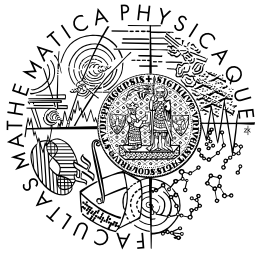


FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

BACHELOR THESIS

Václav Volhejn

Smoothness of Functions Learned by Neural Networks

Institute of Formal and Applied Linguistics

Supervisor of the bachelor thesis: Mgr. Tomáš Musil
Study programme: Computer Science
Study branch: General Computer Science

Prague 2020

I declare that I carried out this bachelor thesis independently, and only with the cited sources, literature and other professional sources. It has not been used to obtain another or the same degree.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date ....

Author's signature

I would like to thank Prof. Christoph Lampert for his invaluable guidance during my internship at IST Austria, where the research was carried out. I would also like to thank my supervisor, Mgr. Tomáš Musil, for advice on scientific writing and his support throughout.

Title: Smoothness of Functions

Learned by Neural Networks

Author: Václav Volhejn

Institute: Institute of Formal and Applied Linguistics

Supervisor: Mgr. Tomáš Musil, Institute of Formal and Applied Linguistics

Abstract: Modern neural networks can easily fit their training set perfectly. Surprisingly, they generalize well despite being "overfit" in this way, defying the bias–variance trade-off. A prevalent explanation is that stochastic gradient descent has an implicit bias which leads it to learn functions that are simple, and these simple functions generalize well. However, the specifics of this implicit bias are not well understood. In this work, we explore the hypothesis that SGD is implicitly biased towards learning functions that are smooth. We propose several measures to formalize the intuitive notion of smoothness, and conduct experiments to determine whether these measures are implicitly being optimized for. We exclude the possibility that smoothness measures based on first derivatives (the gradient) are being implicitly optimized for. Measures based on second derivatives (the Hessian), on the other hand, show promising results.

Keywords: machine learning, neural network, smoothness, generalization, implicit bias

Contents

Introduction

Classical machine learning wisdom suggests that the expressive power of a model class (the *capacity*) must be carefully balanced: if the capacity is too low, the model is *underfit* and does not manage to fit the training set, let alone the test set. If the capacity is too high, the model does fit the training set, but is *overfit* to spurious patterns and fails to represent the underlying trend, again failing to generalize well to the test set. Thus, the model generalizes best when the capacity is in a sweet-spot somewhere between underfitting and overfitting. This is known as the *bias–variance trade-o* .

Several papers have noted that neural networks apparently defy the bias–variance trade-off: increasing model capacity may improve generalization performance, even if the network is already "overfit". This phenomenon has been known for over 20 years, e.g. Lawrence et al. [1996], Caruana et al. [2000], but it has only begun receiving wider attention in recent years. This started with the work of Neyshabur et al. [2014], who showed that plotting the test loss as a function of model capacity (represented by the hidden layer size) does not yield the U-shaped curve predicted by the bias–variance trade-off. The actual results are shown in Figure 1: the test loss first decreases, then increases slightly, until the network is able to perfectly fit (*interpolate*) the training set. When the hidden layer size is increased further, the test loss surprisingly *decreases* again.

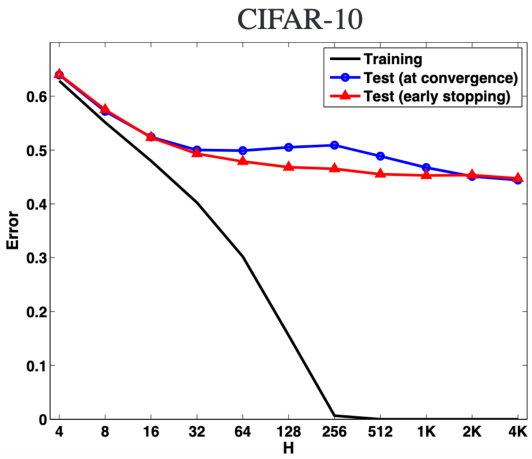


Figure 1: Loss (error) of a two-layer neural network trained on CIFAR-10 as a function of hidden layer size. The test loss at convergence exhibits the double descent phenomenon: it decreases, then increases around the point of interpolation ($H = 256$) and then decreases again. Reprinted from Neyshabur et al. [2014], Figure 1.

Neyshabur et al. [2014] suggest that the surprising generalization performance of "overfit" neural networks might be due to implicit regularization in the training process:

> A possible explanation is that the optimization is introducing some implicit regularization. That is, we are implicitly trying to find a solution with small "complexity", for some notion of complexity, perhaps

> norm. This can explain why we do not overfit even when the number of parameters is huge. Furthermore, increasing the number of units might allow for solutions that actually have lower "complexity", and thus [generalize] better.

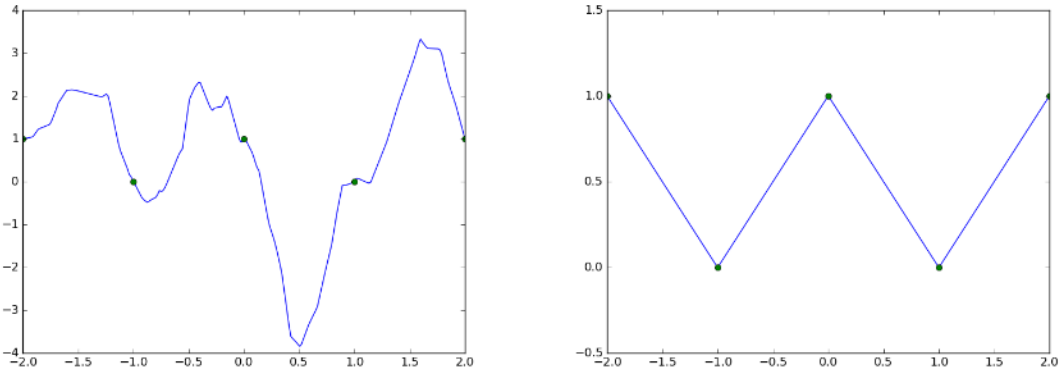


Figure 2: A non-smooth function (left) and smooth function (right) interpolating a one-dimensional dataset. Reprinted from Maennel et al. [2018].

It is still an open question what exactly the implicitly regularized complexity measure is. In this work, we explore the hypothesis that "smoothness" is implicitly regularized, i.e. that stochastic gradient descent tends to produce functions that are not unnecessarily "rough" or "bumpy". For an illustration, see Figure 2. The examined hypothesis may be summarized as follows:

1. Neural networks trained with SGD learn smooth functions (for some definition of smoothness).
2. Smooth functions generalize well.
3. Therefore, neural networks trained with SGD generalize well.

We focus primarily on empirically verifying premise 1: we perform experiments designed to measure the smoothness of neural networks trained with SGD. We design four smoothness measures and through experiments, we exclude two of them as possible candidates for the implicitly regularized measure. The remaining two show promising results, although further investigation remains as future work.

The work is organized as follows. Chapter 1 introduces the mathematical concepts used throughout our work. Chapter 2 discusses related work. Chapter 3 presents our proposed measures of smoothness and Chapter 4 discusses our experiments and their results. The final chapter summarizes our conclusions.

1. Preliminaries

In this chapter, we introduce the necessary concepts and notation from machine learning and mathematics used throughout this work.

1.1 Regression

We study the problem of learning a function $f\colon \mathbb{R}^{d_{in}} \to \mathbb{R}$ to approximate a data distribution $\mathbb{P}_{(X,Y)}$. Though we do not have direct access to $\mathbb{P}_{(X,Y)}$, we have available a training dataset $D = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)\}$ where we assume $(\boldsymbol{x}_i, y_i) \sim \mathbb{P}_{(X,Y)}$ are i.i.d. variables. We select in advance a hypothesis class $\mathcal{F}$, which might be e.g. the set of functions representable by a linear function. We want to select (*learn*) a function $f \in \mathcal{F}$, which minimizes a loss function $L(f)$. The loss function measures how well f fits the training dataset D. Our loss function is the mean squared error:

$$L(f) \coloneqq \frac{1}{n}\sum_{i=1}^{n}(f(\boldsymbol{x}_i) - y_i)^2 \tag{1.1}$$

We are primarily interested in the generalization properties of models that perfectly fit or *interpolate* the training data, meaning the learned function satisfies $L(f) = 0$. For numerical reasons, we only require $L(f) < \varepsilon$ with a small ε (typically 10^{-5}) in practice.

1.2 Neural networks

Neural networks (NNs) are machine learning models that can be used to perform regression. For our purposes, a neural network can be viewed as a directed acyclic graph, in which each node (or *unit*) u holds a scalar value a_u, referred to as the node's *activation*. Each edge has a weight; let us denote the weight of the edge from u to v as w_{uv}.

As input, we are given the values of the source nodes, i.e. nodes with no incoming edges. The value of the other nodes is computed using the edges: an edge from u to v with weight w_{uv} means "add $w_{uv}a_u$ to a_v". The outputs of the network are the values of the sink nodes, i.e. nodes with no outgoing edges. When performing regression, we require a scalar output, so let us assume that the graph only has one sink node.

Because the graph is acyclic, the values of the nodes are well-defined and can be computed easily: for a node u with a set of predecessors P, we have

$$a_u = \sum_{p \in P} w_{pu}a_p \tag{1.2}$$

which we can compute in topological order; this is known as *forward propagation*.

To add expressive power to the network, we typically also add a constant *bias term* b_u to a_u. Thus, we obtain a modified equation:

$$a_u = \sum_{p \in P} w_{pu}a_p + b_u \tag{1.3}$$

In the current formulation, the value of any node can be represented as a linear combination of the inputs. Therefore, these linear neural networks are only as expressive as linear functions. To allow for non-linearity, we employ *activation functions*: we assign in advance to each node u a certain activation function $\phi_u \colon \mathsf{R} \to \mathsf{R}$ which we apply after summing up the terms:

$$a_u = \phi_u\Big(\sum_{p \ P} w_{pu} a_p + b_u\Big) \tag{1.4}$$

A neural network is therefore specified by two parts: first, its *architecture*, which consists of the graph structure along with the activation function used in each node. This part is typically determined in advance. The second part are the *parameters*, meaning the network's weights w_{uv} and biases b_u. The parameters are determined when the network is trained using *stochastic gradient descent*.

1.3 Stochastic gradient descent

We may view a neural network with a fixed architecture as a function $f_{\boldsymbol{\theta}} \colon \mathsf{R}^{d_{in}} \to \mathsf{R}$ parametrized by $\boldsymbol{\theta}$, a vector containing all of the parameters. Here d_{in} is the number of source nodes. We wish to train the network to fit some dataset, meaning we want to find the value of $\boldsymbol{\theta}$, which minimizes a loss function $L(f_{\boldsymbol{\theta}})$.

