. We proceed as in section 3.3 of [1] and study the training of ResNet8 [17] on the CIFAR-10 data set. More precisely, we compare the performance of training in a low-dimensional subspace to the performance of training in the full parameter space of dimension 75,290. The data set is normalized by channel-wise mean and variance. Furthermore we apply horizontal flipping with probability of 0.5 as well as cropping and 4-pixel padding. At first we conduct the full parameter training via SGD with learning rate of 0.1, batch size of 128 and momentum of 0.9. During the first 30 epochs of that training, we save the network's parameter vector after each epoch, that is we generate a trajectory $(w_k)_{k=0}^{30}$ in $\mathbb{R}^{75,290}$. In a follow-up experiment, these samples are used to extract a low-dimensional subspace of dimension 15. Note that the choice of this specific dimension is arbitrary and was imitated from section 3.3 of [1]. To investigate the low-dimensional trajectory hypothesis from a practical perspective, we then train the same network from scratch, using P-SGD in the 15-dimensional subspace. For reasons of comparability, we start from the same initialization and use the same hyperparameters as for regular SGD.



(a) (b)

Figure 5: Experiments for ResNet8 on CIFAR-10. (a) Training performance of SGD and P-SGD in 15D subspace. (b) PCA variance ratio of the samples $W = [w_1, \dots, w_{30}]$.

Figure 5(a) compares the performance of SGD and P-SGD. With 3 runs, SGD obtains averagely 85.41% test accuracy and P-SGD obtains averagely 85.00%. Note that within the first 30 epochs of training, the performance of P-SGD is significantly better than the performance of SGD. This indicates, that P-SGD is able to optimize the parameters by itself, so the good performance isn't a result of the information provided by the sampling stage. Empirically, this simple experiment strongly supports the low-dimensional trajectory hypothesis.

Figure 5(b) illustrates the PCA variance ratio of the sampled parameters $W = [w_1, \dots, w_{30}]$. This concept can be explained as follows. Given a singular value decomposition

$$W = U\Sigma V^{\mathsf{T}},$$

where $\Sigma = \operatorname{diag}(\sigma_1, \dots, \sigma_{30})$, the PCA variance ratio of the k-th component is defined as

$$\frac{\text{variance of component } k}{\text{total variance}} = \frac{\sigma_k}{\sum_{i=1}^{30} \sigma_i}.$$

The fact that the first five components contribute nearly 80% to the total variance, indicate that, indeed the optimization trajectory could be embedded in a very low-dimensional affine set. This coincides with the good performance of training in a subspace of dimension 15.

5.1 Verification on Various Architectures

Next we verify the previous findings for more complex neural network architectures. Similar to section 5.1 and 5.2 of [1], we train 10 neural network architectures on the CIFAR-100 data set. These include widely known architectures like VGG11, DenseNet121 or GoogLeNet. Just like before we normalize the data set by channel-wise mean and variance and perform the same data augmentations as for CIFAR-10. At first we conduct the full parameter training via SGD with batch size of 128, momentum of 0.9 and weight decay of 10^{-4}. All of the models are trained for 200 epochs each, using a learning rate of 0.1 for the first 150 epochs. For the final 50 epochs a reduced learning rate of 0.01 is used. During this training we adopt the same sampling strategy as for the motivational experiment, which is to sample ones after each epoch. However, to allow for a larger subspace dimension we sample the parameters for the first 100 epochs. This setup is imitated from section 5.1 and 5.2 of [1].

Table 1 reports the test accuracy of SGD after training for 50, 100 and 200 epochs. The results are exactly the findings of table 2 in [1]. Note that the authors report the actual accuracy after the corresponding number of epochs. For example this can be seen for the SqueezeNet, where the accuracy after 100 epochs is lower than after 50 epochs. In order to ensure a fair comparison to the performance of training in low-dimensional subspaces, instead we consider the maximal test accuracy after 50, 100 and 200 epochs, reported in table 2. Comparing the number of parameters in these two tables, we notice slight differences. This is most likely explained by negligence in [1] and irrelevant for the further analysis.

After training the models in full parameter space, we conduct further experiments on the existence of low-dimensional optimization trajectories. The detailed approach is to train 40 epochs of P-SGD in a 40-dimensional subspace. Again, the specific choice of dimension 40 is arbitrary and was imitated from [1]. Note that this dimension is used for all of the 10 models. The hyperparameter setting is as follows. Similar to regular SGD training we adopt the same batch size of 128 and momentum of 0.9. For P-SGD however, no weight decay is used and the initial learning rate is set to 1, which is reduced to 0.1 after the first 30 epochs. For reasons of comparability, SGD and P-SGD start with the same initialization. The maximal test accuracy, when using the first 50 or the first 100 samples for subspace extraction is reported in table 2. Note that these are two different subspaces, so two independent optimizations, that should not be confused with the sequential results reported for regular SGD training.

