Learning deep neural networks with very few samples

Christian Fiedler
I hereby declare that this thesis is my own work and that no other sources have been used except those clearly indicated and referenced.

Christian Fiedler, München, 30.07.2019
Abstract

The present work is concerned with extending, analysing and evaluating a recently developed method for data-efficiently approximating components (in particular weights) of feedforward neural networks, generalizing from the case of one and two hidden layers to multiple hidden layers. We assume that the neural networks have linear activation in the output and regular (i.e. differentiable) nonlinear activation functions in the hidden layers and work in the setting of active querying and no noise on the output. First we present and analyse a dimensionality reduction method for reducing the input dimensionality, adapted from previous work on one and two hidden layer networks. Next, an approximation method for a certain matrix subspace, again adapted from previous work to our more general setting, is introduced and analysed, complemented by presenting a novel alternative approximation strategy. Third, utilizing a previously developed approximate projected subgradient algorithm as well as a new optimization scheme weights of the network are recovered from the approximated matrix subspaces.

Zusammenfassung

In der vorliegenden Arbeit wird eine kürzlich vorgestellte Methode zur daten-effizienten Approximation von Komponenten (insbesondere Gewichten) von Feedforward Neural Networks vom Fall ein bzw. zwei verdeckter Schichten auf mehrere erweitert, analysiert und evaluiert. Wir beschränken uns auf Netzwerke mit linearer Aktivierungsfunktion am Ausgang und regulären (d.h. differenzierbaren) Aktivierungsfunktionen in den verdeckten Schichten und nehmen an, dass Eingangsdaten des Netzwerks vorgegeben werden können und keine Fehler bei den Ausgangsdaten vorliegen. Zunächst wird eine für ein und zwei verdeckte Schichten entwickelte Dimensionsreduktionsmethode auf unseren allgemeineren Fall angepasst. Danach wird eine Approximationsmethode für bestimmte Matrizenunterräume, ebenfalls basierend auf Vorarbeiten, vorgestellt und analysiert sowie eine neue Alternative präsentiert. Schließlich wird ein bereits bekannter approximate projected subgradient Algorithmus sowie als Alternative ein neuer Optimierungsansatz zur Approximation von Netzwerkgewichten aus den Matrizenunterräumen angewendet.
Preface

In this Master’s thesis we are concerned with generalizations and extensions of a novel method for recovery of weights from shallow neural networks (i.e. with only one hidden layer) with a certain architecture [34]. The present work follows and extends the Master’s thesis [82] which presented partial generalizations to neural networks with two hidden layers.

For convenience we collect our contributions in the present work:

- Generalization of the dimensionality reduction scheme from [34] and [82] from neural networks with one and two hidden layers, respectively, to arbitrary deep networks (Chapter 2)

- Generalization of the structural result [82, (2.0.17)] from two hidden layers to arbitrary depths (Section 1.4).

- Generalization of [82, Algorithm 1] from two hidden layers to arbitrary depths (Section 3.1) and analysing the generalized schemes (Sections 3.2, 3.3). Furthermore, we made the analysis from [82, Section 3.3] more modular and provided additional guidance.

- Proposing an alternative approximation scheme (essentially an extension of [82, Algorithm 1]). We derive it heuristically (and based on observations from numerical experiments) and perform first numerical experiments to evaluate it (Section 3.4).

- Proposing an alternative approach for distinguishing weights and mixed weights (Section 4.5).

- Proposing a simple alternative weight recovery scheme based on optimization with Riemannian manifolds. We motivate it heuristically, provide a formal derivation and perform numerical experiments evaluating it (Sections 4.3, 4.4).

A detailed overview of the complete work can be found in Section 1.2. Open questions and limitations of the present work will be collected in Chapter 5 which also includes a concise summary of our results. During the finalization of the present work the preprint [32] was made available which contains modified concentration results (relevant to our Chapter 3), a modified version of Algorithm 4.2 tailored to the more general setting of [82], [32] and this work, together with a detailed convergence analysis (relevant to our Chapter 4). Furthermore, some of the open questions (cf. Chapter 5) have been dealt with empirically in [32] for the case of two hidden layers. Due to time and space constraints we did not adapt the present work to [32] and leave this to future work.
Acknowledgments

First and foremost I would like to sincerely thank my supervisor and advisor Prof. Dr. Massimo Fornasier for introducing me to this fascinating topic and letting me work on such a recent and innovative research question. During and from the work on this thesis I really learned and gained a lot, including getting to know mathematical concepts new to me, a deeper understanding of familiar concepts, extensive practice in doing actual mathematical research and performing numerical experiments, getting to know (and hopefully having improved upon) my weaknesses and shortcomings in communicating and presenting ideas and results, and organizing and structuring extensive work over a longer period of time. All of this would certainly not have been possible without his excellent and helpful supervision. Furthermore, I am very grateful for his constant support, patience and encouragement even during rather difficult times while working on this thesis.

In addition I would like to thank Michael Rauchensteiner for introducing me to his work, numerous helpful discussions, checking scripts for numerical experiments, reading and commenting on drafts of this text and supporting me throughout the work on this thesis. I would also like to thank Timo Klock for interesting and helpful discussions.

Finally I would like to thank and express my deep gratitude to my friends and especially my family for their constant and steady support, patience and encouragement, even during the most stressful and difficult times.
# Contents

1. **Introduction**  
   1.1. Background ................................................. 2  
      1.1.1. Artificial neural networks ............................... 2  
      1.1.2. Learning with artificial neural networks ................. 3  
      1.1.3. Theoretical perspectives on learning with neural networks 4  
   1.2. Overview and summary ....................................... 6  
   1.3. Notation and important definitions .......................... 7  
   1.4. The starting point: Structure of derivatives ............... 9  

2. **Dimension Reduction in the input layer**  
   2.1. Dimension Reduction with approximate span of weights in first layer 18  
   2.2. Approximating the span of first layer weights .................. 20  
   2.3. Numerical experiments ...................................... 28  

3. **Learning the auxiliary matrix space**  
   3.1. Introduction and setup .................................... 32  
   3.2. Concentration of Hessians .................................. 34  
   3.3. Approximation by finite differences ........................ 52  
   3.4. Alternative approach ...................................... 55  

4. **Recovering weights from the auxiliary matrix space**  
   4.1. Introduction and overview .................................. 62  
   4.2. Weight recovery using an approximate projected gradient algorithm 63  
   4.3. Weight recovery using gradient ascent on the unit sphere ....... 66  
   4.4. Numerical experiments: Recovering weights from full auxiliary space 72  
   4.5. Distinguishing weights and mixed weights .................... 80  
   4.6. Numerical experiments: Weight recovery from alternative auxiliary space 87  

5. **Conclusion** ............................................... 89  

S. **Supplementary material** ..................................... 93  
   S.1. Background material from Linear Algebra and Matrix Analysis .... 93  
      S.1.1. Singular values and singular vectors ........................ 93  
      S.1.2. Norms .................................................... 94  
      S.1.3. Vectorization ............................................ 95  
      S.1.4. Near-orthogonality ...................................... 96  
   S.2. Background material from probability theory ................ 97  
      S.2.1. Measure concentration for scalar random variables .......... 97
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>S.2.2.</td>
<td>Measure concentration for random matrices</td>
<td>99</td>
</tr>
<tr>
<td>S.2.3.</td>
<td>Technical results for selected distributions</td>
<td>101</td>
</tr>
<tr>
<td>S.3.</td>
<td>Technical auxiliary results</td>
<td>104</td>
</tr>
<tr>
<td>S.4.</td>
<td>Guide to notation</td>
<td>111</td>
</tr>
</tbody>
</table>

References 112
1. Introduction

Machine Learning is currently a very active discipline [59], with successful applications in a variety of fields including computer vision, natural language processing, bioinformatics and control, see e.g. [46]. A particularly popular subfield is Deep Learning which is responsible for many of recent practical successes of machine learning [65]. Impressive examples can be found in computer vision, including automatic generation of image descriptions [104], image style transfer [39], image inpainting and denoising [103], [105]. Other recent successes of deep learning can be found in reinforcement learning [73], [86] (see [6] for a survey), as well as natural language processing, bioinformatics applications [71], [8] and various other domains where deep learning approaches often achieve state of the art performances, see e.g. [85, Section 5.22]. However, despite its practical success in various domains deep learning still faces a large amount of open issues and questions, especially of theoretical nature. Among these problems are robustness issues like susceptibility to adversarial attacks [94], [42], [62] (see [106] for a recent review and [18] for a review placing this phenomena in the broader context of adversarial machine learning), overconfidence of the trained network [47] or the nature of generalization in neural networks [107]. However, note that the last issue is currently (re-)investigated in machine learning in general, see for example [12], [102]. Another open issue is a thorough understanding of the learning process itself for (deep) neural networks. Currently mostly variants of Stochastic Gradient Descent (SGD) together with Backpropagation (BP) are used to often great success [65], but this relies mostly on empirical experience and heuristics. A mathematically rigorous theory and a deep understanding of the learning process, as available for example for Support Vector Machines (SVM) [91], is only starting to emerge, see e.g. [79] for developments in this direction.