We use the fact that the entire computation is differentiable (assuming the activation functions used are differentiable). This allows us to compute $L(f_{\boldsymbol{\theta}})$, the gradient of the loss function w.r.t. the parameters. This allows us to apply gradient descent, an algorithm for minimizing differentiable functions:

Definition 1. *Let $F \colon \mathsf{R}^n \to \mathsf{R}$ be a differentiable function. Gradient descent is an algorithm which finds $\boldsymbol{\theta} \in \mathsf{R}^n$ such that $F(\boldsymbol{\theta})$ is a local minimum of F. The algorithm is defined as follows:*

1. *Initialize $\boldsymbol{\theta}$ to a random vector using some initialization method.*
2. *Compute $\nabla F(\boldsymbol{\theta})$.*
3. *Set $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \nabla F(\boldsymbol{\theta})$.*
4. *Repeat steps 2.–4. until $L(f_{\boldsymbol{\theta}})$ stops improving.*

Here α is the learning rate, a scalar value that determines how quickly the parameters change between steps.

Essentially, in each step of the algorithm, we proceed in the steepest direction possible.

To train a neural network, we run gradient descent with $F(\boldsymbol{\theta}) \coloneqq L(f_{\boldsymbol{\theta}})$. In practice, however, computing $\nabla L(f_{\boldsymbol{\theta}})$ might be expensive as it requires evaluating f for each input $\boldsymbol{x}$ in the potentially very large dataset (see Equation (1.1)). Thus in practice, we approximate $\nabla L(f_{\boldsymbol{\theta}})$ by computing $\nabla L_B(f_{\boldsymbol{\theta}})$, an approximation of the loss for a small subset (a "batch") B of the training dataset. We then compute $\nabla L_B(f_{\boldsymbol{\theta}})$ and take a step in its direction (or rather in the opposite direction, due to the minus sign in step 3).

To ensure that the samples in the dataset are represented equally, we use the following method to create the batches. We select a batch size b and order our dataset randomly into a queue. In each step, we form a batch of the queue's first b elements. When the queue becomes empty, we re-shuffle the dataset and place it into the queue again. This version of the algorithm is known as *stochastic gradient descent*.[1]

1.4 Two-layer ReLU networks

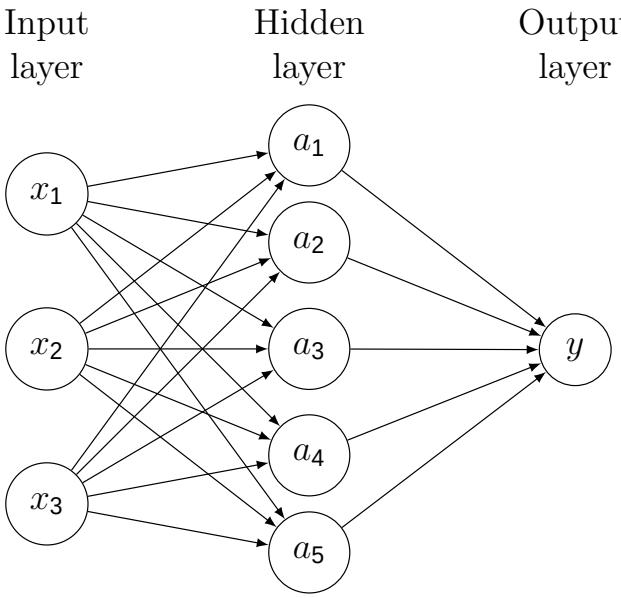


Figure 1.1: A two-layer ReLU network with $d_{in} = 3$ and $h = 5$. The activation function is ReLU for nodes a_i and the identity function for the output node y.

We study a very specific type of neural networks: two-layer neural networks with ReLU activations. This is perhaps the simplest NN architecture on which the surprising generalization properties of neural networks are still observable, and it has been used in several other works studying NN generalization (Neyshabur et al. [2014], Maennel et al. [2018], Savarese et al. [2019]).

For brevity, we refer interchangeably to a node and its activation. The neural network is organized into two layers in addition to the input layer $\boldsymbol{x}$. The first of these is the hidden layer, which contains h nodes, denoted $a_1, \ldots, a_h$. Every input node x_i is connected to every hidden node a_j. These hidden nodes use the rectified linear unit (ReLU) activation function (Nair and Hinton [2010]), defined

[1] A technical note: more precisely, the algorithm described here is stochastic gradient descent *with mini-batches*; "vanilla" SGD deals only with one sample at a time. However, the version with mini-batches described here is the one prevalent in practice since it is more e□cient and less noisy. Thus, the last part of the name is typically omitted.

as $[a]_+ \coloneqq \max(0, a)$. We can concisely express the activation a_i as:

$$a_i \coloneqq \left[\sum_{j=1}^{d_{in}} w_{ij}^{(1)} x_j + b_i^{(1)}\right]_+ = \left[\ \boldsymbol{w}_i^{(1)}, \boldsymbol{x}\ + b_i^{(1)}\right]_+ \tag{1.5}$$

The nodes in the hidden layer are then connected to the output layer, which consists of a single node y. This node has no activation function, or equivalently, it uses the identity function $\phi(x) = x$ as its activation. This is because using ReLU would prevent the network from having negative output values. Summarized in an equation, we have:

$$y \coloneqq \sum_{i=1}^{h} w_i^{(2)} a_i + b^{(2)} \tag{1.6}$$

Figure 1.1 illustrates this type of neural network. Since the graph has such a simple structure, we can easily express the entire computation in a single equation:

Definition 2. *A two-layer ReLU network is a function* $f_{\boldsymbol{\theta}} \colon \mathsf{R}^{d_{in}} \to \mathsf{R}$ *defined as follows:*

$$f_{\boldsymbol{\theta}}(\boldsymbol{x}) \coloneqq \sum_{i=1}^{h} w_i^{(2)}\left[\ \boldsymbol{w}_i^{(1)}, \boldsymbol{x}\ + b_i^{(1)}\right]_+ + b^{(2)} \tag{1.7}$$

where $[\boldsymbol{a}]_+ \coloneqq \max(0, \boldsymbol{a})$ *taken elementwise,* h *is the number of hidden units and* $\boldsymbol{\theta}$ *represents the network's weights:*

$$\boldsymbol{\theta} \coloneqq (\boldsymbol{W}^{(1)}, \boldsymbol{b}^{(1)}, \boldsymbol{w}^{(2)}, b^{(2)})$$

where $\boldsymbol{W}^{(1)} \quad \mathsf{R}^{h \times d_{in}}, \boldsymbol{b}^{(1)} \quad \mathsf{R}^{h}, \boldsymbol{w}^{(2)} \quad \mathsf{R}^{h}, b^{(2)} \quad \mathsf{R}$*, and* $\boldsymbol{w}_i^{(1)}$ *is the* i*-th column of* $\boldsymbol{W}^{(1)}$*.*

In Section 1.3, we mentioned that a neural network's activation functions should be differentiable to allow us to compute $L(f_{\boldsymbol{\theta}})$, yet ReLU is not differentiable at 0. In practice, this is not a problem since ReLU is still differentiable almost everywhere, and we can simply postulate that its derivative at 0 is 1.

1.5 Regularization

Consider a measure S, which accepts a model $f \colon \mathsf{R}^{d_{in}} \to \mathsf{R}$ as input, and outputs a scalar value. We say that a measure S is being *regularized* during training if, in addition to minimizing loss, the training process tends to find functions that have a low value of S. Note that "regularization" is sometimes taken to have a broader meaning, but this does not concern us here.

Regularization may be explicit or implicit. When training a model $f_{\boldsymbol{\theta}}$ parametrized by a vector $\boldsymbol{\theta}$, we may regularize a measure explicitly by adding a regularization term to the loss:

$$L_{reg}(f_{\boldsymbol{\theta}}) \coloneqq L(f_{\boldsymbol{\theta}}) + \lambda S(f_{\boldsymbol{\theta}}) \tag{1.8}$$

where λ is a coefficient determining the strength of regularization—the relative importance of the regularization term. In this way, the model is "punished" for

large values of S. In particular, if there is a value of $\boldsymbol{\theta}$ for which the model fits the data perfectly ($L(f_{\boldsymbol{\theta}}) = 0$), the minimizer of $L_{reg}(f_{\boldsymbol{\theta}})$ is the $\boldsymbol{\theta}$ which minimizes $S(f_{\boldsymbol{\theta}})$ in addition to perfectly fitting the data.

One popular choice of S is the L_2 norm of the parameter vector $\boldsymbol{\theta}$, defined as $S(f_{\boldsymbol{\theta}}) = \|\boldsymbol{\theta}\|^2$. L_2 regularization leads training to favor low-norm solutions, which often correspond to simpler models.

Regularization can also be *implicit*: due to some properties of the training algorithm or the model class, training may favor models that have specific properties, e.g. a low value of some complexity measure. For instance, using a smaller batch size in SGD is known to improve performance; Keskar et al. [2017] propose that this is because, with a smaller batch size, the optimization tends to find flat minima of the loss function, which have favorable generalization properties. We can understand this as implicit regularization of the sharpness of the loss function minimum.

1.6 Total variation

For measuring smoothness, a useful notion is the *total variation*, which is a measure of "how much a function is moving". We begin with a special case and proceed to a more general definition. Let us define total variation for a differentiable function $f\colon [a,b] \to \mathbb{R}$ as:

$$TV(f) \coloneqq \int_a^b |f'(x)|\, dx \tag{1.9}$$

If we interpret $f(t)$ as the position of an object on the number line at time t, $TV(f)$ is the total distance travelled by the object from time a to time b.

The requirement that f be differentiable is unnecessarily strict; by analogy to the definition of the Riemann integral, we may extend the definition of TV to non-differentiable functions. We define the total variation for a (not necessarily differentiable) function $f\colon [a,b] \to \mathbb{R}$ as:

$$TV(f) \coloneqq \sup_{P \in \mathbb{P}_{a,b}} \sum_{i=1}^{|P|} |f(x_i) - f(x_{i-1})| \tag{1.10}$$

Here $\mathbb{P}_{a,b}$ is the set of all partitions of the interval $[a,b]$:

$$\mathbb{P}_{a,b} \coloneqq \left\{ P = (x_0, x_1, \ldots, x_{|P|}) \,\middle|\, a = x_0 < x_1 \cdots < x_{|P|} = b \right\} \tag{1.11}$$

As an example, consider $f\colon [-2,2] \to \mathbb{R}$, where $f(x) \coloneqq |x|$. Then by using the partition $(-2, 0, 2)$, we get $TV(f) \geq 4$. No other partition gives a higher TV, so $TV(f) = 4$. The definition from Equation (1.10) is also applicable to non-continuous functions, such as the Heaviside step function:

$$H(x) \coloneqq \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases} \tag{1.12}$$

On the interval $[-1,1]$, we get $TV(H) = 1$. In the "moving object" analogy, the definition from Equation (1.10) allows the object to jump instantaneously, with jumps counting towards the total distance moved. Our analogy also offers a

natural way to extend the definition to vector-valued functions $f\colon [a,b] \to \mathbb{R}^d$: the imagined object is now simply moving around in $\mathbb{R}^d$ rather than on the number line.[2]

Definition 3. *We define the total variation of* $f\colon [a,b] \to \mathbb{R}^d$ *as*

$$TV(f) \coloneqq \sup_{P \in P_{a,b}} \sum_{i=1}^{|P|} \left\| f(x_i) - f(x_{i-1}) \right\| \tag{1.13}$$

where $P_{a,b}$ *is the set of all partitions of the interval* $[a,b]$*, as defined in Equation* (1.11)*.*

Another good intuition for Definition 3 is that total variation is the arc length of the curve "drawn" by f in $\mathbb{R}^d$.