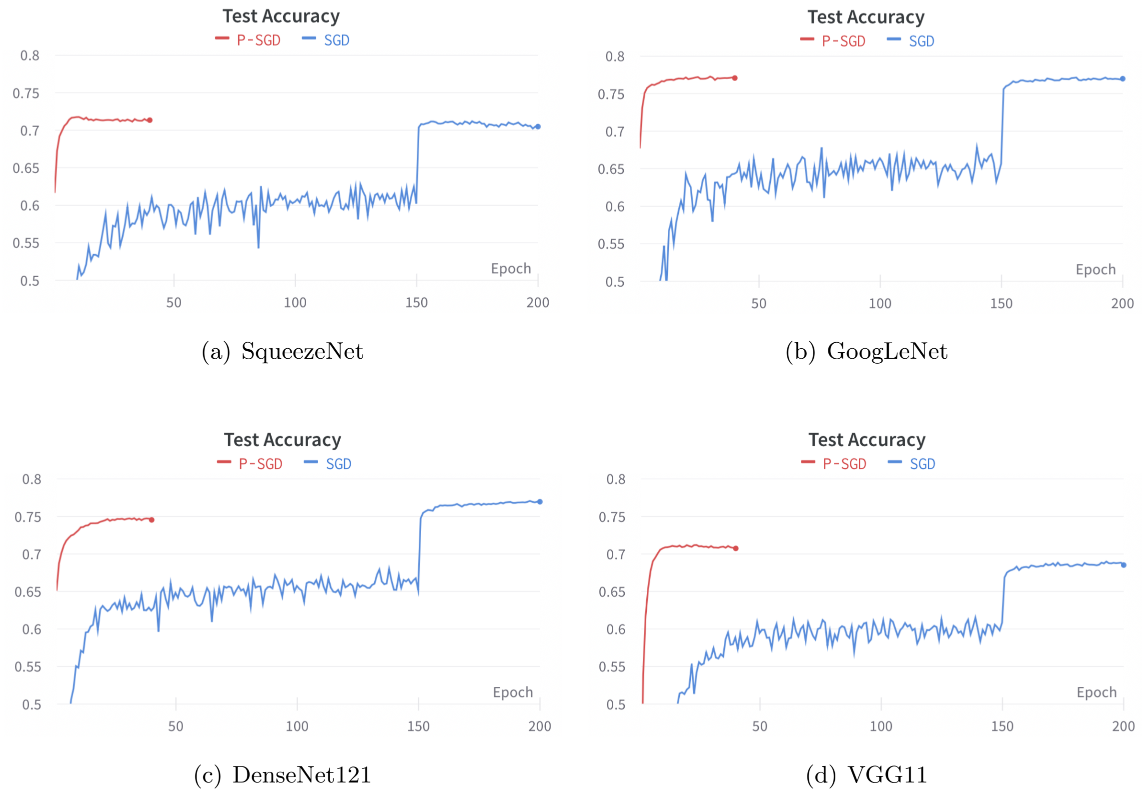
model	#params	SGD			P-SGD	
		#training epochs			#sampling epochs	
		50	100	200	50	100
SqueezeNet [18]	0.78M	59.52	58.56	70.29	69.89	70.60
ShuffleNetv2 [19]	1.3M	62.90	63.15	72.06	71.34	72.29
MobileNet [20]	3.3M	57.15	58.67	67.94	66.86	68.00
EfficientNet [21]	4.14M	62.53	63.68	72.94	71.68	72.64
GoogLeNet [22]	6.2M	62.32	66.32	76.88	75.66	77.27
DenseNet121 [23]	7.0M	65.39	64.57	76.76	74.25	76.34
SEResNet18 [24]	11.4M	64.74	64.68	74.95	74.33	75.09
Xception [25]	21.0M	64.57	65.81	75.47	75.68	75.56
Inceptionv3 [26]	22.3M	61.68	64.00	76.25	75.15	76.83
VGG11_bn [27]	28.5M	58.38	59.90	68.87	68.72	70.18

Table 1: Test accuracy on CIFAR-100 using regular SGD training and P-SGD training in 40-dimensional subspaces. Source: Table 2 in [1].

model	#params	SGD			P-SGD	
		#training epochs			#sampling epochs	
		50	100	200	50	100
SqueezeNet [18]	0.78M	60.96	62.49	71.14	69.45	71.70
ShuffleNetv2 [19]	1.36M	63.84	64.53	72.57	72.13	73.54
MobileNet [20]	3.32M	58.69	59.84	67.94	67.01	68.76
EfficientNet [21]	4.14M	62.53	63.44	73.21	71.52	73.47
GoogLeNet [22]	6.40M	66.20	67.78	77.09	75.67	77.22
DenseNet121 [23]	7.05M	66.02	67.12	77.01	74.62	74.71
SEResNet18 [24]	11.31M	65.64	66.97	74.82	74.94	75.59
Xception [25]	21.01M	66.51	67.45	75.81	75.35	75.55
Inceptionv3 [26]	22.32M	64.69	65.80	75.57	73.78	75.89
VGG11_bn [27]	28.52M	60.88	61.14	68.94	69.94	71.14

Table 2: Maximal test accuracy on CIFAR-100 using regular SGD training and P-SGD training in 40-dimensional subspaces.

Clearly, the experiments indicate a comparable performance of training in a 40-dimensional subspace to training in the full parameter space. These findings are empirically verified for 10 different architectures of various complexity, which strongly supports the existence of low-dimensional optimization trajectories. Comparing the performance of P-SGD for subspaces extracted based on 50 and 100 sampling epochs yields another finding. Generally, the performance of P-SGD seems to be better if the subspace extraction method has access to a longer piece of the regular optimization trajectory. A fact that is not directly visible in the table is the fast convergence of P-SGD. Note that we have used an initial learning rate of 1 which is usually too large for regular SGD training. In the low-dimensional subspace however, there are just a few optimizable variables, such that a larger learning rate seems to be applicable. Exemplary, the fast convergence is visualized for the smallest model (SqueezeNet), the largest model (VGG11) and two medium sized models (DenseNet121 and GoogLeNet) in figure 6. Clearly, P-SGD quickly surpasses the performance of SGD and needs just 10 to 20 epochs to almost reach its peak. These observations also apply to the remaining six models. Their corresponding charts can be found in figure 10 in the appendix.



(a) SqueezeNet

(b) GoogLeNet

(c) DenseNet121

(d) VGG11

Figure 6: Training performance on CIFAR-100 using regular SGD training and P-SGD training in 40-dimensional subspaces.

Having verified the possible existence of low-dimensional optimization trajectories, we now focus on using these findings to design powerful optimization methods. As of now, training in low-dimensional subspaces was always preceded by a heavy sampling stage based on regular SGD training. As illustrated in figure 6, the actual training in the low-dimensional subspace seems to happen pretty fast. Thus, to design a cost efficient but powerful low-dimensional optimization method, it is key to minimize the computational effort spent on subspace extraction. Recall that the subspace extraction is achieved by the DLDR algorithm in which the key challenge consists of finding a suitable dimension, generating meaningful samples and extracting an orthonormal basis via principal component analysis.

To empirically investigate the complexity of DLDR, we compare the wall-clock time spent on sampling the parameters to the time spent on performing PCA on these samples. Note that generating one sample corresponds to training for one epoch of SGD. For the PCA we analyze the extraction of 40-dimensional subspaces based on $t = 50, 100, 150, 200$ samples which includes singular value decomposition of $(t+1) \times (t+1)$ matrices. Both, the parameter sampling and the PCA are performed on the GPU. The results are reported in table 3.

models	#params	training time	PCA time for #samples			
		per epoch	50	100	150	200
SqueezeNet [18]	0.78M	16.830	0.016	0.020	0.023	0.043
ShuffleNetv2 [19]	1.36M	20.707	0.012	0.020	0.018	0.022
MobileNet [20]	3.32M	16.237	0.020	0.023	0.030	0.104
EfficientNet [21]	4.14M	23.132	0.018	0.028	0.051	0.190
GoogLeNet [22]	6.40M	28.068	0.025	0.030	0.048	0.051
DenseNet121 [23]	7.05M	43.932	0.027	0.033	0.245	0.226
SEResNet18 [24]	11.31M	18.377	0.027	0.250	0.219	0.081
Xception [25]	21.01M	31.819	0.412	0.759	0.473	0.634
Inceptionv3 [26]	22.32M	68.253	0.050	0.584	0.513	0.677
VGG11_bn [27]	28.52M	14.528	0.049	0.076	0.647	0.654

Table 3: DLDR time consumption in seconds (s) to extract 40-dimensional subspaces for training on CIFAR-100.