The present work is considered with developing and analysing learning algorithms for neural networks in a particular simplified setting. Although our setting will be rather different from standard supervised or unsupervised learning with deep neural networks it allows a clear understanding of the learning process as well as deriving theoretical learning guarantees. Of particular interest will be the data-efficiency, i.e. how many samples are required for successful learning. In contemporary deep learning SGD-based algorithms frequently need huge amounts of data (often in the range of millions of samples or more) and hence obvious questions arise: Is so much data, are so many samples really needed? Are the underlying learning problems so difficult that huge amounts of data are needed? Or are the learning algorithms, SGD and its variants in particular, just very data-inefficient? Presenting practical data-efficient learning algorithms for neural networks - albeit in a simplified setting - might be a first step towards theoretically grounded answers to these questions [34].
1.1. Background

In this section a brief overview of learning with (artificial) neural networks is given. Different types of neural networks are introduced and the type of interest to us - **feedforward neural networks** - is presented in details. In addition a brief review of standard learning algorithms for neural networks is given.

1.1.1. Artificial neural networks

The computational and mathematical study of artificial neural networks originated from research concerned with biological neural networks. On a phenomenological level the serious study of neurons in biology started with the work of Ramon y Cajal in the 19th century, resulting in the neuron doctrine [67]. Early important theoretical work started in the first half of the 20th century with the McCulloch-Pitts-model [69][1] and the Hodgkin-Huxley-equations describing spike generation in neurons [55]. An important step towards modern learning with neural networks was made with the Perceptron of Rosenblatt [83], [75], though its limitations lead to some discouragement [72], [85, Section 5.5]. There has been a proliferation of different neural models, some inspired by and closely related to biological neurons, some rather removed from biology, see [85] for a comprehensive and insightful overview of the history of neural networks. Some of the more successful models that still play an important role in machine learning today are feedforward neural networks, recurrent neural networks, Boltzmann machines [52] and the RBM generalizations Restricted Boltzmann Machines (RBM) [88] and Deep Belief Networks (DBN) [50], [51]. In the following only feedforward neural networks, also known as Multilayer Perceptrons (MLPs) will be considered. This is a natural choice since it is a rather simple yet powerful model and also the most popular type in current machine learning [41, Chapter 6]. Furthermore, more complicated models can be seen as extensions of feedforward neural networks, e.g. recurrent neural networks are a natural extensions of feedforward neural networks to sequence inputs [41, Chapter 10]. Note that feedforward neural networks as well as most other neural network models considered in machine learning are rather removed from any biological basis, though there are some attempts to reconnect them to biological neural networks, see e.g. [85, Section 5.26] and [15].

Feedforward neural networks can be seen as a representation of a certain class of functions. A single “neuron” corresponds to a computational unit consisting of a predefined function known as an activation function or less-precisely a nonlinearity that receives a weighted sum of inputs from other neurons. This can be interpreted as a very crude simplification of the biological situation of a neuron receiving inputs from other neurons via a synapse. The inputs to a given neural network, say \( x_1, \ldots, x_d \in \mathbb{R} \), are usually interpreted as input neurons. Formally, the action of a neuron on its inputs \( y_1, \ldots, y_N \)

\[ y_i = f \left( \sum_{j=1}^{d} w_{ij} x_j \right) \]

where \( w_{ij} \) are the weights of the connections from neuron \( j \) to neuron \( i \), and \( f \) is the activation function.

Note that this work did not consider learning, the first investigations into learning on a neuronal level seems to be the work of Hebb (Hebbian learning) [49], [85, Section 5.1]
can hence be described as

$$g \left( \sum_{i=1}^{N} a_i y_i \right),$$

where $g$ is the activation function and $a_1, \ldots, a_N \in \mathbb{R}$ are the weights. Note that usually activation functions are of the form $g(s) = \phi(s + b)$ where $\phi$ is a known function and $b \in \mathbb{R}$ is called the bias of the neuron. Common choices of $\phi$ are linear functions, rectifier linear units (ReLU) and saturating sigmoidal functions like $\tanh$ and the sigmoid function. For more details, comparisons and illustrations cf. e.g. [?, Section 6.2, 6.3]. In feedforward neural networks the neurons are connected in a feedforward manner and hence it can be described by an acyclic weighted directed graph where the edge weights correspond to the connection weights. Note in particular that there are no reverse connections and hence the output of the network given certain inputs can be computed by simply propagating the input in a forward manner through the network. The function represented by such a feedforward neural network is structurally easy to describe, though the notation gets necessarily complicated, see Section 1.3 for details. Note that a function represented by a feedforward neural network can always be represented by a neural network consisting of distinct layers, cf. [41, Chapter 6]. The number of layers is referred to as the depth of a neural network, the number of neurons in layer as the width of the network. Note that sometimes the “input layer” is counted as layer and the internal layers are referred to as hidden layers. For a detailed exposition of our notation and nomenclature we refer again to Section 1.3. The architecture and in particular the type of the last activation function depends on the kind of task to be solved by the neural network and specific choices lead to well-known special cases. For more examples of activation functions and as well as a thorough discussion we refer to [41, Section 6.3].

### 1.1.2. Learning with artificial neural networks

As discussed in the preceding section a feedforward neural network represents a certain function. Given an architecture (i.e. number of layers, width of layers and types of activations) with undetermined weights (and potentially biases) a neural network can be interpreted as a parametric model in machine learning. Learning means then finding good weights for the task at hand from given data. Currently supervised learning with neural networks mostly done in the statistical learning theory framework, i.e. performing empirical (regularized) risk minimization with respect to a suitable loss function encoding the learning goal (see e.g. [41, Chapter 5] for a high-level overview and [91, Chapters 1,2,3] for an in-depth perspective). The standard approach to actually perform the risk minimization is to use stochastic gradient descent together with an efficient way to calculate the gradient of the neural network with respect to its weights (and biases) [84] (see [85] for an in-depth history as well as an overview of alternative approaches and [41, Chapter 8] for a modern introduction).
1.1.3. Theoretical perspectives on learning with neural networks

Approximation properties of feedforward neural networks

The classical approximation theory for functions represented by a (feedforward) neural network is well-developed. Since the late 1980s and early 1990s it is well-known that even feedforward neural networks with only one layer enjoy universal approximation properties, [27], [38], [57], [66]. Also questions like approximation of derivatives [58] or non-asymptotic questions like rates [9] have been settled and we refer to [77] for a thorough review of the classical results.

There are also various approximation theoretic results for modern types of neural networks, e.g. concerning the expressive power of Restricted Boltzmann Machines and Deep Belief Networks [63], [93], [64]. Note that the setting of these results is different from the classical approximation results mentioned at the beginning of this section.

The goal of the present work is to make advances towards generalizing the approach of [34] and [82] from one hidden layer (respectively two hidden layers) towards arbitrary depths (potentially with additional assumptions on the network). Hence the question arises what changes from an approximation-theoretic perspective since already networks with only one hidden layer are universal, cf. the remarks at the beginning of this section.