Example. Let $f\colon [-1,1] \to \mathbb{R}^2$ be a function defined as $f(t) \coloneqq (|t|, t)$. We interpret f as the definition of a parametric curve in $\mathbb{R}^2$. The curve starts in $(1,-1)$ for $t = -1$, then goes linearly to $(0,0)$ and finally reaches $(1,1)$ when $t = 1$.

Consider the partition $(-1, 0, 1)$. Then we get

$$TV(f) \geq \|(0,0) - (1,-1)\| + \|(1,1) - (0,0)\| = \sqrt{2} + \sqrt{2} = 2\sqrt{2} \tag{1.14}$$

and indeed one can verify that it is not possible achieve a higher value by using a different partition, so $TV(f) = 2\sqrt{2}$, which is the curve's arc length.

Total variation allows us to measure how much "unnecessary movement" a function is doing. Let $\mathcal{F}$ be the set of functions $f\colon [a,b] \to \mathbb{R}^d$ for which $f(a) = \boldsymbol{c}_a$ and $f(b) = \boldsymbol{c}_b$. We have

$$\min_{f \in \mathcal{F}} TV(f) = \|\boldsymbol{c}_a - \boldsymbol{c}_b\| \tag{1.15}$$

More can be said if $d = 1$: then the functions $f \in \mathcal{F}$ which minimize TV are exactly those which are weakly monotonic (either non-decreasing or non-increasing), formally:

$$f \in \arg\min_{g \in \mathcal{F}} TV(g) \qquad \Leftrightarrow \qquad f \text{ is weakly monotonic} \tag{1.16}$$

An analogous result holds even for $d > 1$, but is not relevant for the present work.

[2] There are other ways to define total variation for vector-valued functions; see Goldluecke and Cremers [2010] for a discussion. Our definition is a special case of the *pointwise Frobenius norm total variation* discussed in the paper.

2. Related work

To the best of our knowledge, the first modern paper which observed the unexpected generalization behavior of neural networks is Neyshabur et al. [2014]. The authors focus on the fact that the test loss keeps decreasing with the number of hidden units but do not investigate the increase in test loss around the interpolation point (see the blue curve in Figure 1). This "bump" was later observed in several papers (Advani and Saxe [2017], Spigler et al. [2019]), and the effect was dubbed *double descent* by Belkin et al. [2019], because of the fact that loss decreases in two parts of the curve. Nakkiran et al. [2020] confirm the findings in extensive experiments and observe that double descent occurs not only as a function of model size but also the number of training epochs.

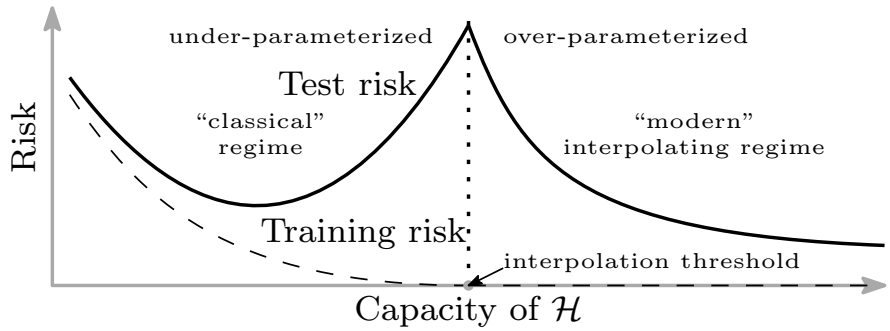


Figure 2.1: A diagram of the double descent curve. Reprinted from Belkin et al. [2019], Figure 1. The term *risk* refers to the expected value of the loss.

In a widely-cited thought-provoking work, Zhang et al. [2017] show that modern convolutional neural networks (CNNs) are able to fit datasets with random labels. This means that the networks can memorize the dataset since they learn even when there is no relationship between the inputs and outputs. Thus, their capacity is so large that they should be significantly overfit. Nevertheless, CNNs generalize well on real data.

A popular form of explanation introduced already by Neyshabur et al. [2014] is that the training procedure is implicitly biased towards solutions with low complexity. However, it is still unknown what the implicitly regularized complexity is. There have been several suggestions, for instance: sharpness of the reached loss function minimum (Keskar et al. [2017]), distance from initialization (Nagarajan and Kolter [2019]), Fisher–Rao norm (Liang et al. [2019]), or various measures based on parameter norms (Bartlett et al. [2017], Neyshabur et al. [2015]). We refer the reader to Jiang et al. [2019] for an empirical comparison of many of the proposed complexity measures.

Our approach is inspired by Maennel et al. [2018], who observe that under certain conditions, training shallow ReLU networks in the 1D setting using gradient descent yields "simple" functions that are close to a linear interpolation of the training data—see Figure 2. The initial question of the present work was whether a similar tendency towards simple/smooth functions exists in higher dimensions as well.

Novak et al. [2018] explore how the average norm of the Jacobian correlates with generalization; this is the measure which we refer to as *gradient norm* in this

work. The authors perform an empirical study and find that in the classification setting, the gradient norm is correlated with a network's generalization ability. This is especially interesting in connection with our work as our results suggest that the gradient norm is *not* the measure that is being implicitly regularized. Rifai et al. [2011] and Sokolic et al. [2017] regularize the gradient norm explicitly and find that it improves performance.

LeJeune et al. [2019] propose a measure of "rugosity" (roughness) based on the learned function's Hessian, which in our framework is a second-order smoothness measure (since it is calculated based on the second derivative). The authors draw parallels between rugosity and data augmentation; they find empirically that data augmentation decreases rugosity. They also experiment with explicitly regularizing rugosity in hopes of simulating the effect of data augmentation but find that unlike with data augmentation, the test loss *increases* when explicit regularization of rugosity is used.

Kubo et al. [2019] investigate a second-order smoothness measure based on how the learned function's gradient changes when interpolating between two samples; this is similar to one of our proposed measures. The authors empirically show that the measure's value is low, meaning that overparametrized networks interpolate almost linearly between the samples. This result is consistent with our findings.

3. Measuring Smoothness

In machine learning literature, the notion of smoothness of a function is often used intuitively (e.g. Kawaguchi et al. [2017], Belkin et al. [2019], or Kubo et al. [2019]), but is rarely defined formally.[1]Here we present several smoothness measures which assign a scalar smoothness value to a function $f\colon \mathsf{R}^{d_{in}} \to \mathsf{R}$.

Intuitively, f may be arbitrarily non-smooth ("rough"); we can always make it less smooth by adding noise. On the other hand, there is some kind of maximum smoothness upon which we cannot improve. For this reason, we define the measures S such that $S(f) \geq 0$ and such that the higher $S(f)$ is, the less smooth f is. The value $S(f) = 0$ thus means that f is as smooth as possible. For a measure to be consistent with our intuition, this ideal smoothness should be reached by constant functions, or perhaps more generally by linear functions.

Our measures may be classified into two categories: *first-order* and *second-order* smoothness measures. First-order measures are based on the gradient or first difference of f, whereas second-order measures are based on the Hessian or second difference. In the one-dimensional case, an example of a first-order measure is the total variation $TV(f)$, whereas an example of a second-order measure is the total variation of the derivative $TV(f')$.

Within one category of measures, our experiments show that the measures are strongly correlated. There is some correlation between first-order and second-order measures as well, but minimizing first-order smoothness does not necessarily imply minimizing second-order smoothness or vice versa, as we show in Section 3.3.

In this chapter, we propose four smoothness measures and discuss their properties, and we study them in more detail in the one-dimensional setting ($d_{in} = 1$).

3.1 First-order smoothness measures

A simple way to formalize smoothness is to identify it with steepness; if a function is very steep (has a large gradient), then it is not smooth.

3.1.1 Gradient norm

We define the *gradient norm* $GN(f)$ as the expected norm of the gradient of f:

Definition 4. *Let $f\colon \mathsf{R}^{d_{in}} \to \mathsf{R}$ be a differentiable function and let P_X be a probability distribution over $\mathsf{R}^{d_{in}}$. We define the gradient norm as*

$$GN(f) := \mathsf{E}_{\boldsymbol{x} \sim \mathsf{P}_X} \left\| \nabla_{\boldsymbol{x}} f(\boldsymbol{x}) \right\| \tag{3.1}$$

This definition is analogous to the definition of the total variation, as discussed in Goldluecke and Cremers [2010]. However, we take the expectation over the dataset distribution rather than integrating $\left\| \nabla_{\boldsymbol{x}} f(\boldsymbol{x}) \right\|$ over $\mathsf{R}^{d_{in}}$. Integrating would be intractable in higher dimensions, not to mention the fact that for ReLU networks, the integral would most likely converge to $+\infty$.

[1] *Smoothness* is also a technical term in mathematical analysis, but this is not the meaning of the term used in the present work.

Note that we assume f is differentiable, which is not entirely accurate since ReLU is not differentiable at 0. This shortcoming could be rectified formally by taking the distributional derivative, a weaker form of derivative. Alternatively, we could define a differentiable version ReLU by smoothing the function very slightly. Again, we may simply postulate that the derivative of ReLU at 0 is 1 to fix this in practice.

We approximate $GN(f)$ by taking 1000 samples from the test set. Note that GN is non-negative and is 0 for those functions whose gradient is zero almost everywhere.