The complexity of generating t samples far exceeds the complexity of performing PCA on these samples, which coincides with the theoretical analysis from section 4.4. Generally, the PCA computation needs more time the larger the model and the more samples are used, but the computational effort spent is still negligible. Therefore, for our purpose of designing a cost efficient but powerful low-dimensional optimization method, we should focus on minimizing the number of SGD epochs trained during the sampling stage. As of now, the subspace dimension was chosen by random guessing and the samples were generated by a very naive technique, namely sampling once after each epoch. With regard to that, the upcoming experiments focus on addressing the following two main questions.

Which strategy should be used for sampling?
Which subspace dimension should be used for training?

5.2 Sampling Strategies

Sampling once after each epoch of SGD as in [1] is a very basic stratgey, that can be affected by a lot of randomness. A more sophisticated and stable technique is to take the average of all parameter vectors during one epoch. That is to extract the network's parameter vector after each batch and to use the mean of these parameters as the final sample per epoch. Intuitively, this should not only balance the randomness of discrete samples but also extract more information from the optimization trajectory, leading to fewer samples needed for the subspace extraction. To study the performance of this improved sampling strategy we adopt the same experiment setup as in section 5.1 and compare the results of training in 40-dimensional subspaces using the naive discrete sampling strategy to the average sampling strategy.

model	Discrete		Average				
	#sampling epochs		#sampling epochs				
	50	100	50	75	100	125	150
SqueezeNet [18]	69.45	71.70	70.19	71.55	71.99	72.36	72.52
ShuffleNetv2 [19]	72.13	73.54	72.27	72.98	73.42	73.33	73.35
MobileNet [20]	67.01	68.76	67.81	68.60	68.63	68.89	68.92
EfficientNet [21]	71.52	73.47	72.15	72.90	73.66	74.05	74.15
GoogLeNet [22]	75.67	77.22	75.85	76.65	77.16	77.19	77.13
DenseNet121 [23]	74.62	74.71	74.54	74.55	74.48	74.52	74.49
SEResNet18 [24]	74.94	75.59	74.99	75.58	75.59	75.70	75.79
Xception [25]	75.35	75.55	75.66	75.54	75.59	75.64	75.58
Inceptionv3 [26]	73.78	75.89	74.72	75.88	75.98	75.91	75.86
VGG11_bn [27]	69.94	71.14	69.98	70.91	71.22	71.21	71.24

Table 4: Maximal test accuracy on CIFAR-100 using 40 epochs of P-SGD training in 40-dimensional subspaces.

Generally, the performance of training in subspaces extracted based on 50 samples seems to be better when using the average sampling technique rather than the discrete one. For 100 samples however there seems to be no big difference at all. Presumably the averaging effect indirectly occurs for the discrete sampling strategy too, due to the larger number of samples. Another finding is, that the number of samples needed to extract the subspace differs from model to model. For example the performance of SqueezeNet is significantly better, the more samples are used for the subspace extraction where as 50 or even fewer samples seem to be sufficient for the DenseNet121. Therefore, in order to minimize the number of SGD epochs trained, it would be useful to have a criterium to decide how many samples are needed for the subspace extraction. Let us keep this idea in mind for later purposes, but first investigate further sampling strategies and analyze different choices of subspace dimensions.

Two address the two main questions, we compare the performance of training in subspaces of various dimensions using several sampling strategies. We step back to the setting in the motivational experiment, which is training a ResNet8 on CIFAR-10. Hyperparameters are also adopted. For reasons of comparability, in all of these experiments we sample based on the same optimization trajectory of 30 epochs length. However, to allow for larger subspaces and to investigate several sampling strategies, we vary the frequency of samples per epoch from 1 to 50. This enables us to generate up to 1500 samples during the first 30 epochs of regular training with SGD. Note that with a batch size of 128, one epoch consist of 391 batches, such that a frequency of 50 samples per epoch roughly translates to sampling after 8 batches each. Sampling even more frequently makes no sense, since the distance between samples would become too small. Similar to the motivational experiment, we use these samples to extract the lower-dimensional subspaces. The detailed approach is to test all sampling strategies for training in spaces of various dimensions, ranging from 2 up to 1000. We apply the average sampling technique and report the results after 40 epochs of training with P-SGD in table 5.

		dimension of subspace								
		2	5	15	25	50	100	500	1000	best
#samples	30	50.15	85.44	85.22	85.16	-	-	-	-	85.44
	90	63.71	85.42	85.27	85.15	85.10	-	-	-	85.42
	150	69.48	85.43	85.27	85.13	85.08	85.14	-	-	85.43
	300	76.16	85.47	85.27	85.15	85.14	85.14	-	-	85.47
	900	83.41	85.45	85.29	85.12	85.20	85.09	85.14	-	85.45
	1500	85.43	85.44	85.33	85.14	85.20	85.15	85.28	85.42	85.44
	best	85.43	85.47	85.33	85.16	85.20	85.15	85.28	85.42	85.47

Table 5: Maximal test accuracy of ResNet8 trained on CIFAR-10 using P-SGD in subspaces extracted with average samples.

The experiment empirically provides three major findings. The first one is based on the last column in table 5. Clearly, for ResNet8, there seems to be no big difference in performance when using different frequencies of samples per epoch. The second finding is based on the last row of table 5. Also, for ResNet8, there seems to be no big difference in performance when training in a subspace of arbitrary dimension between 5 and 1000. Note that the best results were achieved by training in a subspace of dimension 5 which could be explained by some kind of noise reduction due to only few variables being optimized. The third and final finding is, that for most of the sampling strategies, a two-dimensional subspace does not sufficiently cover the optimization trajectory. However, it is remarkable that for higher sampling frequencies, ResNet8 can still be trained in just a two-dimensional subspace. In order to avoid confusion, we should note that this result does not imply that the network's training dynamics can be described by just two independent variables. Rather the network's optimization trajectory is approximately contained in a 2-dimensional plane. Moreover, we still have to find the orthogonal projection onto that plane, to apply the P-SGD optimizer.

Assuming that the findings in table 5 translate to other architectures and data sets as well, we can finally answer the two main questions as follows. The dimension of the subspace should be chosen large enough, such that the frequency of samples per epoch does not matter and small enough, so that the computational effort is kept small and potential noise reduction effects can occur. With respect to that choice, samples should be generated only once per epoch to further reduce the computational effort. Furthermore, the average sampling strategy should be applied. With respect to the findings in table 5, the optimal choice for ResNet8 would be a 5-dimensional subspace extracted based on 30 samples.