The investigation of advantages of depth of neural networks is currently a very active field of research. First results indicate that for some relevant function classes there are indeed approximation theoretic advantages of depth, cf. for example [77, Section 7] for early investigations in this direction and [80], [70] for a contemporary perspective. Furthermore, depth might help with high dimensional input spaces and therefore can alleviate the curse of dimensionality, cf. [13, Section 3.2], [41, Section 6.4.1, Section 15.5] and references therein. Note that recently there has been some activity in using techniques from harmonic analysis to investigate modern neural networks, some examples are [45], [101].

Learning neural networks and generalization

Despite the practical success of deep learning, in particular SGD-based learning algorithms, many theoretical questions remain unanswered. In particular, backpropagation based approaches faces various fundamental challenges. First of all, from a computational complexity point of view learning a neural network is a very difficult problem. Even for very simple settings and very simple architectures NP-completeness results exists, [20], [61], [87]. Even allowing for approximation errors hardness results exists, e.g. [10], [11], [89].

Since the empirical risk landscape can be highly nonconvex gradient-based methods might get stuck in bad local minima. Indeed, there is both theoretical [90], [7], [37] as well as empirical [22], [44] evidence for this problem. However, there is evidence that the situation is qualitatively different for modern neural networks, cf. e.g. [43] for empirical and [28], [24] for theoretical investigations. Note that closely related to the problem of bad local minima is the question of generalization.

Apart from potentially bad local minima the learning process with gradients itself
can be problematic, especially for deep neural networks. Investigations starting in the early 1990s [53], [16], [54], [76] discovered that the gradient propagation in deep (or recurrent) neural networks can lead to a vanishing or exploding gradient. For a comprehensive overview of methods to mitigate this problem we refer to [85, Section 5.9]. Note that there are other gradient-related problems like saddle points or cliffs, cf. e.g. [41, Section 8.2] for an overview. The success of deep learning (in the contemporary sense) started in the early 2000s when layer-wise methods were employed, [50], [14], [81], [51] and we refer to [41, Section 15.1] for a practical modern view. Note that due to advances in initialization strategies, optimization algorithms and network architecture design (see e.g. [41, Sections 8.2-8.7]) nowadays supervised training of deep neural networks is usually done without layerwise pretraining [41, Section 15.1], i.e. in practice the phenomena presented above are not insurmountable challenges to learning with deep neural networks.

Identifying neural networks from input-output-samples

Most of deep learning research, especially when oriented toward applications, is focused on an statistical learning theory perspective [41, Chapter 5] (which mostly leads to regularized empirical risk minimization problems [41, Chapter 8]) or generative modelling perspective [41, Chapter 20]. However, the point of view relevant for this and related work is different. As already discussed above, a fully specified feedforward neural network represents a real multivariate function, i.e. it implements a function $f : D \rightarrow \mathbb{R}$, $D \subseteq \mathbb{R}^d$, with $d \geq 1$ often rather large. Apart from approximation theoretic questions (i.e. which properties has a function represented by a feedforward neural network) some rather obvious questions arise: Given a function $f$ represented by a feedforward neural network, what can be said about the underlying neural network? For example, is its architecture (number of layers and their widths, types of activation functions) uniquely determined by the function? Or given a fixed architecture, are the weights uniquely determined? If so, can they be inferred from input-output-samples $(x_i, f(x_i))$?

In the special case of feedforward neural networks with $tanh$-activation functions very strong uniqueness results for architecture (number of layers and their widths), weights and biases have been derived in [29], [30]. However, the question how to constructively (and also in an robust and efficient manner) reconstruct e.g. the weights from input-output-samples has been left open.

As a special case sums of ridge functions have received a lot of attention (see e.g. [78] for an overview). Relevant to the present setting is [23] where higher order differentiation has been used to extract weights from sums of ridge functions. [25] used (finite difference) differentiation and also tackled the question of sample complexity. [33] introduced a universal (non-adaptive) sampling scheme for a related model and finally [35] introduced the method and analysis used in the present work. Whereas [35] considered only shallow neural networks (i.e. sum of ridge functions) [82] started to extend the approach to two hidden layers. The latter work will be our starting point. Note that originally only active

---

querying (mostly without noise) has been considered in most of these works, however, in the latest version of [35] also passive sampling and noise has been considered. Note that a very different approach (tensor method for initialization of gradient descent) has been used in [109] to arrive at recovery guarantees (with respect to weights and biases).

1.2. Overview and summary

We now give a short summary of the present work and sketch the general structure. In the remainder of this introduction we give an overview of the notation used (Section 1.3) and investigate the structure of first (gradient) and second order (Hessian) derivatives of the class of neural networks considered here (Section 1.4). This will be, very much like in the case of [35] and [82], the starting point for all following developments. Note that in the remainder we focus on the setting of active querying without noise, i.e. we assume that we can evaluate the neural network at hand at arbitrary (admissible) inputs and the true evaluations of the network (without noise) are available.

In Chapter 2 the structure of the gradients will be used to devise an efficient dimensionality reduction scheme which is a direct adaption of the work in [35, Chapter 2]. We first recall that having (an approximation of) the span of first layer weights leads to a simple reduction scheme in the input dimension (Section 2.1). Assuming an active querying setting without noise we generalize the approach and analysis from [35, Section 2.2.1] from one to several hidden layers (Section 2.2). Note that the analysis requires some elementary but very lengthy calculations which therefore have been moved to the supplementary material at the end of this work (Section S.3). If the precise form of the constants is not of interest the corresponding section of the supplementary material can be skipped without loss of continuity. Furthermore some numerical experiments can be found in Section 2.3.

In the case of only one hidden layer the Hessians have a particularly simple structure which allows the approximation of a certain matrix auxiliary subspace. Under appropriate assumption the weights (from the single hidden layer) can be retrieved (approximately) from this auxiliary space. However, with two or more hidden layers the structure of the Hessians becomes more complicated. For two hidden layers [82] showed that a (now more complicated) auxiliary space can be approximated which could be used for retrieving the weights. In Chapter 3 we introduce the corresponding auxiliary space and an approximation approach for two or more layers. In particular, we generalize the concentration results from [82, Chapter 3] to our setting (Sections 3.2 and 3.3). However, since the more complicated setting poses various challenges we propose a different approximation scheme which should lead to a simpler (and more amenable to weight recovery, cf. the following chapter) auxiliary space and higher approximation accuracy (Section 3.4). We motivate and explain this scheme heuristically and based on numerical evidence. Furthermore, instead of rigorous approximation guarantees we restrict ourselves to numerical experiments. A precise mathematical analysis (and potential improvements arising therefrom) are left to future work.
Finally, assuming an approximation of the auxiliary space from Chapter 3 (or the alternative approximation proposed there) some approaches to extract the weights (or rather directions corresponding to weights) are investigated in Chapter 4. Similar to \[82, Chapter 5\] we simply use the approach from \[35, Chapter 3\] to recover the weights from the approximated auxiliary space (Section 4.1). Since the approach considered there was originally developed for a single hidden layer we follow \[82, Chapter 5\] and instead of a rigorous analysis focus only on numerical experiments. For this we recall a simple iterative algorithm from \[35\] (essentially an approximate projected subgradient algorithm) and provide a formal derivation for completeness (Section 4.2). Next, we propose an alternative algorithm (essentially gradient ascent on the unit sphere interpreted as a Riemannian manifold), first motivating it heuristically and then deriving it formally (Section 4.3). Again, for now we will focus on numerical experiments and do not provide rigorous guarantees. A precise mathematical analysis is left to future work. We close the chapter with the results of various numerical experiments of the two recovery approaches (Section 4.4).

We conclude the present work with a short summary, discussion of the results and observations as well as interesting open questions (and possible ways to tackle them) in Chapter 5.

Chapter S contains background material as well as technical results used in the main text and some extensions.