3.1.2 Function path length

Another approach is to sample pairs of points and consider the one-dimensional segments between these points in the input space. On these one-dimensional "slices" of f, we can compute measures which would not be tractable in higher dimensions, such as the total variation. We define the total variation along a segment as follows:

Definition 5. *Let $f\colon \mathsf{R}^{d_{in}} \to \mathsf{R}^{d_{out}}$ be a function and let $\boldsymbol{a}, \boldsymbol{b} \quad \mathsf{R}^{d_{in}}$. We define the total variation of f along a segment $[\boldsymbol{a}, \boldsymbol{b}]$ as*

$$TV_{[\boldsymbol{a},\boldsymbol{b}]}(f) \coloneqq TV(f_{[\boldsymbol{a},\boldsymbol{b}]}) \tag{3.2}$$

where $f_{[\boldsymbol{a},\boldsymbol{b}]}\colon [0,1] \to \mathsf{R}$ is a "slice" of f along the segment $[\boldsymbol{a}, \boldsymbol{b}]$:

$$f_{[\boldsymbol{a},\boldsymbol{b}]}(t) \coloneqq f((1 - t)\boldsymbol{a} + t\boldsymbol{b}) \tag{3.3}$$

Now we define the *function path length* as the expected value of $TV_{[\boldsymbol{a},\boldsymbol{b}]}$ taken over the data distribution:

Definition 6. *Let $f\colon \mathsf{R}^{d_{in}} \to \mathsf{R}^{d_{out}}$ be a function and let P_X be a probability distribution over $\mathsf{R}^{d_{in}}$. We define the* function path length *as*

$$PL_{\mathsf{P}}(f) \coloneqq \mathsf{E}_{\boldsymbol{a},\boldsymbol{b} \quad \mathsf{P}_X} TV_{[\boldsymbol{a},\boldsymbol{b}]}(f) \tag{3.4}$$

Of course, we do not have direct access to P_X, so we approximate PL_{P} by utilizing the information about P_X that we do have, namely the training dataset $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ (labels are not necessary):

$$PL_{\mathsf{P}}(f) \approx \frac{1}{n^2} \sum_{i=1}^{n} \sum_{j=1}^{n} TV_{[\boldsymbol{x}_i,\boldsymbol{x}_j]}(f) \tag{3.5}$$

Since the computation time scales quadratically in n, PL_{P} quickly becomes impractically slow to compute. We thus sample a fixed number of pairs from the dataset to get a fast approximation. We also approximate $TV_{[\boldsymbol{a},\boldsymbol{b}]}$ by selecting a fixed equidistant partition of the segment $[\boldsymbol{a}, \boldsymbol{b}]$:

$$\widehat{TV}_{[\boldsymbol{a},\boldsymbol{b}]}(f, n) \coloneqq \sum_{i=1}^{n-1} f(\boldsymbol{p}_i) - f(\boldsymbol{p}_{i-1}) \tag{3.6}$$

where $\boldsymbol{p}_i \coloneqq \frac{i}{n-1}\boldsymbol{a} + \left(1 - \frac{i}{n-1}\right)\boldsymbol{b}$ for $i \quad \{0, \ldots, n - 1\}$. We sample 1000 pairs of points and use $n = 100$.

The disadvantage of PL_0 is that by being based on total variation, it cannot distinguish between some functions which we would not consider equally smooth. As described by equation (1.16), the total variation of a function with fixed endpoints is minimal for any weakly monotonic function. Consider $f(x) = x$, $g(x) = \max(0, 2x - 10)$. Then $TV(f, 0, 10) = TV(g, 0, 10) = 10$, but intuitively, f is smoother than g, since g is linear whereas f has a "tooth". The gradient norm measure shares the same problem.

3.2 Second-order smoothness measures

To be able to distinguish between functions which are monotonic, we want a measure which takes the second derivative into account. Since the ReLU activation function does not have a second derivative in the regular sense, we need a weaker notion.

3.2.1 Gradient path length

A natural way to do this is to use another segment-based measure which looks at the values of the gradient along a segment, rather than the values of the function itself.

Definition 7. *Let $f\colon \mathsf{R}^{d_{in}} \to \mathsf{R}$ be a differentiable function and let P_X be a probability distribution over $\mathsf{R}^{d_{in}}$. We define the gradient path length as*

$$PL_1(f) \coloneqq PL_0(\nabla_{\boldsymbol{x}} f) = \mathsf{E}_{\boldsymbol{a},\boldsymbol{b} \sim \mathsf{P}_X} TV_{[\boldsymbol{a},\boldsymbol{b}]}(\nabla_{\boldsymbol{x}} f) \tag{3.7}$$

Now for two datapoints $(\boldsymbol{x}_1, y_1)$ and $(\boldsymbol{x}_2, y_2)$, if $f(\boldsymbol{x}_1) = y_1$ and $f(\boldsymbol{x}_2) = y_2$, we have $TV_{[\boldsymbol{x}_1,\boldsymbol{x}_2]}(\nabla_{\boldsymbol{x}} f) = 0$ iff $\nabla_{\boldsymbol{x}} f$ is constant on the segment $(\boldsymbol{x}_1, \boldsymbol{x}_2)$. This means that the minimizer is a linear interpolation between the endpoints.

Calculating PL_1 requires access to $\nabla_{\boldsymbol{x}} f$, but second-order partial derivatives need not exist. This allows us to apply the measure to ReLU networks (again after defining the derivative of ReLU at 0 to be 1).

3.2.2 Weights product

Another second-order measure we propose is the *weights product*. It is defined as follows:

Definition 8. *Let $f_{\boldsymbol{\theta}}\colon \mathsf{R}^{d_{in}} \to \mathsf{R}$ be a two-layer ReLU network, where $\boldsymbol{\theta} = (\boldsymbol{W}^{(1)}, \boldsymbol{b}^{(1)}, \boldsymbol{w}^{(2)}, b^{(2)})$. We define the weights product WP as*

$$WP(f_{\boldsymbol{\theta}}) \coloneqq \sum_{i=1}^{h} \left|w_i^{(2)}\right| \cdot \left\|\boldsymbol{w}_i^{(1)}\right\| \tag{3.8}$$

Each summand is the norm of the difference of the gradient on the two sides of the ReLU. Thus WP is a second-order measure as it is based on the changes of the gradient.

Unlike the previous smoothness measures, WP is only defined for two-layer ReLU networks but has the advantage of being easily computable and being

independent of the dataset. Compared to the gradient path length, it is also less noisy since we can compute it without resorting to approximation.

We show in the following section that in the one-dimensional case, WP is equal to the total variation of the derivative.

3.3 One-dimensional case

Here we restrict ourselves to the simplified case where $d_{in} = 1$ and the input distribution P_X is a uniform distribution over the interval $[a, b]$. Thus the learned function is $f\colon [a, b] \to \mathsf{R}$. We analyze the behavior of the following simplified measures:

$$\begin{aligned} PL_0^{(1)}(f) &\coloneqq TV(f) \\ PL_1^{(1)}(f) &\coloneqq TV(f\,') \end{aligned} \tag{3.9}$$

Note that for this class of functions, there also exists a simple relationship between GN and $PL_0^{(1)}$: $GN(f) = \frac{1}{b-a} PL_0^{(1)}(f)$. The measure $PL_1^{(1)}$ is a second-order measure related to PL_1, which differs in the weighing. If f is twice differentiable, the measures simplify to:

$$\begin{aligned} PL_0^{(1)}(f) &\coloneqq \int_a^b |f\,'|\, dx \\ PL_1^{(1)}(f) &\coloneqq \int_a^b |f\,''|\, dx \end{aligned} \tag{3.10}$$

Under these measures, what are the smoothest functions which interpolate (perfectly fit) a certain dataset? Consider a dataset $D = \{(x_1, y_1), \ldots, (x_n, y_n)\}$, where $a = x_1 < x_2 < \cdots < x_n = b$. Let us denote by F the set of continuous piecewise linear functions $f\colon \mathsf{R} \to \mathsf{R}$ which interpolate D. (Continuous piecewise linear functions are representable by two-layer ReLU networks.)

Among the functions in F, both $PL_0^{(1)}$ and $PL_1^{(1)}$ are minimized by linear spline interpolation $f_L\colon [a, b] \to \mathsf{R}$, defined as:

$$f_L(x) \coloneqq y_i + \frac{y_{i+1} - y_i}{x_{i+1} - x_i}(x - x_i) \tag{3.11}$$

where i is chosen for a given x as the unique index for which $x_i \le x \le x_{i+1}$. First, we show that f_L minimizes TV $(= PL_0^{(1)})$.

Proof. Denote by F the subset of F which minimizes TV, and by f an arbitrarily chosen function from F. By the definition of the total variation (Equation (1.10)), we can bound TV from below by considering a fixed partition. We take the partition $P = (x_1, \ldots, x_n)$ and obtain

$$TV(f\,') \ge \sum_{i=2}^{n} |f(x_i) - f(x_{i-1})| \tag{3.12}$$

We compare this to the total variation of f_L:

$$TV(f_L) = \sum_{i=2}^{n} |f(x_i) - f(x_{i-1})| \tag{3.13}$$

Combining the two equations, we obtain $TV(f\ ) \geq TV(f_L)$, and since f F, we get $TV(f\ ) = TV(f_L)$, and so f_L F.

□

For proof of that $PL_1^{(1)}$ is minimized by f_L, see Theorem 3.3 in Savarese et al. [2019]. The measure to be minimized is slightly different, but the proof is analogous, so we only sketch it here. The core idea is to use the mean value theorem on each of the segments $[x_i, x_{i+1}]$ to show that for any function f F, its derivative f must obtain the value $\frac{y_{i+1}-y_i}{x_{i+1}-x_i} = f_L(x)$, meaning that $TV(f\ ) \geq TV(f_L)$ and so f_L minimizes $PL_1^{(1)}$.

The situation is complicated by the fact that to use the mean value theorem in this way, we need f F to be twice differentiable, which in general, it is not. The solution is to approach f by a sequence (f_r) of smoothed versions of f, which are twice differentiable. We then get the desired result in the limit.

The function f_L is not the unique minimizer of $PL_0^{(1)}$: any function which interpolates S and is piecewise monotonic on of the intervals $[x_1, x_2], \ldots, [x_{n-1}, x_n]$ achieves minimal TV (and thus $PL_0^{(1)}$). This is a consequence of a generalized version of Equation (1.16).

The measure $PL_1^{(1)}$ is also not minimized uniquely by f_L, though it is more difficult to characterize which functions minimize $PL_1^{(1)}$ for a given dataset. Informally, the function must not "overshoot" the values of f; they should be as monotonic as possible. A complete characterization is not the focus here, so, let us merely show an example of a minimizer distinct from f_L for a specific dataset:

Example. Let $D = ((-2, 1), (-1, 0), (1, 0), (2, 1))$ and define f as

$$f(x) := \begin{cases} |x| - 1 & \text{if } 1 \leq |x| \leq 2 \\ \frac{x^2-1}{2} & \text{if } |x| \leq 1 \end{cases} \tag{3.14}$$

For an illustration, see Figure 3.1. The function f interpolates D, and we have $PL_1^{(1)}(f_L) = PL_1^{(1)}(f) = 2$. However, the values of $PL_0^{(1)}$ differ, since $PL_0^{(1)}(f_L) = 2$, but $PL_0^{(1)}(f) = 3$. This example thus also shows that minimizing one of the measures does not imply minimizing the other. (The other implication, namely that minimizing $PL_0^{(1)}$ does not imply minimizing $PL_1^{(1)}$, is left as an exercise to the reader.)

We also note that for two-layer ReLU networks, $PL_1^{(1)}$ can be expressed purely as a function of the network's weights. The definition of two-layer ReLU networks (Definition 2) simplifies in the one-dimensional case to:

$$f_{\boldsymbol{\theta}}(x) = \sum_{i=1}^{h} w_i^{(2)} \left[w_i^{(1)} x + b_i^{(1)} \right]_+ + b^{(2)} \tag{3.15}$$

For short, let us write $f_{\boldsymbol{\theta}} = f$. Taking the derivative of f, we obtain:

$$f\ (x) = \sum_{i=1}^{h} w_i^{(2)} w_i^{(1)} H(w_i^{(1)} x + b_i^{(1)}) \tag{3.16}$$

where H is the Heaviside step function, defined as $H(x) = 1$ if $x \geq 0$ and $H(x) = 0$ otherwise. Note that again we are neglecting the fact that ReLU is not differentiable at 0 since it is inconsequential.