Before we turn these discoveries into powerful optimization methods, we still have to investigate how many regular SGD epochs are needed to sufficiently extract the low-dimensional subspace. With respect to that we conduct a follow-up experiment on ResNet8. Its main focus is to investigate the spacial information included in the early stages of regular SGD training. Note that table 4 already indicates that for certain models, the spacial information seems to still evolve after a lot of epochs. However, in the following experiment we are mainly interested in how much spacial information is contained in the early stages of training. The detailed procedure is to train ResNet8 via P-SGD for 40 epochs in subspaces of dimension 5, 15 and 25. For the subspace extraction we always use a fixed number of 30 samples. However, these samples are generated with several frequencies, translating to different number of epochs needed in the sampling stage. The results are reported in table 6.

sampling strategy			dimension of subspace		
frequency	#epochs	#samples	5	15	25
1/epoch	30	30	85.44	85.22	85.16
2/epoch	15	30	83.25	83.29	83.30
3/epoch	10	30	81.54	81.44	81.45
5/epoch	6	30	78.06	78.26	78.20
10/epoch	3	30	72.16	72.45	72.38

Table 6: Maximal test accuracy of ResNet8 trained on CIFAR-10 using P-SGD for 40 epochs.

Clearly, the performance decreases when using a smaller piece of the regular optimization trajectory for the subspace extraction. This is what we would expect with regard to the previous experiments. It is remarkable however, that training in a subspace, extracted based on the first 6 epochs of SGD training still yields more than 90% of the performance of regular SGD training. As a consequence, a reasonable approach would be to combine the information-packed early samples from multiple SGD runs to improve the subspace extraction. Clearly, using random initializations for these different runs would provide very diverse optimization trajectories due to the different starting points in the parameter space. Instead one could train the network several times from the same initialization, which would still produce different optimization trajectories due to the randomness in SGD. Combining the spacial information from these optimization trajectories could then improve the subspace extraction. To investi-

sampling strategy			#trajectories			
frequency	#epochs	#samples	1	3	5	10
1/epoch	30	30	85.22	85.51	85.51	84.77
2/epoch	15	30	83.29	83.58	83.26	81.80
3/epoch	10	30	81.44	80.85	81.05	79.77
5/epoch	6	30	78.26	78.01	77.61	76.54
10/epoch	3	30	72.45	71.25	71.44	69.75

Table 7: Maximal test accuracy of ResNet8 trained on CIFAR-10 using P-SGD in 15D subspace for 40 epochs.

gate this idea in a numerical experiment, we adapt the same setup as before and train for 40 epochs in a 15-dimensional subspace. However, to extract this subspace we combine the samples from a different number of optimization trajectories starting from the same initialization.

Table 7 indicates that, in general, the performance decreases the more trajectories are combined for the subspace extraction. Most likely the additional trajectories are still very similar due to the average sampling technique. Anyways, the idea of combining multiple short optimization trajectories to generate the whole spacial information seems not to work so that we are still forced to train a certain amount of SGD epochs.

5.3 Algorithm Design

Next, the prior findings will be used to design cost efficient but powerful low-dimensional optimization algorithms. The basic idea is still to use SGD to generate samples, extract a subspace based on these samples and finally train via P-SGD in the low-dimensional subspace. As the PCA computation is negligible by table 3, one can focus on minimizing the computational effort spent on the sampling stage. For this, two approaches suggest themselves. One approach is to speed-up the evolution of the optimization trajectory, so to generate the same information in a shorter period of time. The other approach is to determine the minimal length of the optimization trajectory needed, to provide good performance of P-SGD.

Let us start with the first approach of speeding-up the trajectory evolution. In table 6, we have seen that, at least for ResNet8, a lot of spacial information seems to be contained in the early epochs of SGD training. Since convergence happens a lot faster in low-dimensional subspaces, one could train a few epochs of SGD to extract an approximation of the optimal subspace and train a little in this approximated subspace to speed things up. Of course, training in the subspace won't provide further information on the optimization trajectory, so one should fall back to regular SGD training after a while to generate more samples. These could then be used to extract an improved approximation of the optimal subspace. An iterative application of this idea could reach a final performance comparable to regular SGD training, while saving some time due to the fast convergence in subspaces. Of course, this is completely hypothetical so far and needs some practical investigation. Therefore, we train a SqueezeNet for 10 epochs with SGD to generate 10 samples. Based on these, we extract a 5-dimensional subspace and train for one epoch of P-SGD. After that we fallback to regular SGD training for another 10 epochs which are again used to train one epoch of P-SGD in a refined 5-dimensional subspace. This procedure is repeated 10 times for a total of 110 epochs (100 epochs SGD and 10 epochs P-SGD). In contrast to the previous experiments, where P-SGD always started from scratch, each P-SGD phase now starts with the final parameters from its preceding SGD phase. Both for SGD and P-SGD a learning rate of 0.1 is used. The remaining experiment setup is adopted from section 5.1.

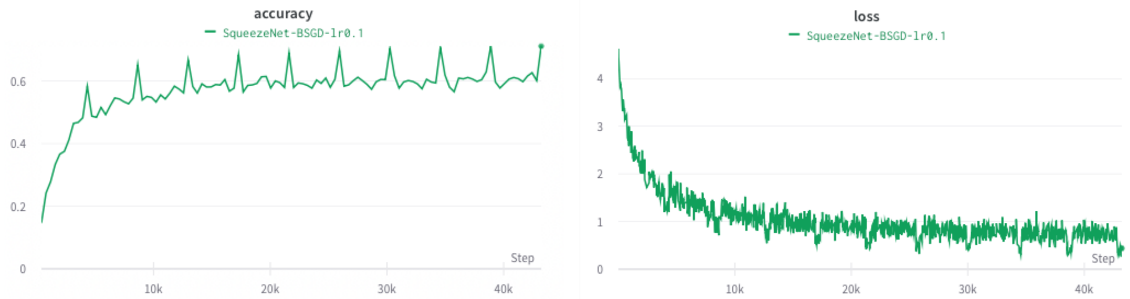
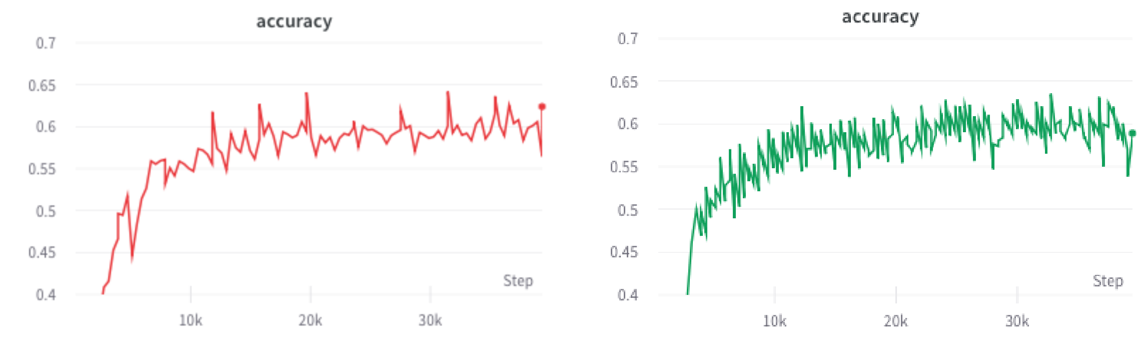