1.3. Notation and important definitions

General notation and symbols

We use mostly standard definitions and notations. In particular we use \(\mathbb{N} = \{1, 2, 3, 4 \ldots\}\) for the natural numbers (for convenience we do not include 0 in \(\mathbb{N}\)) and \(\mathbb{N}_0 = \{0, 1, 2, 3, \ldots\}\) the natural numbers with 0, and \(\mathbb{R}\) the real numbers. We denote by \(\mathbb{R}^n\) the \(n\)-dimensional Euclidean space and assume the \(\| \cdot \|_2\)-norm on it. Vectors \(x \in \mathbb{R}^n\) are always interpreted as column vectors (or \(n \times 1\) matrices)\(^3\) and hence \(x^T\) is interpreted as a row-vector (or \(1 \times n\) matrix). The space of all \(m \times n\)-matrices, \(m, n \in \mathbb{N}\), is denoted by \(\mathbb{R}^{m \times n}\) and when required we will explicitly state the norm used on this space (usually the \(\| \cdot \|_F\)-norm). Given a matrix \(A \in \mathbb{R}^{m \times n}\) we denote by \(A_{ij}\) the entry in the \(i\)-th row and \(j\)-th column, by \(A_j\) the \(j\)-th column of \(A\) (a column vector) and by \(A_i\) the \(i\)-th row (a row-vector). Let \(d \in \mathbb{R}^n\) be a vector, then \(\text{diag}(d) = \text{diag}(d, i = 1, \ldots, n)\) is a \(n \times n\) diagonal matrix with \(d\) on its diagonal, i.e.

\[
\text{diag}(d)_{ij} = \text{diag}(d, i = 1, \ldots, n)_{ij} = \begin{cases} d_i & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}
\]

Compare also with the notation preceding Definition S.1. As already demonstrated we will sometimes omit the ranges of indices if no confusion can arise. However, we will

\(^3\)With this convention matrix-vector and matrix-matrix-multiplication are compatible with each other
never write $A_{ij}$ for $A$ (i.e. running implicitly over the indices), an expression like $A_{ij}$ always refers to the element in row $i$ and column $j$. Given a vector space $V$ and a subset $A \subseteq V$ we denote by $spanA$ (or if $A = \{v_1, \ldots, v_N\}$ similarly $span\{v_1, \ldots, v_N\}$) the linear span of the subset (we usually work only with real vector spaces, so no ambiguities regarding the underlying field arise). We use a variety of different norms, for the notation used as well as basic properties we refer to Section S.1.2. Furthermore, frequently vectorization and its inverse (devectorization or matrication) are used, for the corresponding notation and basic properties cf. Section S.1.3. We will frequently encounter very lengthy expressions and in order to simplify the notation and avoid special cases we assume the usual conventions
\[ \sum_{i=n_0}^{n_1} = 0, \quad \prod_{i=n_0}^{n_1} = 1 \]
for $n_0 > n_1$.

**Feedforward neural networks with linear output and unit scalings**

Although neural networks are rather simple and highly structured objects, describing them formally can lead to cumbersome and confusing notation. Here we try to find the right balance between generality and convenience. The formal definition of neural networks in the form used here is given next.

**Definition 1.1. Neural Network**

A (feedforward) neural network with $L \in \mathbb{N}$ hidden layers is a function of the form
\[ f : D \to \mathbb{R}, \quad x \mapsto \sum_{l=1}^{m[L]} y_{i_{L}}^{(L)}(x), \]
where $D \subseteq \mathbb{R}^d$ is the domain of the neural network function, $d \in \mathbb{N}$ the input dimension and we set
\[ y_{i_{L}}^{(l)}(x) = g_{i_{L}}^{(l)} \left[ z_{i_{L}}^{(l)}(x) \right], \quad l = 1, \ldots, L, \quad i_{l} = 1, \ldots, m^{[l]} \]
\[ y_{i_{0}}^{(0)}(x) = x_{i_{0}} \quad \text{if} \quad i_{0} \in [m^{[0]}], \quad m^{[0]} = d \]
with $m^{[l]} \in \mathbb{N}$ being the number of neurons in layer $l$. Furthermore,
\[ g^{[l]} : D^{[l]} \to \mathbb{R}^{m^{[l]}}, \quad l = 1, \ldots, L \]
are the nonlinearities (or activation functions) in layer $l$ with domain $D^{[l]} \subseteq \mathbb{R}^{m^{[l]}}$ and
\[ z_{i_{l}}^{[l]}(x) = \langle a_{i_{l}}^{[l]}, g^{[l-1]}(x) \rangle, \]
where $a_{i_{l}}^{[l]} \in \mathbb{R}^{m^{[l-1]}}$ are the weights (of neuron $i_{l}$ in layer $l$), collected in matrices
\[ W^{[l]} = \begin{pmatrix} a_{1}^{[l]} \phantom{\ldots} \vdots \phantom{\ldots} a_{m^{[l]}}^{[l]} \end{pmatrix} \in \mathbb{R}^{m^{[l]} \times m^{[l-1]}} \]
Remark 1.2. 1. Often all components $g_{i_t}^{[l]}$ of the nonlinearity in a given hidden layer are the same or just translates of each other, i.e. they are of the form $g_{i_t}^{[l]}(r) = \phi^{[l]}(s + b_{i_t}^{[l]})$, where $b_{i_t}^{[l]}$ is called a bias. However, we do not make this assumption, unless explicitly noted.

2. Using the notation above $f$ can also be compactly and intuitively written as

$$f(x) = 1^T g^{[L]}(W^{[L]}g^{[L-1]}(W^{[L-1]}g^{[L-2]}(\ldots)))$$

In the following we are mainly interested in differentiable activation functions, so we assume implicitly that all used derivatives exist and are continuous. Furthermore, unless otherwise noted, the domain of all activation functions is of the form

$$D^{[l]} = D_1^{[l]} \times \ldots D_i^{[l]}$$

for $D_i^{[l]} \subseteq \mathbb{R}$. Boundedness of the activation functions and their derivatives will be essential, so for convenience we define

$$\kappa_k^{[l]} = \max_{i=1, \ldots, m^{[l]}} \sup_{x \in D_i^{[l]}} \left| \left( g_{i_t}^{[l]}(s) \right)^k \right|$$

Assumptions and setting

Unless noted otherwise we will consider functions $f$ represented by feedforward neural network in the sense of Definition 1.1. The meaning of the various components of a neural network (described in the preceding section) should be clear from the context. We will assume throughout the text that $m^{[0]} \geq m^{[1]}$, $L \geq 1$ (in most cases we are interested in $L \geq 2$) and that all domains $D^{[l]}$ are non-empty and well-defined (i.e. the weighted outputs from previous layers are always assumed to be included in the domains of the following layer). Furthermore, all activation functions are assumed to be twice continuously differentiable with Lipschitz continuous second derivatives (in particular the bounds $\kappa_k^{[l]}$ will always be available). We implicitly assume for this that all activation functions are actually defined on open sets containing the respective domains. In addition we assume usually that the weights are normalized (i.e. $\|a_{i_t}^{[l]}\|_2 \leq 1$) and linearly independent (it should always be clear when these assumption are not necessary). Furthermore, since we often assume that the weights are nearly orthogonal we assume unless noted otherwise that $m^{[l]} \geq m^{[l+1]}$ for $l = 0, \ldots, L-1$, i.e. the networks considered here have pyramidal shape.

1.4. The starting point: Structure of derivatives

The starting point for all following developments is the structure of the gradient and Hessian of a neural network function. Note that the following results are direct generalizations of [34] and [82, Chapter 2] from $L = 1$ and $L = 2$ to arbitrary $L$. 