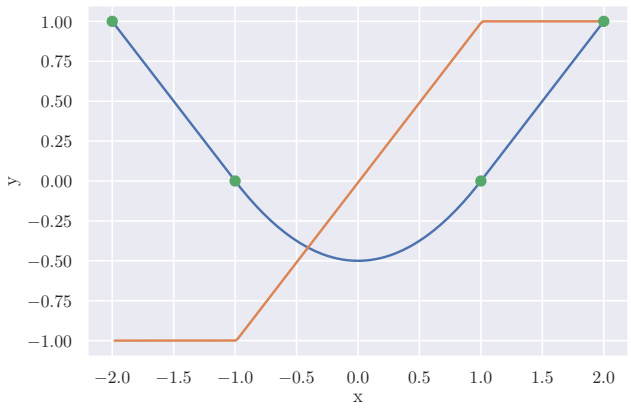


Figure 3.1: A function f (in blue) interpolating a dataset D (green points), along with its derivative f' (in orange). Among functions which interpolate D, this function minimizes $PL_1^{(1)}$ but does not minimize $PL_0^{(1)}$.

We now want to compute $PL_1^{(1)}(f) = TV(f')$. Since f' has a special form, this can be done under mild assumptions. Let us denote $c_i := -\frac{b_i^{(1)}}{w_i^{(1)}}$; when $x = c_i$, the argument of the i-th Heaviside step function in Equation (3.16) is 0, meaning that the i-th node switches between being activated and deactivated at this position.

The function f' is piecewise constant, with the knot positions c_i determining the borders of the pieces. If we assume that the values c_i are unique, the value of f' changes at position c_i by $w_i^{(2)} w_i^{(1)}$, because at this position we add or remove the i-th summand (depending on the sign of $w_i^{(1)}$).

Now if we assume that every knot position c_i lies within the input interval $[a, b]$, we have:

$$PL_1^{(1)}(f) = TV(f') = \sum_{i=1}^{h} \left| w_i^{(2)} w_i^{(1)} \right| \tag{3.17}$$

The assumption of the uniqueness of the values c_i is only weak, as this holds for almost all parameter configurations $\boldsymbol{\theta}$. Formally, if we select $\boldsymbol{\theta}$ randomly (e.g. sample each parameter independently from a Gaussian distribution), the probability that $c_i = c_j$ is 0. To prove this, we expand the right side of the equation and obtain $c_i = -b_j^{(1)} / w_j^{(1)}$. Assume we have already (randomly) determined c_i and $b_j^{(1)}$; then there is only a single value of $w_j^{(1)}$ which would make the equation hold, and the probability of randomly selecting this value is 0.

Equation 3.17 highlights the relationship between the gradient path length PL_1 and the weights product WP:

$$PL_1(f_{\boldsymbol{\theta}}) = WP(f_{\boldsymbol{\theta}}) \tag{3.18}$$

Indeed, the one-dimensional case was the original motivation for the weights product measure. In higher dimensions, the interpretation of WP is less obvious, but the measure is still correlated with PL_1.

4. Experiments

In this chapter, we describe the experiments performed and present their results.

Though we have described several smoothness measures, it is not straightforward to apply them to determine whether smooth functions are being learned. The reason is that we need a sense of scale: if a network f has a gradient norm of $GN(f) = 1.2$, is it smooth or not? The central issue which our experiments have to tackle is then how to use the measures to obtain interpretable results.

In each experiment, we train several models with varying values of certain hyperparameters and measure the trained models. We then compare the experimental results with our expectations and draw conclusions.

Note that the design of the experiments is very general: they could be used to reason not only about smoothness but also about any other complexity measure, such as distance from initialization or sharpness of the loss function.

4.1 Hyperparameters

First, let us describe the common hyperparameters used in training the networks. We use a batch size of 64, and, unless specified otherwise, a learning rate of 0.01. Each network has a hidden layer size of $h = 256$. This is enough to perfectly fit the training set for the datasets that we experiment with.

We found empirically that training loss is correlated with the smoothness measures, so we fix training loss to avoid spurious correlations: we train each network until it reaches a training loss of 10^{-5}. Additionally, by fixing the training loss, we ensure that we study models that perfectly fit the training set (ideally, we would have 0 training error, but this is not feasible numerically).

Regarding network initialization, the network's biases are initially set to 0. The weights in each of the two layers are initialized by drawing uniformly from the interval $[-\ell, \ell]$, where

$$\ell := \sqrt{\frac{3\alpha}{n_{in} + n_{out}}} \tag{4.1}$$

and n_{in} is the number of input units of the layer, n_{out} is the number of output units, and α is the *initialization scale*. For the first layer, we have $n_{in}^{(1)} = d_{in}$ and $n_{out}^{(1)} = h$, whereas for the second layer, $n_{in}^{(2)} = n_{out}^{(1)} = h$ and $n_{out}^{(2)} = 1$. Recall that d_{in} is the dimensionality of the dataset (e.g. $28 \times 28 = 784$ for MNIST) and h is the size of the hidden layer.

When $\alpha = 1$, the initializer reduces to the widely used Glorot uniform initializer (Glorot and Bengio [2010]). We set $\alpha := 0.01$ in our experiments. The motivation behind scaling the initialization comes from Maennel et al. [2018], who find that in their one-dimensional setup, using a smaller initialization scale leads to a smoother learned function. We find that this initialization scheme makes the trends in the experimental results more evident. We verify in one of the experiments (Subsection 4.3.2) that the general trend holds even when $\alpha = 1$.

4.2 Implementation

The experiments were implemented in Python, using the TensorFlow deep learning framework (Abadi et al. [2016]). To ensure experimental results are not lost, we used the Sacred package (Greff et al. [2017]). We also used the Scipy library (Virtanen et al. [2020]) for various scientific computation. For working with Gaussian processes, we used GPy[1].

We analyzed the results using Jupyter notebooks (Kluyver et al. [2016]). To store and manipulate data, we used Pandas (Reback et al. [2019]). For plotting results, we used the Seaborn plotting library (Waskom et al. [2020]) which builds upon Matplotlib (Hunter [2007]), another plotting library.

The code is available in Attachment A.1, as well as a GitHub repository at `https://github.com/vvolhejn/neural_network_smoothness`.

4.3 Increasing training set size

This experiment relies on the observation that if the smoothest possible function is learned, then adding samples to the training set should lead to a decrease in the smoothness of the learned function because a more complicated function is required to fit the dataset.

More formally, consider a family $\mathcal{F}$ of functions f. This family could be e.g. every possible function expressible with a fixed network architecture. We view the training process as a way of selecting a function $f \in \mathcal{F}$ which best fits a given dataset D. We limit ourselves to the case where it is possible to interpolate D, meaning there exist functions in $\mathcal{F}$ which achieve zero loss on D. Let us denote the set of functions interpolating dataset D by $\mathcal{F}_D$. If $|\mathcal{F}_D| > 1$, we must decide between one of the multiple interpolating functions. If the training process maximizes smoothness (and assuming it is able to find functions with zero loss on D), it selects the smoothest function from $\mathcal{F}_D$.

We can empirically verify whether this is the case. Consider a dataset $D = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)\}$ and its subsets $D_i := \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_i, y_i)\}$. Notice that for $i < j$, we have $\mathcal{F}_{D_i} \supseteq \mathcal{F}_{D_j}$, since $D_i \subseteq D_j$ and any function interpolating a dataset also interpolates its subset. Let us define $f_i \in \mathcal{F}_{D_i}$ as the function learned by the training process for dataset D_i. If the training process minimizes smoothness, then f_i must be at least as smooth as f_j.

This observation is the basis of the following experiment: we select a dataset D of n samples which we are able to interpolate using a function from $\mathcal{F}$. Take the sequence of datasets $D_1, \ldots, D_n$. Denote by $f_1, \ldots, f_n$ the corresponding trained functions, and by S a smoothness measure (again, a lower S means a smoother function). If the training process maximizes smoothness, then we have:

$$S(f_1) \leq S(f_2) \leq \cdots \leq S(f_n) \tag{4.2}$$

Due to noise, not all of the inequalities might hold simultaneously. To quantify how close we are to all inequalities holding, we use the Kendall rank correlation coefficient (Kendall [1938]), defined as a normalized number of inversions in the sequence.

[1] `http://github.com/SheffieldML/GPy`

Definition 9. *Let $X := (x_1, \ldots, x_n), x_i \quad \mathbb{R}$ and $Y := (y_1, \ldots, y_n), y_i \quad \mathbb{R}$ be two sequences.*[2] *The Kendall rank correlation coe cient ("Kendall's τ") is defined as*

$$\tau(X, Y) := \frac{1}{\binom{n}{2}} \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \operatorname{sign}(x_j - x_i) \operatorname{sign}(y_j - y_i) \tag{4.3}$$

When Y is increasing, we may omit it and write simply

$$\tau_{inc}(X) := \frac{1}{\binom{n}{2}} \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \operatorname{sign}(x_j - x_i) \tag{4.4}$$

The definition is simpler to understand if one of the sequences is a canonical ordering, as in Equation (4.4). This simplified version τ_{inc} takes values between -1 and 1, where $\tau_{inc}(X) = 1$ if X is an increasing sequence. The value of $\tau_{inc}(X)$ then decreases for each inversion in X and takes the value -1 if X is decreasing. The original τ itself has analogous properties.

One can also use Kendall's τ in a hypothesis test to determine whether the observation is statistically significant.

In practice, we actually use τ_b, a version of τ with a more complex normalization coefficient which accounts for ties. Since τ normalizes by dividing by $\binom{n}{2}$, it is impossible for $\tau(X, Y)$ to reach 1 when some values are tied in X or Y. The modified τ_b rectifies the issue by dividing instead by $\sqrt{(n_p - n_x)(n_p - n_y)}$, where $n_p := \binom{n}{2}$, and n_x is the number of ties in X, i.e. the number of unordered pairs $\{i, j\}$ such that $x_i = x_j$. The value n_y is defined analogously.

4.3.1 Results for one dimension

We begin by studying simple synthetic one-dimensional datasets. To generate these datasets, we sample functions from Gaussian processes. For an introduction to Gaussian processes, we refer the reader to Rasmussen and Williams [2006]. However, Gaussian processes play a small role in this work, and it is sufficient to think of them as a tool for generating synthetic datasets.

Specifically, we use a Gaussian process with an RBF kernel with variance 1.0. The length scale is 0.5, and the noise variance is 0. The test set consists of 100 equally-spaced samples of the function on the interval $[-1, 1]$. For technical reasons, we also set $f(0) := 0$ when sampling functions. Figure 4.1 shows three functions sampled from a Gaussian process with these parameters.

When selecting the order in which to add samples to the training set, we wish for the samples to be well-spaced. To ensure this, we use the following procedure: Denote by X the set of x values of the samples in D, and X_i the corresponding set of D_i. We set $X_2 = \{-1, 1\}$. Then, X_i for $i > 2$ is defined as $X_i := X_{i-1} \quad x_i$, where x_i is one of the samples from X which have not been used yet. We select x_i such that its distance to any sample from X_{i-1} is the largest possible. In the case of a tie, we select the lowest sample. For instance, if we imagine D to be the entire interval $[-1, 1]$, we would begin by adding these points: $0, -0.5, 0.5, -0.75, -0.25, 0.25, \ldots$

[2] The definition of τ only needs ordinal values (we only need to be able to compare two values), so we could define τ more generally; this is not important for our purposes, however.