Figure 7: Training performance of SqueezeNet using 1 epoch of P-SGD after each 10 epochs of SGD.

Figure 7 illustrates the performance of this procedure. As expected, once the P-SGD training starts, both test accuracy and loss experience a jump due to the faster convergence in the subspace. Unfortunately however, when falling back to regular SGD training, we loose the progress made during the P-SGD phase and the performance decreases to its level prior to the P-SGD phase. One explanation for this could be the following. Casually speaking, when training in an approximation of the true subspace, we are optimizing in the wrong direction, so once we fallback to regular SGD training we have to adapt the training direction and loose the progress made. To ensure that we don't optimize too far in the wrong direction, the same experiment is re-run, but the P-SGD phase is shortened to just one single batch. Also, since the PCA computation costs almost nothing, we conduct another experiment where 10 initial SGD epochs are trained to get rid of the initial randomness. After that we train one intermediate batch of P-SGD after each epoch of regular SGD.

Figure 8 illustrates that, again, right after the P-SGD training, the test accuracy peaks but decreases immediately once the next SGD phase starts. Overall the maximal performance is better than the performance of regular SGD training due to the break-outs through P-SGD. The performance during the SGD phases however is similar to the performance of regular SGD training. As a consequence, for the purpose of speeding-up the optimization trajectories evolution, we can not detect any advantages of this procedure over regular SGD training.

We conclude that speeding up the generation of samples by iterative space refinements is not an option. At least not in the proposed manner. Nonetheless, the experiment indicates that training in low-dimensional subspaces can start at any time during regular SGD training and



(a)

(b)

Figure 8: Training performance of SqueezeNet using 3 batches of P-SGD (a) after each 10 epochs of SGD and (b) after each epoch of SGD.

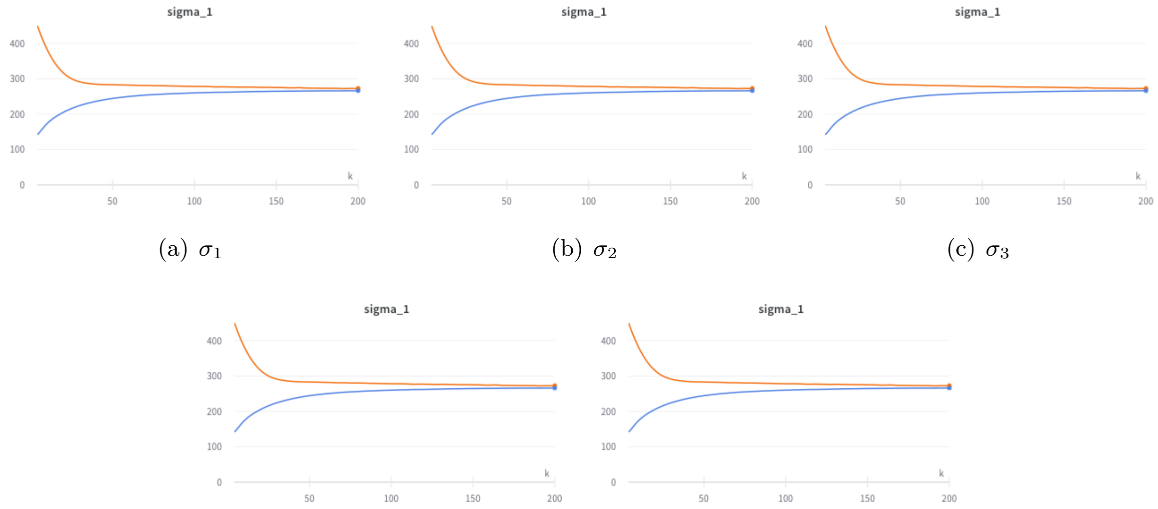
does not have to be conducted from scratch. Also, training in the projected subspace at any time during regular SGD training leads to a significant improvement in performance. It thus might be a good practice to finish regular SGD training with a few final epochs of P-SGD to maximize the performance.

Next, we explore the second approach of determining the minimal length of the optimization trajectory needed to provide good performance of P-SGD. With respect to table 4, it can be said that the necessary number of samples needed for the subspace extraction differs from model to model. Therefore, it would be useful to to have a criterium to decide once the number of samples suffices for the subspace extraction. Having found such a criterium, a future training algorithm could proceed as follows. As usual, the objective model is trained via regular SGD training and samples are generated once per epoch using the average sampling technique. During that process, the algorithm decides once the newly generated samples no longer contribute any useful information to the optimization trajectory. At this point, the SGD phase would terminate and the training could be finished by some final epochs of P-SGD.

There are two major problems of this approach. The first one is that in saddle point like regions, or more generally regions of small gradients, the change in successive samples is small, so that the SGD phase could terminate too early. Though this is a common problem for most of the training algorithms which we can do little to change. The second problem is to find a sophisticated criterium deciding once the samples no longer provide any additional information. A possible approach could be to iteratively increase the length of the optimization trajectory by a certain number of samples, say 5, and to compare the previous trajectory with the new one. The problem with that approach is that the more samples we generate, the smaller the newly generated pieces become in comparison to the existing optimization trajectory. To compensate for that, during the sampling stage, one should either increase the length of the newly generated pieces or disregard the earlier samples. An obvious and fair solution is to only observe the local change in samples by fixing some $t \in \mathbb{N}$ and using at most the last m samples for the comparison. This means once we have generated t samples, for each additional sample appended to the trajectory, we ignore one sample from the beginning.