Gradient

**Proposition 1.3. Partial derivate of Neural Network**

We have

\[ \frac{\partial}{\partial x_i} y^{[L]}(x) = G^{[L]}(x)W^{[L]}G^{[L-1]}(x) \ldots G^{[1]}(x)W^{[1]}_i \]

where

\[ G^{[l]}(x) = \text{diag} \left( \left( g^{[l]}_{i_1} \right)' \left[ z^{[l]}_{i_1}(x) \right], \, i_1 = 1, \ldots, m_l \right), \quad l = 1, \ldots, L \]

**Proof.** Use induction over \( L \).

\( L = 1 \) For all \( i = 1, \ldots, m^{[0]} \) one simply gets

\[ \frac{\partial}{\partial x_i} y^{[1]}(x) = \left( (g^{[1]}_{i_1})'\left[ z^{[1]}_{i_1}(x) \right](a^{[1]}_{i_1})_i \right)_{i_1=1,\ldots,m^{[1]}} = \text{diag} \left( \left( g^{[1]}_{i_1} \right)' \left[ z^{[1]}_{i_1}(x) \right], \, i_1 = 1, \ldots, m^{[l]} \right) W^{[1]}_i \]

\( L - 1 \rightarrow L \) For all \( i = 1, \ldots, m^{[0]} \) we have

\[ \frac{\partial}{\partial x_i} y^{[L]}(x) = \left( \left( g^{[L]}_{i_L} \right)' \left[ z^{[L]}_{i_L}(x) \right] \left( a^{[L]}_{i_L} \frac{\partial}{\partial x_i} y^{[L-1]}(x) \right) \right)_{i_L=1,\ldots,m^{[L]}} \]

\[ = \left( \left( g^{[L]}_{i_L} \right)' \left[ z^{[L]}_{i_L}(x) \right] W^{[L]}_{i_L} \frac{\partial}{\partial x_i} y^{[L]}(x) \right)_{i_L=1,\ldots,m^{[L]}} \]

\[ = \left( \left( g^{[L]}_{i_L} \right)' \left[ z^{[L]}_{i_L}(x) \right] W^{[L]}_{i_L} G^{[L-1]}(x) W^{[L-1]}(x) \ldots G^{[1]}(x) W^{[1]}_i \right)_{i_L=1,\ldots,m^{[L]}} \]

where we used the induction hypothesis for \( \text{diag} \).

\[ \square \]

**Corollary 1.4. Gradient of Neural Network**

We have

\[ \nabla f(x) = W^{[1]^T} G^{[1]}(x) \cdots W^{[L]^T} G^{[L]}(x) \mathbb{1} \]

\[ = \sum_{i_1=1}^{m^{[1]}} \left( G^{[1]}(x) W^{[1]^T} \ldots G^{[L]}(x) \mathbb{1} \right)_{i_1} a^{[1]}_{i_1} \]

hence \( \nabla f(x) \in \text{span}_{\mathbb{R}^{n_0}} \left\{ a^{[1]}_{i_1}, \ldots, a^{[1]}_{m^{[1]}} \right\} \)

**Proof.** We have from Proposition 1.3

\[ \frac{\partial}{\partial x_i} f(x) = \frac{\partial}{\partial x_i} \left( \sum_{i_L=1}^{m^{[L]}} y^{[L]}_{i_L}(x) \right) \]

\[ = \frac{\partial}{\partial x_i} \left( \mathbb{1}^T y^{[L]}(x) \right) \]

\[ = \mathbb{1}^T G^{[L]}(x) W^{[L]} G^{[L-1]}(x) \ldots G^{[1]}(x) W^{[1]}_i \]
and hence
\[
\nabla f(x) = (I^T G^{[L]}(x) W^{[L]} G^{[L-1]}(x) \ldots G^{[1]}(x) W^{[1]})^T \\
= \sum_{i_1=1}^{m^{[L]}} (G^{[1]}(x) W^{[2]^T} G^{[2]}(x) \ldots W^{[L]^T} G^{[L]}(x) I)_{i_1} a_{i_1}^{[1]}
\]
\]

**Hessian**

Much of the following development relies on the form of the second derivative of the neural network function, i.e. the Hessian. In order to formulate the central result on the structure of the Hessian we first have to introduce and prove the following proposition.

**Proposition 1.5. Sum of second derivates**

Let \( b \in \mathbb{R}^{m^{[L]}} \) be arbitrary, then we have
\[
H \left( \langle b, y^{[L]}(\cdot) \rangle \right)(x) = \sum_{l=1}^{L} V^{[L]^T}(x) S^{[L]}_b(x) V^{[L]}(x)
\]
where
\[
V^{[l]}(x) = W^{[l]} G^{[l-1]}(x) W^{[l-1]} \ldots G^{[1]}(x) W^{[1]}, \quad l = 2, \ldots, L \\
V^{[1]}(x) = W^{[1]} \\
G^{[l]}(x) = \text{diag} \left( \left( g^{[l]}_{i_1} \right)' \left[ z^{[l]}_{i_1}(x) \right], i_1 = 1, \ldots, m^{[l]} \right), \quad l = 1, \ldots, L - 1
\]
and
\[
S^{[l]}_b(x) = \text{diag} \left( s^{[l]}_{i_1,b}(x), i_1 = 1, \ldots, m^{[l]} \right), \quad l = 1, \ldots, L \\
s^{[l]}_{i_1,b}(x) = \left( g^{[l]}_{i_1} \right)' \left[ z^{[l]}_{i_1}(x) \right] \sum_{i_{l+1}=1}^{m^{[l+1]}} \ldots \sum_{i_L=1}^{m^{[L]}} \left( g^{[l+1]}_{i_{l+1}} \right)' \left[ z^{[l+1]}_{i_{l+1}}(x) \right] \ldots \left( g^{[L]}_{i_L} \right)' \left[ z^{[L]}_{i_L}(x) \right] \\
\left( a^{[l+1]}_{i_1} \right)_{i_{l+1}} \left( a^{[l+2]}_{i_{l+2}} \right)_{i_{l+2}} \ldots \left( a^{[L]}_{i_L} \right)_{i_L} b_{i_L}, \quad l = 1, \ldots, L - 1 \\
s^{[L]}_{i_L,b}(x) = \left( g^{[L]}_{i_L} \right)' \left[ z^{[L]}_{i_L}(x) \right] b_{i_L}
\]

**Proof.** We use again induction over \( L \).
\[ L = 1 \]
\[ H[\langle b, y^{[1]}(\cdot) \rangle](x) = \left( \sum_{i_1=1}^{m^{[1]}} b_{i_1} \frac{\partial^2}{\partial x_i \partial x_j} y_i^{[1]}(x) \right)_{i,j=1, \ldots, m^{[0]}} \]
\[ = \left( \sum_{i_1=1}^{m^{[1]}} b_{i_1} \left( g_i^{[1]} \right)^{\prime\prime} \left[ z_i^{[1]}(x) \right] \left( a_i^{[1]} \right)_{i,j} \right)_{i,j=1, \ldots, m^{[0]}} \]
\[ = W^{[1]^T} \text{diag} \left( b_{i_1} \left( g_i^{[1]} \right)^{\prime\prime} \left[ z_i^{[1]}(x) \right] , \ i_1 = 1, \ldots, m^{[1]} \right) W^{[1]} \]
\[ = V^{[1]^T} y_{i_1}^{[1]}(x) V^{[1]}(x) \]

\[ L - 1 \rightarrow L \]
For each component \((i, j) \in \{1, \ldots, m^{[0]}\}^2\) of the Hessian we have by a simple calculation
\[ H[\langle b, y^{[L]}(\cdot) \rangle]_{i,j}(x) = \sum_{i_L=1}^{m^{[L]}} b_{i_L} \frac{\partial^2}{\partial x_i \partial x_j} y_{i_L}^{[L]}(x) \]
\[ = \sum_{i_L=1}^{m^{[L]}} b_{i_L} \left( g_i^{[L]} \right)^{\prime\prime} \left[ z_i^{[L]}(x) \right] \left( a_i^{[L]} \right) \left( \frac{\partial}{\partial x_i} y^{[L-1]}(x) \right) \left( \frac{\partial}{\partial x_j} y^{[L-1]}(x) \right) \]
\[ + \sum_{i_L=1}^{m^{[L]}} b_{i_L} \left( g_i^{[L]} \right)^{\prime} \left[ z_i^{[L]}(x) \right] \left( a_i^{[L]} \right) \left( \frac{\partial^2}{\partial x_i \partial x_j} y^{[L-1]}(x) \right) \]
\[ = I + II \]