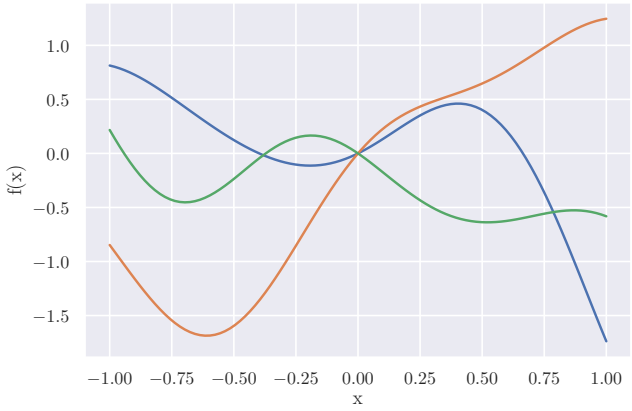


Figure 4.1: Three functions sampled from a Gaussian process.

We sample 20 datasets using this procedure. For each one, we try dataset sizes $\{2, \ldots, 10\}$. The reason we use such small training sets is that due to the dataset's simplicity, adding new points soon stops forcing the functions to be less smooth, and the smoothness measures plateau. To reduce variance, we train 10 networks for each seed and training set size. We train the networks until they reach 10^{-3} training loss. We compute Kendall's τ for each dataset separately. This yields a distribution of τ's.

Because the size of the training set is small, we use "vanilla" gradient descent rather than SGD. We train the networks for 50 000 epochs with a learning rate of 0.1.

We found that the networks did not manage to fit some of the datasets, even though they had so few training samples—this is an anomaly of the one-dimensional setting. Specifically, out of the 20 9 = 180 dataset configurations, none of the 10 trained models reached the desired loss threshold in 23 cases. When calculating τ values, we do not include the unconverged models, though we find that including them does not change the results significantly, skewing the mean τ by at most 0.05.

For comparison, we also calculate our measures when we interpolate the dataset using a high-degree polynomial rather than by using a neural network. For a dataset of k samples, we use a polynomial of degree $k - 1$, so that the solution is unique. Polynomials are known to overfit, and so the smoothness measures should be unstable as we vary the number of training samples. The weights product measure is only defined for two-layer ReLU networks, but in one dimension, we can still calculate it using Equation (3.18).

The results of the experiment are shown in Table 4.1. Polynomials are not as unstable as we could hope to highlight; this might be because the data is simple and noiseless, reducing the risk of overfitting. Nevertheless, the values of τ for polynomials are consistently lower than for neural networks (though there is some variance).

4.3.2 Results for multiple dimensions

To test that the inequalities (4.2) hold even for datasets of higher dimensions, we train on subsets of the MNIST dataset of handwritten digits (LeCun et al.

Measure	NNs	Polynomial
GN	0.72 ± 0.19	0.36 ± 0.20
PL_0	0.62 ± 0.31	0.44 ± 0.22
PL_1	0.79 ± 0.12	0.51 ± 0.14
WP	0.82 ± 0.13	0.38 ± 0.15

Table 4.1: The distribution of the values of Kendall's τ for synthetic datasets, comparing neural networks with polynomial interpolation.

[1998]), limited to some two labels (digits) a and b. We formulate the problem as regression, where the labels are -1 for digit a and 1 for digit b. In this way, we create one dataset for every possible pairs of digits, yielding $\binom{10}{2} = 45$ datasets. Additionally, we balance the classes by truncating the datasets to 10000 examples of each digit. We refer to these datasets as *MNIST-binary* collectively. Having multiple (albeit strongly correlated) datasets allows us to reduce variance and gain more confidence that the observed trends are not just due to randomness.

For each dataset, the training set sizes we try are $N = \{64, 128, 256, 512, 1024, 2048, 4096, 8192\}$. Let us denote by *dataset configuration* a (dataset, training set size) pair. To lower variance, we train three networks for each dataset configuration.

In Table 4.2, we summarize the results. We see that the trend is much stronger in second-order measures (WP and PL_1) than first-order measures. Second order measures consistently achieve the highest value of τ that we can reasonably expect; this is because training three networks for each dataset configuration introduces ties. To reach higher values, we would need two networks trained on the same dataset configuration to reach exactly the same smoothness, which is highly unlikely due to the stochasticity of the training process.

Measure	Kendall's τ
GN	0.15 ± 0.30
PL_0	0.06 ± 0.40
PL_1	0.96
WP	0.96

Table 4.2: The distribution of the values of Kendall's τ for the 45 MNIST-binary datasets. Standard deviation is not listed for WP and PL_1 because these measures reach the same τ for every dataset.

In the previous experiments, we used an initialization scale of $\alpha = 0.01$. To examine the effect of using this smaller initialization, we re-run the preceding experiment with the standard initialization scale of $\alpha = 1$. The results are shown in Table 4.3. The difference between first-order and second-order measures is still visible, though not as pronounced. The explanation could be that by using a lower initialization scale, we are nearing some "idealized behavior" of stochastic gradient descent (SGD), similar to what is observed Maennel et al. [2018]. Compare this effect also to the observation of Nakkiran et al. [2020] that label noise highlights the double descent phenomenon: analogously, a smaller initialization makes the

differences between the smoothness measures more pronounced.

Measure	Kendall's τ
GN	0.77 ± 0.25
PL_0	0.72 ± 0.14
PL_1	0.91 ± 0.08
WP	0.94 ± 0.03

Table 4.3: The distribution of the values of Kendall's τ for the 45 MNIST-binary datasets, with initialization scale α set to 1.

4.3.3 Interpretation

As expected, all four smoothness measures increase as the number of training samples increases. The results show this both absolutely (the values of τ are positive) and relatively (by comparison to polynomials).

Furthermore, the multidimensional case (Table 4.2) highlights the differences between first-order and second-order measures. In this experiment, second-order measures increase perfectly with training set size. We expect this behavior from an implicitly regularized measure.

4.4 Explicit regularization

One way to determine that a complexity measure S is *not* being regularized implicitly during training is to see if we are able to find models that perform equally well on the training set but which are simpler (under this measure) than those found by SGD. If we can find such simpler models, this measure is not being implicitly regularized.

Notice that the converse need not hold: if we are not able to find these simpler models, this does not necessarily mean that they do not exist at all. We were simply not able to find a counterexample but did not show that no counterexamples exist. Jiang et al. [2019] reach the same conclusion when discussing this method.

To search for such simpler models, we use explicit regularization; we replace the original loss function L with its regularized version L_{reg}, in which we penalize high values of S:

$$L_{reg}(f) \coloneqq L(f) + \lambda S(f) \tag{4.5}$$

where $\lambda > 0$ is a regularization coefficient.

The limitation of this approach is that S must be differentiable to allow for the use of SGD. Furthermore, if S is expensive to compute, optimization is slowed down considerably as it is evaluated in every training step. To alleviate this issue, we use stochastic variants of our measures, which are faster to compute.

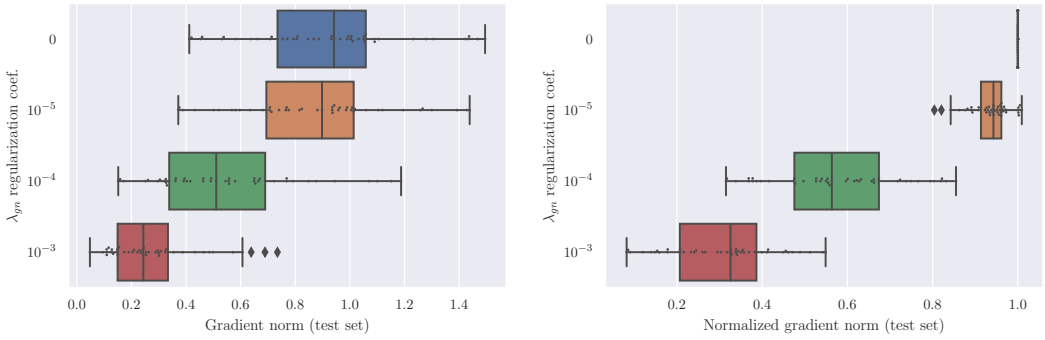
We train the networks on the 45 MNIST-binary datasets to reduce variance. The downside of multiple datasets is that displaying the measures for different

datasets in a single plot might be misleading, as the datasets have different properties (distinguishing between 4 and 9 proved to be more difficult than between 0 and 1, for instance).

We used a learning rate of 0.1 when regularizing the two path length measures and the standard value of 0.01 for the remaining two. The reason we use an increased learning rate for the path length measures is to ensure faster convergence, which is valuable as these measures are computationally expensive.

Note that we formulate the problem as regression despite it essentially being a classification problem. This has the advantage that the network is, in theory, able to fit the training data perfectly, which would not be possible in classification due to the final softmax, which outputs probabilities which are strictly less than 1. Avoiding softmax also has the advantage that the learned function we obtain is piecewise linear, making it easier to reason about. Finally, the justification of the weights product measure also assumes there is no softmax layer.

4.4.1 Gradient norm



(a) Unnormalized (b) Normalized

Figure 4.2: Explicit gradient norm regularization

For the regularization coefficient of gradient norm, we test values $\lambda_{gn} \in \{0, 10^{-5}, 10^{-4}, 10^{-3}\}$. Increasing λ_{gn} further leads to deterioration of training error; models with a larger λ_{gn} did not reach the training loss threshold within a fixed limit of 15000 epochs.

Computing GN is computationally expensive since it is necessary to compute the gradient in the forward pass, so we then have to compute the Hessian during backpropagation. For more efficient regularization, we use a stochastic variant of the gradient norm: when training on a batch $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k$, we approximate the gradient norm as

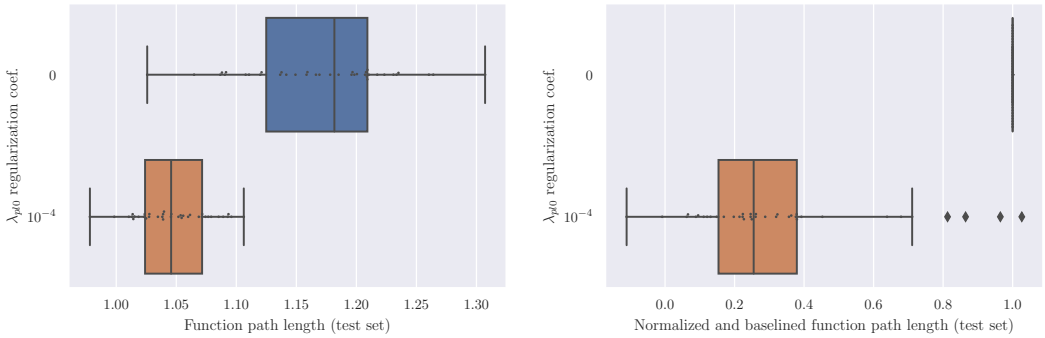
$$\widehat{GN}(f) := \frac{1}{k} \sum_{i=1}^{k} \quad {}_{\boldsymbol{x}} f(\boldsymbol{x}_i) \tag{4.6}$$

whereas normally, we would compute GN by averaging either over the whole training set or the whole test set.