Let us now discuss how to compare the different pieces of the optimization trajectory. One approach, which is similar to the procedure used for space refinement before, would be to extract a subspace for each of the pieces to compare and to train a bit in these subspaces using P-SGD. We could then use the average loss as the criterion of comparison. Of course, this is not a deterministic process. Another and also deterministic approach would be to compare the spaces itself rather than the training performance of P-SGD in these spaces. Recall that by corollary 4.3, these spaces are fully determined by the singular values and the t-dimensional right singular vectors of the $n \times t$ data matrix of samples. A deterministic criterion of comparison could thus be constructed based on these information only.
To start the practical investigation and feasibility of the proposed ideas, we train a SqueezeNet and a DenseNet121 for 200 epochs of SGD using a constant learning rate of 0.1. The remaining experiment setup is adopted from section 5.1. During that training, we sample once per epoch using the average sampling technique. As soon as we have generated 5 samples, the optimization trajectory is iteratively extended by one sample at a time. After each extension, say at sample number $t \in \mathbb{N}$, we compare the trajectory piece $(w_i)_{i=t-5}^{t-1}$ with the trajectory piece $(w_j)_{j=t-4}^{t}$ by performing PCA on these samples. The variation along the principal components, namely the singular values of the sample matrix, is illustrated in figure 9. Further results can be found in the appendix in figures 11 and 12.

With regard to the findings in table 4, the subspace and thus the singular values should stabilize after 50 or even fewer epochs for the DenseNet121 since the performance no longer increased from that point on. For the SqueezeNet however, we measured a significant performance increase when using more samples for the subspace extraction so that the singular values should experience at least a slight change for the first 150 epochs or even more.



(a) σ_1 (b) σ_2 (c) σ_3

(d) σ_4 (e) σ_5

Figure 9: PCA variance evolution of 5 epoch long pieces of the optimization trajectory.

Based on these charts it is hard to decide the number of samples that are sufficient for the subspace extraction. Nonetheless we can formulate some observations. Naturally, in the early

epochs there seems to be a lot of change due to the random initialization. After some epochs of training however, things seem to stabilize and the singular values settle down in a certain region. For the DenseNet121 this stabilization seem to happen around the 50th samples. For the SqueezeNet however, the singular values seem to experience a slight constant growth over time and stabilization occurs only at the end. This coincides with our expectation. Of course, detecting these observations during the training is pretty challenging and does not provide a sophisticated decision criterium yet. Anyhow the experiment indicates that the proposed ideas are worth for further investigation and offer the potential to design a criterium to decide once the SGD sampling stage can be terminated. A summary of the previous ideas, in form of a sketched outline of a future low-dimensional algorithm can be found in the appendix.
The algorithm trains an initial amount of regular SGD epochs to get rid of the initial randomness. Here, the number of epochs is set to 10, but other choices could be better, depending on the objective model. After that initial phase, the optimization trajectory is iteratively extended by one sample at a time. By doing this, the algorithm extracts stepwise shifted pieces of the optimization trajectory of 5 epochs length. Of course, the choices for the number of samples to extend and the length of the pieces to compare still have to be finetuned. Having generated two successive pieces, that are approximately equal according to some comparison criterion, it is time to terminate the SGD sampling stage. Based on all previous examples, not only the most recent five, the low-dimensional subspace is extracted and the dimension is set to d_*, such that the first d_* singular values contribute at least 95% to the PCA variance of all the samples. This proportion also needs to be finetuned. Finally, the training is finished off by training with P-SGD for the remaining number of epochs. Here, the algorithm could also terminate once training in the subspace converges. Similar to the procedure of space refinement, the P-SGD phase should start from the point where SGD training terminates. Again, this algorithm should be seen as a sketch only, which could be used for further works. Most of the parameters are random guesses and need to be adjusted. Also, there is still the major problem of finding a sophisticated comparison criterium.

Algorithm 10

```
Input: Differentiable $E : \mathbb{R}^n \to \mathbb{R}$, batch size $b \in \mathbb{N}$, number of epochs $N \geqslant 10 \in \mathbb{N}$, $w_0 \in \mathbb{R}^n$
Train 10 epochs of SGD and sample parameters $w_1, \dots, w_{10} \in \mathbb{R}^n$;
Initialize $k \leftarrow 11$, $W \leftarrow [w_0, \dots, w_{10}]$, $\tilde{\Sigma} \leftarrow [\ ]$, $\tilde{V} \leftarrow [\ ]$;
while $k \leqslant N$ do:
    Train one epoch of SGD and sample parameters $w_k \in \mathbb{R}^n$;
    Set $W \leftarrow W + [w_k]$;
    Compute the mean $\overline{w}_5 \coloneqq \frac{1}{5}\big(w_{k-4} + \cdots + w_k\big)$;
    Define the recent 5 centralized samples as $W_5 \coloneqq [w_{k-4} - \overline{w}_5, \dots, w_k - \overline{w}_5] \in \mathbb{R}^{n \times 5}$;
    Perform spectral decomposition, such that $W_5^\mathsf{T} W_5 = V\Sigma^2 V^{-1} \in \mathbb{R}^{5 \times 5}$;
    if $\tilde{V} \neq [\ ]$ do:
        Choose an appropriate criterium to compare two subspaces of $\mathbb{R}^n$;
        if $\tilde{V} \approx V$ and $\tilde{\Sigma} \approx \Sigma$ do:
            Compute the mean $\overline{w} \coloneqq \frac{1}{k+1} \sum_{i=0}^{k} w_i$;
            Centralize the samples as $W = [w_0 - \overline{w}, \dots, w_k - \overline{w}]$;
            Perform spectral decomposition such that $W^\mathsf{T} W = V\Sigma^2 V^{-1}$;
            Define the dimension $d_* \coloneqq \min\big\{d \leqslant k \in \mathbb{N} \mid \sum_{i=1}^{d} \sqrt{\Sigma_{i,i}} \geqslant 0.95 \cdot \sum_{i=1}^{k} \sqrt{\Sigma_{i,i}}\big\}$;
            Obtain the $d_*$ largest eigenvalues $\big[\sigma_1^2, \dots, \sigma_{d_*}^2\big]$ with eigenvectors $\big[v_1, \dots, v_{d_*}\big]$;
            Determine the singular vectors $u_i = \frac{1}{\sigma_i} W v_i$ for $i = 1, \dots, d_*$;
            Define the orthonormal basis $Q \coloneqq \big[u_1, \dots, u_{d_*}\big]$;
            Train $N - k$ epochs of P-SGD using $Q$;
            break
        end if
    end if
    Set $\tilde{\Sigma} \leftarrow \Sigma$, $\tilde{V} \leftarrow V$ and $k \leftarrow k + 1$;
end while
```