In turn,
\[ I = \sum_{i_L=1}^{m^{[L]}} b_{i_L} \left( g_i^{[L]} \right)^{\prime\prime} \left[ z_i^{[L]}(x) \right] \left( a_i^{[L]} \right) \left( \frac{\partial}{\partial x_i} y^{[L-1]}(x) \right) \left( \frac{\partial}{\partial x_j} y^{[L-1]}(x) \right) \]

\[ \overset{\circ}{=\circ} \sum_{i_L=1}^{m^{[L]}} b_{i_L} \left( g_i^{[L]} \right)^{\prime\prime} \left[ z_i^{[L]}(x) \right] \left( a_i^{[L]} \right) \left( G^{[L-1]}(x) W^{[L-1]} \ldots G^{[1]}(x) W_i^{[1]} \right) \left( \frac{\partial}{\partial x_i} y^{[L-1]}(x) \right) \left( \frac{\partial}{\partial x_j} y^{[L-1]}(x) \right) \]

\[ \overset{\circ}{=\circ} \sum_{i_L=1}^{m^{[L]}} b_{i_L} \left( g_i^{[L]} \right)^{\prime\prime} \left[ z_i^{[L]}(x) \right] \left( W_{i_L}^{[L]} G^{[L-1]}(x) W^{[L-1]} \ldots G^{[1]}(x) W_i^{[1]} \right) \left( \frac{\partial}{\partial x_i} y^{[L-1]}(x) \right) \left( \frac{\partial}{\partial x_j} y^{[L-1]}(x) \right) \]

\[ \overset{\circ}{=\circ} \sum_{i_L=1}^{m^{[L]}} V_i^{[L]^T} (x)_{i_L} b_{i_L} \left( g_i^{[L]} \right)^{\prime\prime} \left[ z_i^{[L]}(x) \right] V_i^{[L]}(x)_{i_L} \]

\[ = \left( V^{[L]^T} y_{i_1}^{[1]}(x) V^{[L]}(x) \right)_{i,j} \]
where Proposition 1.3 was used for $\odot$, the definition of the weight matrices for $\odot$, the definition of $\mathbf{V}_r \mathbf{L}_{s_1}$ for $\odot$ and finally the definition for $\mathbf{S}_{r_1 \mathbf{L}_{b_1}}$. Similarly,

$$
II = \sum_{i_L=1}^{m^{[L]}} b_{i_L} \left( g^{[L]}_{i_L} \right)' \left[ z^{[L]}_{i_L} (x) \right] \frac{\partial^2}{\partial x_i \partial x_j} \langle a^{[L]}_{i_L}, y^{[L-1]}(x) \rangle
$$

$$
\odot \sum_{i_L=1}^{m^{[L]}} b_{i_L} \left( g^{[L]}_{i_L} \right)' \left[ z^{[L]}_{i_L} (x) \right] \left( \sum_{l=1}^{L-1} V^{[l]} T(x) S^{[l]}_{a_{i_L}}(x) V^{[l]}(x) \right)_{i,j}
$$

$$
= \left( \sum_{l=1}^{L-2} V^{[l]} T(x) \left( \sum_{i_L=1}^{m^{[L]}} b_{i_L} \left( g^{[L]}_{i_L} \right)' \left[ z^{[L]}_{i_L} (x) \right] S^{[L-1]}_{a_{i_L}}(x) \right) V^{[l]}(x) \right)_{i,j}
+ \left( V^{[L-1]} T(x) \left( \sum_{i_L=1}^{m^{[L]}} b_{i_L} \left( g^{[L]}_{i_L} \right)' \left[ z^{[L]}_{i_L} (x) \right] S^{[L-1]}_{a_{i_L}}(x) \right) V^{[L-1]}(x) \right)_{i,j}
$$

$$
= \left( \sum_{l=1}^{L-2} V^{[l]} T(x) \text{diag} \left( \sum_{i_L=1}^{m^{[L]}} b_{i_L} \left( g^{[L]}_{i_L} \right)' \left[ z^{[L]}_{i_L} (x) \right] S^{[L-1]}_{i_L a_{i_L}}(x), i_L = 1, \ldots, m^{[L]} \right) V^{[l]}(x) \right)_{i,j}
+ \left( V^{[L-1]} T(x) \text{diag} \left( \sum_{i_L=1}^{m^{[L]}} b_{i_L} \left( g^{[L]}_{i_L} \right)' \left[ z^{[L]}_{i_L} (x) \right] S^{[L-1]}_{i_L a_{i_L}}(x), i_L = 1, \ldots, m^{[L-1]} \right) V^{[L-1]}(x) \right)_{i,j}
$$

$$
\odot \sum_{l=1}^{L-1} V^{[l]} T(x) S^{[l]}_{b_L}(x) V^{[l]}(x)
$$
where we used the induction hypothesis (with $b = a_{i_L}^{[L]}$) for ① and for ② that

\[(*) = \sum_{i_L=1}^{m^{[L]}} b_{i_L} \left( g_{i_L}^{[L]} \right)' \left[ z_{i_L}^{[L]}(x) \right] s_{i_l,a_{i_L}^{[L]}}^{[L]}(x) \]

\[= \sum_{i_L=1}^{m^{[L]}} b_{i_L} \left( g_{i_L}^{[L]} \right)' \left[ z_{i_L}^{[L]}(x) \right] \left( g_{i_L}^{[L]} \right)^\prime \left[ z_{i_L}^{[L]}(x) \right] \sum_{i_{L-1}=1}^{m^{[L-1]}} \sum_{i_{L-2}=1}^{m^{[L-1]}} \ldots \sum_{i_1=1}^{m^{[L]}} \left( g_{i_{L-1}+1}^{[L-1]} \right)' \left[ z_{i_{L-1}+1}^{[L-1]}(x) \right] \ldots \left( g_{i_{L-1}}^{[L-1]} \right)' \left[ z_{i_{L-1}}^{[L-1]}(x) \right] \left( a_{i_{L-1}+1}^{[L-1]} \right) \ldots \left( a_{i_{L-1}}^{[L-1]} \right) b_{i_L} \]

\[(**) = \sum_{i_L=1}^{m^{[L]}} b_{i_L} \left( g_{i_L}^{[L]} \right)' \left[ z_{i_L}^{[L]}(x) \right] \left( g_{i_L}^{[L-1]} \right)^\prime \left[ z_{i_L}^{[L-1]}(x) \right] \left( a_{i_L}^{[L]} \right) s_{i_{L-1},a_{i_L}^{[L-1]}}^{[L-1]}(x) \]

\[= \sum_{i_L=1}^{m^{[L]}} b_{i_L} \left( g_{i_L}^{[L]} \right)' \left[ z_{i_L}^{[L]}(x) \right] \left( g_{i_L}^{[L-1]} \right)^\prime \left[ z_{i_L}^{[L-1]}(x) \right] \left( a_{i_L}^{[L]} \right) \]

Putting I and II together we arrive at

\[H[\langle b, y^{[L]}(\cdot) \rangle](x) = \sum_{l=1}^{L} V^{[L]}(x) S_b^{[L]}(x) V^{[L]}(x) \]

\[\square\]