Note that we regularize GN on the training set, and then measure it on the test set. This is true for the other smoothness measures as well. Empirically, the results show that the regularization effect carries over between the two sets.

Figure 4.2 summarizes the results. Each boxplot contains 45 datapoints, one per dataset. The results show that we were able to decrease the gradient norm significantly. In Figure 4.2b, we plot the gradient norm divided by the value achieved by the unregularized model trained on the same dataset (so this value is 1 for each of the unregularized models). We find that we reach a gradient norm of roughly a third of the original value; this reduction allows us to conclude that the gradient norm is *not* being implicitly regularized.

4.4.2 Function path length



(a) Unnormalized (b) Normalized, with subtracted baseline

Figure 4.3: Explicit function path length regularization

As the two path length measures are the most expensive to compute, we approximate them stochastically. Namely, we take the batch $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k$ and split it into $\frac{k}{2}$ pairs, which we use as the endpoints for the segments. Also, whereas normally we approximate the total variation by sampling 100 points along the segment, here we only use 10 points. Formally, this gives us the approximation

$$\widehat{PL_0} := \frac{2}{k} \sum_{i=1}^{k} \widehat{TV}_{[\boldsymbol{x}_i, \boldsymbol{x}_{i+k}]}(f, 10) \tag{4.7}$$

where $\widehat{TV}$ is used as defined in Equation (3.6).

Figure 4.3 summarizes the results of the experiment. For 7 of the 45 datasets, the models did not reach the required loss threshold. For the plots to be comparable between one another, we omit from each the datasets on which the regularized models did not converge. The trend looks qualitatively similar even if we do not omit the datasets.

At first glance, we get a fairly small reduction in unnormalized function path length, to about 90%. However, this is because we are approaching a lower bound for PL_0.

For a model f which attains zero loss on the training set, we may use the lower bound for total variation from Equation (1.15) to bound the (empirical) value of PL_0. Let us denote by $y_1, \ldots, y_n$ the labels corresponding to the training set $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$. Then we may bound the empirical function path length from Equation (3.5) by

$$PL_0(f\ ) \geq \frac{1}{n^2} \sum_{i=1}^{n} \sum_{j=1}^{n} |y_i - y_j| \tag{4.8}$$

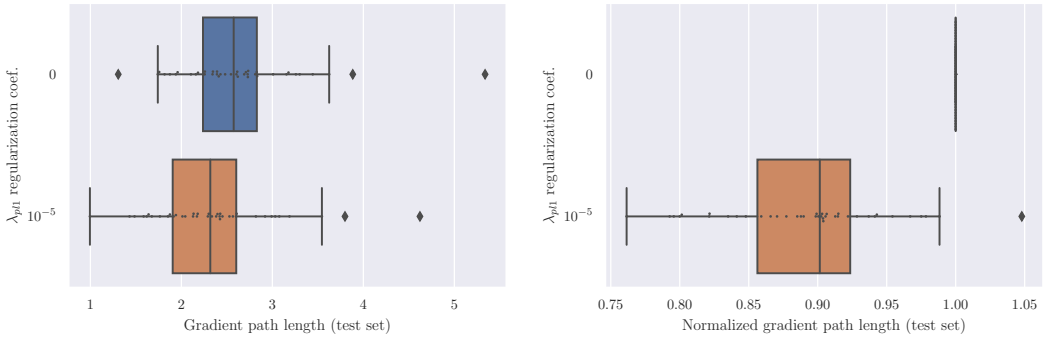
In this specific case, we utilize the fact that in MNIST-binary datasets, half of the dataset has the label -1 and the other half has the label 1. So when sampling pairs from the dataset, with a probability of $\frac{1}{2}$ we get two samples with the same label and with a probability of $\frac{1}{2}$ we get different labels (in which case $|y_1 - y_2| = 2$). This simplifies the bound to

$$PL_0(f\ ) \geq \frac{1}{2}0 + \frac{1}{2}2 = 1$$

Since 1 is the lowest achievable value of PL_0 as opposed to 0 in other measures, it makes sense to normalize the value of $PL_0 - 1$ rather than the original, to see how well we are matching the bound. When we normalize the value in this way, we obtain Figure 4.3b, which shows that the reduction in function path length is actually significant.

The reason why a few models appear to reach a function path length of less than the lower bound of 1 is that we only compute PL_0 approximately.

4.4.3 Gradient path length



(a) Unnormalized (b) Normalized

Figure 4.4: Explicit gradient path length regularization

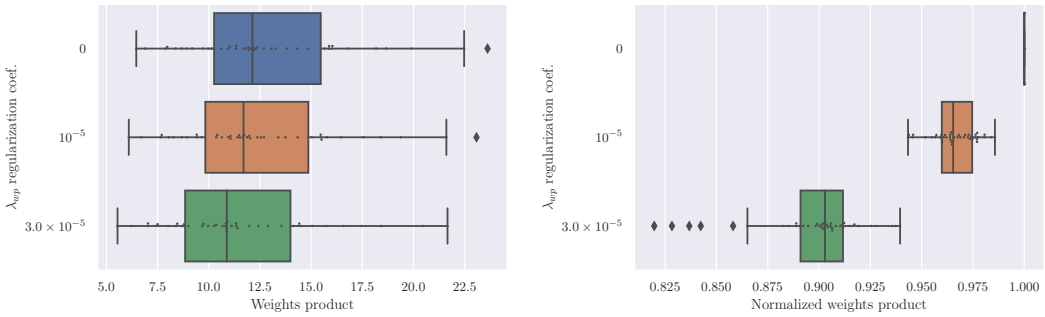
When regularizing gradient path, we again compute it approximately, similar to the function path length. In this case, however, there is no simple lower bound on the value, and so we use the trivial bound $PL_1(f) \geq 0$ when normalizing.

The results in Figure 4.4b show that this measure is not easy to regularize, with the value reaching about 90% of the original. In the figures, we again omit 5 models (out of 45) that did not converge within 25000 epochs.

4.4.4 Weights product

As the weights product is computed purely from the weights, regardless of training data, there is no need for an approximation; it can be computed directly. Thus, in these experiments, we simply add the term $\lambda_{wp} WP(f)$ to the loss. We tried the values $\lambda_{wp} \quad \{0, 10^{-5}, 3 \times 10^{-5}\}$. The largest of these is already significantly slower to train; on 10 of the 45 datasets, the training loss did not reach the required threshold within 25000 epochs.

In Figure 4.5, we report the results of the experiment. Again, we omit from each the 10 datasets on which models with $\lambda_{wp} = 3 \times 10^{-5}$ did not converge.



(a) Unnormalized (b) Normalized

Figure 4.5: Explicit weights product regularization

We were not able to decrease the weights product by a large amount with this method. With the larger λ_{wp}, the value is typically reduced to about 90%, as shown in Figure 4.5b.

4.4.5 Interpretation

Measure	Unreg. mean	Reg. mean	Lower bound	Normalized
GN	0.93	0.29	0	0.31 ± 0.13
PL_0	1.17	1.05	1	0.33 ± 0.27
PL_1	2.62	2.35	0	0.89 ± 0.06
WP	12.84	11.57	0	0.90 ± 0.03

Table 4.4: Explicit regularization experiment results. For each measure, we report the results for the highest regularization coefficient for which we were still able to train the models to achieve 10^{-5} training loss.

Table 4.4 compares the results for different measures. Two of the four measures were easy to decrease: gradient norm and function path length. Note that the *first-order measures* were reduced by explicit regularization, whereas the *second-order measures* were not. Furthermore, results for measures of one category match closely, suggesting that the distinction between first-order and second-order measures is a meaningful one. Thus, we can likely rule out the possibility that a first-order measure is being implicitly regularized. A second-order measure might be implicitly regularized, possibly through a different but correlated measure.

4.5 Discussion

The first experiment leveraged the observation that adding samples to the training set should make the learned function less smooth. We trained networks for several dataset sizes and compared how well the data matches our expectation by using the Kendall rank correlation coefficient. Though all measures display

the expected trend to some extent, it is significantly stronger in second-order measures.

The second experiment was based on the following claim: if a smoothness measure is being regularized implicitly, then it should be difficult to reduce further without adversely affecting the model's performance. Thus, to show that a measure is *not* being regularized implicitly, it is sufficient to be able find these counterexample models. We searched for counterexamples by regularizing our measures *explicitly*, adding a regularization term to the loss.

Using this method, we were able to find models which were much smoother in terms of first-order measures (about 30% of the original value) but could not find much smoother models for second-order measures (we reduced them to about 90%).

The results of the two experiments are consistent: they support the view that it is *not* a first-order measure which is being regularized implicitly. The results for second-order measures, on the other hand, are consistent with what we would expect from an implicitly regularized measure. However, it would be premature to state that one of the second-order measures is the implicit bias that we are searching for.

Our results are consistent with Kubo et al. [2019], who show that a measure similar to the gradient path length (which they dub *gradient deflection*) is kept low by SGD. On the other hand, there is a tension between the failure of first-order smoothness measures in our experiments and the results of Novak et al. [2018]: the authors find that there is correlation between the gradient norm and generalization, but our results suggest that gradient norm is not the implicitly regularized measure. Perhaps the correlation comes from a confounder, another measure that *is* implicitly regularized and also correlates with the gradient norm. Alternatively, this tension might originate from the differences between the experimental setups.

Interestingly, LeJeune et al. [2019] explicitly regularize a measure of "rugosity", which is a second-order smoothness measure. They *are* able to reduce it significantly, in contrast with the results of our experiments with second-order measures. This might be due to the differences in the experimental setup—the authors use CNNs and perform *classification* on CIFAR10—or due to the fact that the original "ideal" rugosity measure is computed via several approximations.

Conclusion

In this work, we empirically studied a hypothesis for explaining the favorable generalization behavior of neural networks. These models do not overfit despite having a very large capacity (Neyshabur et al. [2014], Zhang et al. [2017]): they easily fit the training set perfectly, but unexpectedly, they still generalize well to the test set.

A proposed general explanation is that training using stochastic gradient descent introduces an *implicit bias* that favors functions of low *complexity*, and these generalize well. We investigated whether this complexity measure could be the *smoothness* of the learned function. First, we designed measures of smoothness to be able to quantify it. These measures fall into two categories: first-order and second-order measures. First-order measures are based on the learned function's first derivatives (the gradient), whereas second-order measures are based on the second derivatives (the Hessian).

We proposed two kinds of experiments to determine whether a certain smoothness measure is being implicitly regularized. We showed experimentally that first-order measures are not being implicitly regularized, whereas a second-order measure could be.