6 Conclusion

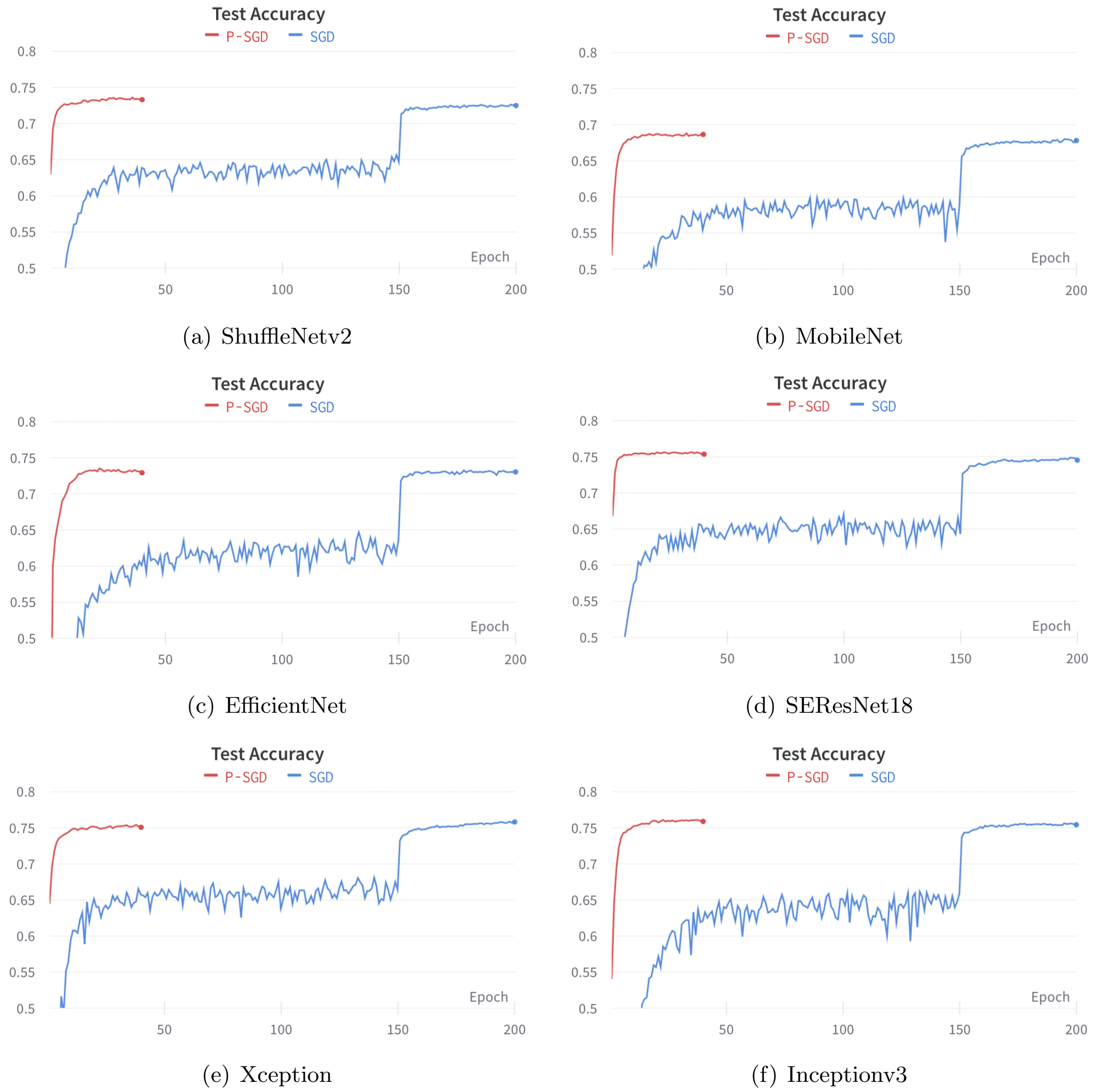
According to Appendix 8 of [9], more precisely equation (A.26), we have the following chain rule for functionals.

Lemma 6.1. *Let $f \in \mathcal{F}$ be differentiable at $x \in \mathbb{R}^{n_0}$ and $\mu : \mathcal{F} \to \mathbb{R}$ be differentiable at f. The functional chain rule states, that*

$$\partial_x \mu[f(x)] = \partial_f \mu[\partial_x f(x)]$$

In conclusion...

A Appendix



(a) ShuffleNetv2

(b) MobileNet

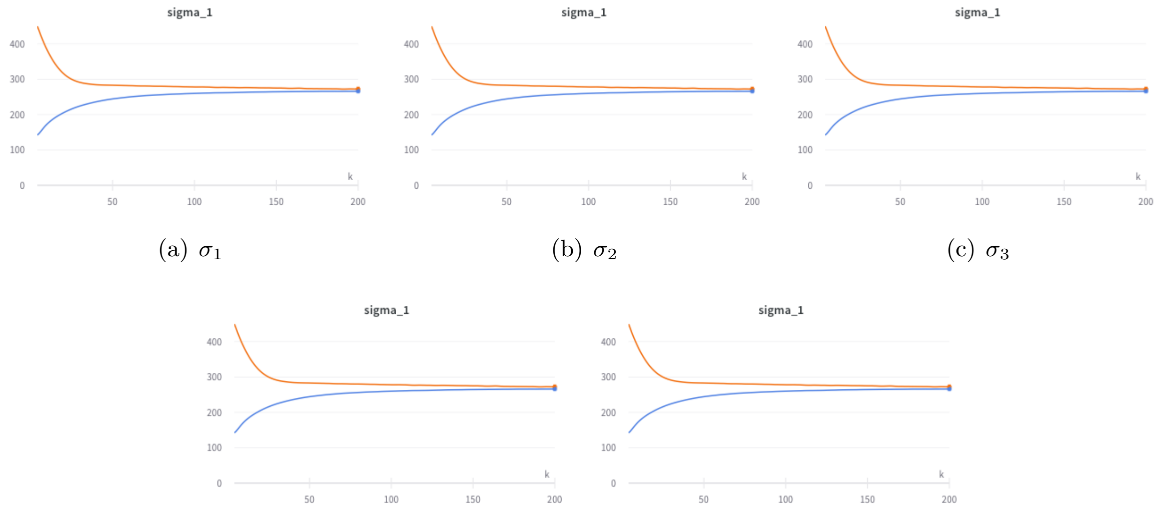
(c) EfficientNet

(d) SEResNet18

(e) Xception

(f) Inceptionv3

Figure 10: Training performance on CIFAR-100 using regular SGD training and P-SGD training in 40-dimensional subspaces.