**Corollary 1.6. Hessian of Neural Network**

We have

\[H[f](x) = \sum_{l=1}^{L} V^{[L]}(x) S^{[L]}(x) V^{[L]}(x) \]

where

\[V^{[l]}(x) = W^{[l]} G^{[l-1]}(x) W^{[l-1]} \ldots G^{[1]}(x) W^{[1]}, \quad l = 2, \ldots, L \]

\[V^{[1]}(x) = W^{[1]} \]

\[G^{[l]}(x) = diag \left( \left( g_i^{[l]} \right)' \left[ z_i^{[l]}(x) \right], i_l = 1, \ldots, m^{[l]} \right), \quad l = 1, \ldots, L \]
and

\[ S^{[l]}(x) = \text{diag}\left( s_{i_1}^{[l]}(x), i_1 = 1, \ldots, m^{[l]} \right) \]

\[ s_{i_1}^{[l]}(x) = \left( g_{i_1}^{[l]} \right)^{\prime} \left[ z_{i_1}^{[l]}(x) \right] \sum_{i_{l+1}=1}^{m^{[l+1]}} \sum_{i_L=1}^{m^{[L]}} \left( g_{i_{l+1}}^{[l+1]} \right)^{\prime} \left[ z_{i_{l+1}}^{[l+1]}(x) \right] \cdots \left( g_{i_L}^{[L]} \right)^{\prime} \left[ z_{i_L}^{[L]}(x) \right] \]

\[ s_{i_L}^{[L]}(x) = \left( g_{i_L}^{[L]} \right)^{\prime} \left[ z_{i_L}^{[L]}(x) \right] \]

**Proof.** Simply use Proposition 1.5 with \( b = 1 \).

For convenience we define

\[ v_{i_1}^{[l]}(x) = V^{[l]}(x)^{\top}_{i_1}, \quad i_1 = 1, \ldots, m^{[l]}, l = 1, \ldots, L \]

for \( V^{[l]}(x) \) from Corollary 1.6.

**Example 1.7. Hessians for small \( L \)**

We will demonstrate the preceding proposition by writing out the Hessians explicitly for small \( L \).

1. **\( L = 1 \)**  We have

\[ H[f](x) \overset{\text{def}}{=} V^{[1]}(x)^{\top}S^{[1]}(x)V^{[1]}(x) \]

\[ \overset{\text{def}}{=} W^{[1]}^{\top}\text{diag}\left( (g_{i_1}^{[1]})^{\prime} \left[ z_{i_1}^{[1]}(x) \right], i_1 = 1, \ldots, m^{[1]} \right) W^{[1]} \]

\[ \overset{\text{def}}{=} \sum_{i_1=1}^{m^{[1]}} (g_{i_1}^{[1]})^{\prime} \left[ z_{i_1}^{[1]}(x) \right] a_{i_1}^{[1]} \otimes a_{i_1}^{[1]} \]

where we used for (1) Corollary 1.6 for \( L = 1 \), for (2) the definitions of Corollary 1.6 and for (3) simply our notation and the definition of matrix-matrix and matrix-vector-multiplication.

Note that this is the same form as [34, Equation (26)].

2. **\( L = 2 \)**  Similar to the case of \( L = 1 \) we have

\[ H[f](x) \overset{\text{def}}{=} V^{[2]}(x)^{\top}S^{[2]}(x)V^{[2]}(x) + V^{[1]}(x)^{\top}S^{[1]}(x)V^{[1]}(x) \]

\[ \overset{\text{def}}{=} \sum_{i_{12}=1}^{m^{[2]}} \left[ a_{i_{12}}^{[2]}(x) \otimes a_{i_{12}}^{[2]}(x) \right] + \sum_{i_1=1}^{m^{[1]}} \left[ a_{i_1}^{[1]}(x) \otimes a_{i_1}^{[1]}(x) \right] \]

\[ \overset{\text{def}}{=} \sum_{i_{12}=1}^{m^{[2]}} (g_{i_{12}}^{[2]})^{\prime} \left[ z_{i_{12}}^{[2]}(x) \right] \left[ \left( \sum_{i_{11}=1}^{m^{[1]}} (g_{i_{11}}^{[1]})^{\prime} \left[ z_{i_{11}}^{[1]}(x) \right] a_{i_{11}}^{[1]}(x) \right) \otimes \left( \sum_{i_{12}}^{m^{[2]}} \left( g_{i_{12}}^{[2]}(x) \right)^{\prime} \left[ z_{i_{12}}^{[2]}(x) \right] a_{i_{12}}^{[2]}(x) \right) \right] \]

\[ + \sum_{i_1=1}^{m^{[1]}} (g_{i_1}^{[1]})^{\prime} \left[ z_{i_1}^{[1]}(x) \right] \sum_{i_{12}=1}^{m^{[2]}} (g_{i_{12}}^{[2]})^{\prime} \left[ z_{i_{12}}^{[2]}(x) \right] \left[ a_{i_1}^{[1]}(x) \otimes a_{i_1}^{[1]}(x) \right] \]
where we used Corollary 1.6 for \( \mathbf{1} \), and for \( \mathbf{2} \) and \( \mathbf{3} \) the notations introduced previously and the definition of matrix-matrix and matrix-vector-multiplication.

The various quantities we defined in Corollary 1.6 take rather simple forms here,

\[
v^{[2]}_{i_2}(x) = V^{[2]T}(x)_{i_2}
= W^{[1]T}G^{[1]}(x)W^{[2]T}_{i_2}
= \sum_{i_1=1}^{m^{[1]}} \left(g^{[1]}_{i_1}\right)' \begin{bmatrix} z^{[1]}_{i_1}(x) \end{bmatrix} \left(a^{[2]}_{i_2}\right)_{i_1}
\]

\[
s^{[2]}_{i_2}(x) = \left(g^{[2]}_{i_2}\right)' \begin{bmatrix} z^{[2]}_{i_2}(x) \end{bmatrix}
\]

\[
s^{[1]}_{i_1}(x) = \left(g^{[1]}_{i_1}\right)' \begin{bmatrix} z^{[1]}_{i_1}(x) \end{bmatrix} \sum_{i_2=1}^{m^{[2]}} \left(g^{[2]}_{i_2}\right)' \begin{bmatrix} z^{[2]}_{i_2}(x) \end{bmatrix} \left(a^{[2]}_{i_2}\right)_{i_1}
\]

Note that the first line is the same as [82, Equation (2.0.18)] if we set \( V_x \leftarrow V^{[1]}(x) \), \( G_x \leftarrow G^{[1]}(x) \), \( S_x \leftarrow S^{[2]}(x) \), \( T_x \leftarrow V^{[2]}(x) \). Furthermore, the third line is the same as [82, Equation (2.0.17)] if we additionally set \( s_{x,\ell} \leftarrow s^{[1]}_{x}(x) \), \( t_{x,ii} \leftarrow s^{[1]}_{i}(x) \) and \( v_{x,\ell} \leftarrow v^{[2]}_{x}(x) \). Finally, the last line is the same as the one after [82, Equation (2.0.13)].

3. \( L = 3 \) We now proceed in the reverse direction, i.e. we start with a direct calculation of the Hessian and recover the form from above at the end.