4.6 Future work

Further investigation of second-order smoothness measures is a promising direction for further research: one could extend the empirical investigation to more complex architectures and datasets, e.g. ImageNet classification using convolutional neural networks. This poses certain challenges, for instance, using CNNs prevents one from using the weights product measure. Investigating classification (as opposed to regression) also complicates the situation: the learned function is now vector-valued, but perhaps more importantly, it is no longer possible to fit the dataset perfectly. This is because the final softmax produces strictly positive outputs, whereas the dataset has one-hot labels, meaning it requires all probabilities but one to be zero. One could solve this by smoothing the labels or training the models until they reach a certain training loss (as we do here). Furthermore, if we study the learned function of a classifier *after* the final softmax, it is no longer piecewise linear, making it more difficult to reason about.

The experiments performed here to determine whether SGD prefers smooth functions could also be used to analyze other proposed complexity measures, allowing even for comparison between different measures, similar to studies such as Jiang et al. [2019].

Alternatively, one could analyze the situation theoretically with our results in mind to determine whether a second-order smoothness measure is truly being minimized.

Bibliography

Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek Gordon Murray, Benoit Steiner, Paul A. Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: A system for large-scale machine learning. In Kimberly Keeton and Timothy Roscoe, editors, *12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016, Savannah, GA, USA, November 2-4, 2016*, pages 265–283. USENIX Association, 2016. URL https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi.

Madhu S. Advani and Andrew M. Saxe. High-dimensional dynamics of generalization error in neural networks. *CoRR*, abs/1710.03667, 2017. URL http://arxiv.org/abs/1710.03667.

Peter L. Bartlett, Dylan J. Foster, and Matus Telgarsky. Spectrally-normalized margin bounds for neural networks. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 6240–6249, 2017. URL http://papers.nips.cc/paper/7204-spectrally-normalized-margin-bounds-for-neural-networks.

Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mandal. Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences*, 116(32):15849–15854, 2019. ISSN 0027-8424. doi: 10.1073/pnas.1903070116. URL https://www.pnas.org/content/116/32/15849.

Rich Caruana, Steve Lawrence, and C. Lee Giles. Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping. In Todd K. Leen, Thomas G. Dietterich, and Volker Tresp, editors, *Advances in Neural Information Processing Systems 13, Papers from Neural Information Processing Systems (NIPS) 2000, Denver, CO, USA*, pages 402–408. MIT Press, 2000. URL http://papers.nips.cc/paper/1895-overfitting-in-neural-nets-backpropagation-conjugate-gradient-and-early-stopping.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and D. Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2010, Chia Laguna Resort, Sardinia, Italy, May 13-15, 2010*, volume 9 of *JMLR Proceedings*, pages 249–256. JMLR.org, 2010. URL http://proceedings.mlr.press/v9/glorot10a.html.

B. Goldluecke and D. Cremers. An approach to vectorial total variation based on geometric measure theory. In *2010 IEEE Computer Society Conference on*

Computer Vision and Pattern Recognition, pages 327–333, June 2010. doi: 10.1109/CVPR.2010.5540194.

Klaus Greff, Aaron Klein, Martin Chovanec, Frank Hutter, and Jürgen Schmidhuber. The sacred infrastructure for computational research. In *Proceedings of the Python in Science Conferences-SciPy Conferences*, 2017.

J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007. doi: 10.1109/MCSE.2007.55.

Yiding Jiang, Behnam Neyshabur, Hossein Mobahi, Dilip Krishnan, and Samy Bengio. Fantastic generalization measures and where to find them. *CoRR*, abs/1912.02178, 2019. URL `http://arxiv.org/abs/1912.02178`.

Kenji Kawaguchi, Leslie Pack Kaelbling, and Yoshua Bengio. Generalization in deep learning, 2017.

Maurice G Kendall. A new measure of rank correlation. *Biometrika*, 30(1/2): 81–93, 1938.

Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL `https://openreview.net/forum?id=H1oyRlYgg`.

Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian E. Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica B. Hamrick, Jason Grout, Sylvain Corlay, Paul Ivanov, Damián Avila, Safia Abdalla, Carol Willing, and et al. Jupyter notebooks - a publishing format for reproducible computational workflows. In Fernando Loizides and Birgit Schmidt, editors, *Positioning and Power in Academic Publishing: Players, Agents and Agendas, 20th International Conference on Electronic Publishing, Göttingen, Germany, June 7-9, 2016*, pages 87–90. IOS Press, 2016. doi: 10.3233/978-1-61499-649-1-87. URL `https://doi.org/10.3233/978-1-61499-649-1-87`.

Masayoshi Kubo, Ryotaro Banno, Hidetaka Manabe, and Masataka Minoji. Implicit regularization in over-parameterized neural networks, 2019.

Steve Lawrence, C. Lee Gilesyz, and Ah Chung Tsoi. What size neural network gives optimal generalization? convergence properties of backpropagation. Technical report, Institute for Advanced Computer Studies, University of Maryland, 1996.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11): 2278–2324, 1998.

Daniel LeJeune, Randall Balestriero, Hamid Javadi, and Richard G Baraniuk. Implicit rugosity regularization via data augmentation. *arXiv preprint arXiv:1905.11639*, 2019.

Tengyuan Liang, Tomaso A. Poggio, Alexander Rakhlin, and James Stokes. Fisher-rao metric, geometry, and complexity of neural networks. In Kamalika Chaudhuri and Masashi Sugiyama, editors, *The 22nd International Conference on Artificial Intelligence and Statistics, AISTATS 2019, 16-18 April 2019, Naha, Okinawa, Japan*, volume 89 of *Proceedings of Machine Learning Research*, pages 888–896. PMLR, 2019. URL http://proceedings.mlr.press/v89/liang19a.html.

Hartmut Maennel, Olivier Bousquet, and Sylvain Gelly. Gradient descent quantizes relu network features. *arXiv preprint arXiv:1803.08367*, 2018.

Vaishnavh Nagarajan and J. Zico Kolter. Generalization in deep networks: The role of distance from initialization, 2019.

Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In Johannes Fürnkranz and Thorsten Joachims, editors, *Proceedings of the 27th International Conference on Machine Learning (ICML-10), June 21-24, 2010, Haifa, Israel*, pages 807–814. Omnipress, 2010. URL https://icml.cc/Conferences/2010/papers/432.pdf.

Preetum Nakkiran, Gal Kaplun, Yamini Bansal, Tristan Yang, Boaz Barak, and Ilya Sutskever. Deep double descent: Where bigger models and more data hurt. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL https://openreview.net/forum?id=B1g5sA4twr.

Behnam Neyshabur, Ryota Tomioka, and Nathan Srebro. In search of the real inductive bias: On the role of implicit regularization in deep learning. In *ICLR 2015*, 2014.

Behnam Neyshabur, Ryota Tomioka, and Nathan Srebro. Norm-based capacity control in neural networks. In Peter Grünwald, Elad Hazan, and Satyen Kale, editors, *Proceedings of The 28th Conference on Learning Theory, COLT 2015, Paris, France, July 3-6, 2015*, volume 40 of *JMLR Workshop and Conference Proceedings*, pages 1376–1401. JMLR.org, 2015. URL http://proceedings.mlr.press/v40/Neyshabur15.html.

Roman Novak, Yasaman Bahri, Daniel A. Abolafia, Jeffrey Pennington, and Jascha Sohl-Dickstein. Sensitivity and generalization in neural networks: an empirical study. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL https://openreview.net/forum?id=HJC2SzZCW.

Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006.

Jeff Reback, Wes McKinney, Joris Van den Bossche, jbrockmendel, Tom Augspurger, Phillip Cloud, gfyoung, Sinhrks, Adam Klein, Jeff Tratner, Chang She, Matthew Roeschke, William Ayd, Terji Petersen, Andy Hayden, Simon

Hawkins, Jeremy Schendel, Marc Garcia, Vytautas Jancauskas, Pietro Battiston, Skipper Seabold, chris b1, h vetinari, Stephan Hoyer, Wouter Overmeire, Mortada Mehyar, Christopher Whelan, behzad nouri, Thomas Kluyver, and Ka Wo Chen. pandas-dev/pandas: v0.25.3, October 2019. URL https://doi.org/10.5281/zenodo.3524604.

Salah Rifai, Pascal Vincent, Xavier Muller, Xavier Glorot, and Yoshua Bengio. Contractive auto-encoders: Explicit invariance during feature extraction. In Lise Getoor and Tobias Scheffer, editors, *Proceedings of the 28th International Conference on Machine Learning, ICML 2011, Bellevue, Washington, USA, June 28 - July 2, 2011*, pages 833–840. Omnipress, 2011. URL https://icml.cc/2011/papers/455_icmlpaper.pdf.

Pedro H. P. Savarese, Itay Evron, Daniel Soudry, and Nathan Srebro. How do infinite width bounded norm networks look in function space? In Alina Beygelzimer and Daniel Hsu, editors, *Conference on Learning Theory, COLT 2019, 25-28 June 2019, Phoenix, AZ, USA*, volume 99 of *Proceedings of Machine Learning Research*, pages 2667–2690. PMLR, 2019. URL http://proceedings.mlr.press/v99/savarese19a.html.

Jure Sokolic, Raja Giryes, Guillermo Sapiro, and Miguel R. D. Rodrigues. Robust large margin deep neural networks. *IEEE Trans. Signal Process.*, 65(16): 4265–4280, 2017. doi: 10.1109/TSP.2017.2708039. URL https://doi.org/10.1109/TSP.2017.2708039.

S Spigler, M Geiger, S d' Ascoli, L Sagun, G Biroli, and M Wyart. A jamming transition from under- to over-parametrization affects generalization in deep learning. *Journal of Physics A: Mathematical and Theoretical*, 52(47):474001, Oct 2019. ISSN 1751-8121. doi: 10.1088/1751-8121/ab4c8b. URL http://dx.doi.org/10.1088/1751-8121/ab4c8b.

Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, CJ Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake Vand erPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1. 0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. doi: https://doi.org/10.1038/s41592-019-0686-2.

Michael Waskom, Olga Botvinnik, Joel Ostblom, Maoz Gelbart, Saulius Lukauskas, Paul Hobson, David C Gemperline, Tom Augspurger, Yaroslav Halchenko, John B. Cole, Jordi Warmenhoven, Julian de Ruiter, Cameron Pye, Stephan Hoyer, Jake Vanderplas, Santi Villalba, Gero Kunter, Eric Quintero, Pete Bachant, Marcel Martin, Kyle Meyer, Corban Swain, Alistair Miles, Thomas Brunner, Drew O'Kane, Tal Yarkoni, Mike Lee Williams, Constantine Evans, Clark Fitzgerald, and Brian. mwaskom/seaborn: v0.10.1 (april 2020), April 2020. URL https://doi.org/10.5281/zenodo.3767070.

Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL https://openreview.net/forum?id=Sy8gdB9xx.

A. Attachments

A.1 Codebase

The electronic attachment contains the code. Alternatively, the code is also available as a repository on GitHub at `https://github.com/vvolhejn/neural_network_smoothness`. The repository is organized into the following directories:

- `smooth`: Python package for running experiments
- `thesis_data`: data from experiments that appear in the thesis, along with configuration YAML files that can be used to reproduce the experiments.
- `thesis_notebooks`: simple Jupyter notebooks which calculate the reported results from data in `thesis_data`