(a) σ_1

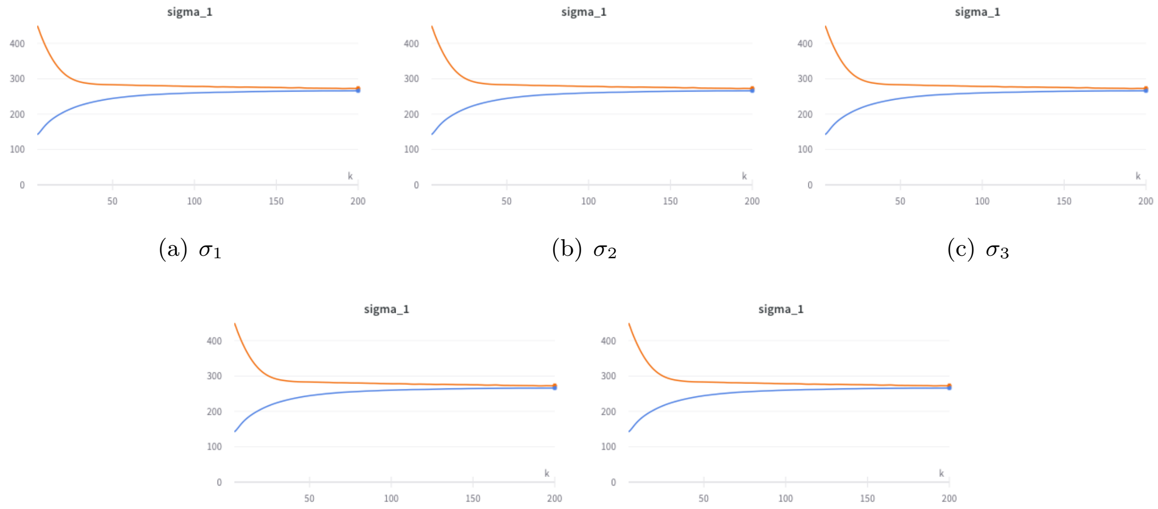
(b) σ_2

(c) σ_3

(d) σ_4

(e) σ_5

Figure 11: PCA variance evolution of 5 epoch long pieces of the optimization trajectory.



(a) σ_1

(b) σ_2

(c) σ_3

(d) σ_4

(e) σ_5

Figure 12: PCA variance evolution of 5 epoch long pieces of the optimization trajectory.

References

[1] Tao Li, Lei Tan, Qinghua Tao, Yipeng Liu, Xiaolin Huang. *Low Dimensional Trajectory Hypothesis is True: DNNs can be Trained in Tiny Subspaces.* IEEE Transactions on Pattern Analysis and Machine Intelligence, 2022.

[2] Augustin Cauchy. *Méthode générale pour la résolution des systemes d'équations simultanées.* Comptes rendus de l'Académie des sciences, volume 25, pages 536-538, 1847.

[3] Yurii Nesterov. *Introductory Lectures on Convex Optimization.* Applied Optimization, volume 87, 2004.

[4] Richard H. Byrd, Jorge Nocedal, Robert B. Schnabel. *Representations of Quasi-Newton Matrices and their Use in Limited Memory Methods.* Mathematical Programming, volume 60(1), pages 129-156, 1994

[5] Léon Bottou, Frank E. Curtis, Jorge Nocedal. *Optimization Methods for Large-Scale Machine Learning.* SIAM Review, volume 60(2), pages 223-311, 2018.

[6] Diederik P. Kingma, Jimmy Lei Ba. *Adam: A Method for Stochastic Optimization.* International Conference on Learning Representations (ICLR), 2015.

[7] Larry Armijo. *Minimization of Functions Having Lipschitz Continuous First Partial Derivatives.* Pacific Journal of Mathematics, volume 16, no. 1, pages 1-3, 1966.

[8] Arthur Jacot, Franck Gabriel, Clément Hongler. *Neural Tangent Kernel: Convergence and Generalization in Neural Networks.* Advances in Neural Information Processing Systems, volume 31, pages 8571-8580, 2018.

[9] Robert G. Parr, Weitao Yang. *Density-Functional Theory of Atoms and Molecules.* International Series of Monographs on Chemistry Books, volume 16, 1989.

[10] Erwin Kreyszig. *Introductory Functional Analysis with Applications.* Wiley Classics Library edition, 1989.

[11] Charles A. Micchelli, Massimiliano Pontil. *On Learning Vector-Valued Functions.* Neural Computation, 17:177-204, 2005.

[12] Ali Rahimi, Ben Recht. *Random Features for Large-Scale Kernel Machines.* Advances in Neural Information Processing Systems, volume 20, pages 1177-1184, 2017.

[13] Gene H. Golub, Charles F. Van Loan. *Matrix Computations.* Third edition, 1996.

[14] Zhou Fan, Zhichao Wang. *Spectra of the Conjugate Kernel and Neural Tangent Kernel for Linear-Width Neural Networks.* Advances in Neural Information Processing Systems, volume 33, pages 7710-7721, 2020.

[15] Amnon Geifman, Abhay Yadav, Yoni Kasten, Meirav Galun, David Jacobs, Ronen Basri. *On the Similarity between the Laplace and Neural Tangent Kernels.* Advances in Neural Information Processing Systems, volume 33, pages 1451-1461, 2020.

[16] Mehryar Mohri, Afshin Rostamizadeh, Ameet Talwalkar. *Foundations of Machine Learning.* Second edition, 2018.

[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. *Deep Residual Learning for Image Recognition.* Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 770-778, 2016.

[18] Forrest N. Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, Kurt Keutzer. *SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size.* arXiv:1602.07360, 2016.

[19] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, Jian Sun. *ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design.* Proceedings of the European Conference on Computer Vision (ECCV), pages 116-131, 2018.

[20] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, Hartwig Adam. *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications.* arXiv:1704.04861, 2017.

[21] Mingxing Tan, Quoc V. Le. *EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks.* International Conference on Machine Learning, pages 6105-6114, 2019.

[22] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich. *Going Deeper with Convolutions.* Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 1-9, 2015

[23] Gao Huang, Zhuang Liu, Laurens van der Maaten, Kilian Q. Weinberger. *Densely Connected Convolutional Networks.* Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 4700-4708, 2017.

[24] Jie Hu, Li Shen, Samuel Albanie, Gang Sun, Enhua Wu. *Squeeze-and-Excitation Networks.* Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 7132-7141, 2018.

[25] François Chollet. *Xception: Deep Learning with Depthwise Separable Convolutions.* Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 1251-1258, 2017.

[26] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, Zbigniew Wojna. *Rethinking the Inception Architecture for Computer Vision.* Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 2818-2826, 2016.

[27] Karen Simonyan, Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition.* International Conference on Learning Representations (ICLR), 2015.