\[
H[f](x)_{ij} = \sum_{i_3=1}^{m^{[3]}} \frac{\partial^2}{\partial x_i \partial x_j} y^{[3]}_{i_3}(x)
= \sum_{i_3=1}^{m^{[3]}} \left(g^{[3]}_{i_3}\right)' \begin{bmatrix} z^{[3]}_{i_3}(x) \end{bmatrix} \left(\sum_{i_2=1}^{m^{[2]}} \left(a^{[3]}_{i_3}\right)_{i_2} \frac{\partial}{\partial x_i} \left(a^{[2]}_{i_2}\right)_{i_2}\right) \left(\sum_{i_2=1}^{m^{[2]}} \left(a^{[3]}_{i_3}\right)_{i_2} \frac{\partial}{\partial x_i} \left(a^{[2]}_{i_2}\right)_{i_2}\right)
+ \left(g^{[3]}_{i_3}\right)' \begin{bmatrix} z^{[3]}_{i_3}(x) \end{bmatrix} \left(\sum_{i_2=1}^{m^{[2]}} \left(a^{[3]}_{i_3}\right)_{i_2} \frac{\partial^2}{\partial x_i \partial x_j} \left(a^{[2]}_{i_2}\right)_{i_2}\right)
= \sum_{i_3=1}^{m^{[3]}} \left(g^{[3]}_{i_3}\right)' \begin{bmatrix} z^{[3]}_{i_3}(x) \end{bmatrix} \left(\sum_{i_2=1}^{m^{[2]}} \left(a^{[3]}_{i_3}\right)_{i_2} \left(g^{[2]}_{i_2}\right)' \begin{bmatrix} z^{[2]}_{i_2}(x) \end{bmatrix} \left(\sum_{i_1=1}^{m^{[1]}} \left(a^{[2]}_{i_2}\right)_{i_1} \left(g^{[1]}_{i_1}\right)' \begin{bmatrix} z^{[1]}_{i_1}(x) \end{bmatrix} \left(a^{[1]}_{i_1}\right)_{i_1}\right)\right)
\cdot \left(\sum_{i_2=1}^{m^{[2]}} \left(a^{[3]}_{i_3}\right)_{i_2} \left(g^{[2]}_{i_2}\right)' \begin{bmatrix} z^{[2]}_{i_2}(x) \end{bmatrix} \left(\sum_{i_1=1}^{m^{[1]}} \left(a^{[2]}_{i_2}\right)_{i_1} \left(g^{[1]}_{i_1}\right)' \begin{bmatrix} z^{[1]}_{i_1}(x) \end{bmatrix} \left(a^{[1]}_{i_1}\right)_{i_1}\right)\right)
+ \left(g^{[3]}_{i_3}\right)' \begin{bmatrix} z^{[3]}_{i_3}(x) \end{bmatrix} \left(\sum_{i_2=1}^{m^{[2]}} \left(a^{[3]}_{i_3}\right)_{i_2} \left(g^{[2]}_{i_2}\right)' \begin{bmatrix} z^{[2]}_{i_2}(x) \end{bmatrix} \left(\sum_{i_1=1}^{m^{[1]}} \left(a^{[2]}_{i_2}\right)_{i_1} \left(g^{[1]}_{i_1}\right)' \begin{bmatrix} z^{[1]}_{i_1}(x) \end{bmatrix} \left(a^{[1]}_{i_1}\right)_{i_1}\right)\right)
\cdot \left(\sum_{i_2=1}^{m^{[2]}} \left(a^{[3]}_{i_3}\right)_{i_2} \left(g^{[2]}_{i_2}\right)' \begin{bmatrix} z^{[2]}_{i_2}(x) \end{bmatrix} \left(\sum_{i_1=1}^{m^{[1]}} \left(a^{[2]}_{i_2}\right)_{i_1} \left(g^{[1]}_{i_1}\right)' \begin{bmatrix} z^{[1]}_{i_1}(x) \end{bmatrix} \left(a^{[1]}_{i_1}\right)_{i_1}\right)\right)
+ \left(g^{[3]}_{i_3}\right)' \begin{bmatrix} z^{[3]}_{i_3}(x) \end{bmatrix} \left(\sum_{i_2=1}^{m^{[2]}} \left(a^{[3]}_{i_3}\right)_{i_2} \left(g^{[2]}_{i_2}\right)' \begin{bmatrix} z^{[2]}_{i_2}(x) \end{bmatrix} \left(\sum_{i_1=1}^{m^{[1]}} \left(a^{[2]}_{i_2}\right)_{i_1} \left(g^{[1]}_{i_1}\right)' \begin{bmatrix} z^{[1]}_{i_1}(x) \end{bmatrix} \left(a^{[1]}_{i_1}\right)_{i_1}\right)\right).
Rearranging and using the definitions from Corollary 1.6 leads to

\[
H(f)(x)_{ij} = \sum_{i_3=1}^{m[3]} (a_{i_3}^{[3]})^\top [z_{i_3}^{[3]}(x)] \left( \sum_{i_2}^{m[2]} (a_{i_2}^{[2]})_{i_2} \right) \left( \sum_{i_1}^{m[1]} a_{i_1}^{[1]} \right) [z_1^{[1]}(x)] \left( a_1^{[1]} \right)
\]

\[
+ \sum_{i_3=1}^{m[3]} (a_{i_3}^{[3]})^\top [z_{i_3}^{[3]}(x)] \left( \sum_{i_2}^{m[2]} (a_{i_2}^{[2]})_{i_2} \right) \left( \sum_{i_1}^{m[1]} a_{i_1}^{[1]} \right) [z_1^{[1]}(x)] \left( a_1^{[1]} \right)
\]

\[
+ \sum_{i_3=1}^{m[3]} (g_{i_3}^{[3]})^\top [z_{i_3}^{[3]}(x)] \left( \sum_{i_2}^{m[2]} (a_{i_2}^{[2]})_{i_2} \right) \left( \sum_{i_1}^{m[1]} a_{i_1}^{[1]} \right) [z_1^{[1]}(x)] \left( a_1^{[1]} \right)
\]

\[
+ \sum_{i_3=1}^{m[3]} (g_{i_3}^{[3]})^\top [z_{i_3}^{[3]}(x)] \left( \sum_{i_2}^{m[2]} (a_{i_2}^{[2]})_{i_2} \right) \left( \sum_{i_1}^{m[1]} a_{i_1}^{[1]} \right) [z_1^{[1]}(x)] \left( a_1^{[1]} \right)
\]

\[
+ \sum_{i_3=1}^{m[3]} (g_{i_3}^{[3]})^\top [z_{i_3}^{[3]}(x)] \left( \sum_{i_2}^{m[2]} (a_{i_2}^{[2]})_{i_2} \right) \left( \sum_{i_1}^{m[1]} a_{i_1}^{[1]} \right) [z_1^{[1]}(x)] \left( a_1^{[1]} \right)
\]

\[
+ \sum_{i_3=1}^{m[3]} (g_{i_3}^{[3]})^\top [z_{i_3}^{[3]}(x)] \left( \sum_{i_2}^{m[2]} (a_{i_2}^{[2]})_{i_2} \right) \left( \sum_{i_1}^{m[1]} a_{i_1}^{[1]} \right) [z_1^{[1]}(x)] \left( a_1^{[1]} \right)
\]

\[
+ \sum_{i_3=1}^{m[3]} (g_{i_3}^{[3]})^\top [z_{i_3}^{[3]}(x)] \left( \sum_{i_2}^{m[2]} (a_{i_2}^{[2]})_{i_2} \right) \left( \sum_{i_1}^{m[1]} a_{i_1}^{[1]} \right) [z_1^{[1]}(x)] \left( a_1^{[1]} \right)
\]

\[
+ \sum_{i_3=1}^{m[3]} (g_{i_3}^{[3]})^\top [z_{i_3}^{[3]}(x)] \left( \sum_{i_2}^{m[2]} (a_{i_2}^{[2]})_{i_2} \right) \left( \sum_{i_1}^{m[1]} a_{i_1}^{[1]} \right) [z_1^{[1]}(x)] \left( a_1^{[1]} \right)
\]

\[
= \sum_{i_3=1}^{m[3]} \left( W^{[3]}c^{[2]}(x)W^{[2]}c^{[1]}(x)W^{[1]} \right) \left( W^{[3]}c^{[2]}(x)W^{[2]}c^{[1]}(x)W^{[1]} \right)_{i_3j}
\]

\[
+ \sum_{i_2=1}^{m[2]} \left( W^{[2]}c^{[1]}(x)W^{[1]} \right)_{i_2j}
\]

\[
+ \sum_{i_1=1}^{m[1]} \left( a_1^{[1]} \right)_{i_1j}
\]

\[
= \sum_{i_3=1}^{m[3]} \left( a_3^{[3]}(x) \otimes a_3^{[3]}(x) \right)_{i_3j} + \sum_{i_2=1}^{m[2]} \left( a_2^{[2]}(x) \otimes a_2^{[2]}(x) \right)_{i_2j} + \sum_{i_1=1}^{m[1]} \left( a_1^{[1]}(x) \otimes a_1^{[1]}(x) \right)_{i_1j}
\]

Rewriting this expression with the definition of \( V^{[l]}(x) \) and \( S^{[l]}(x) \) we get

\[
H(f)(x) = V^{[3]T}(x)diag \left( s_3^{[3]}(x), i_3 = 1, \ldots, m[3] \right) V^{[3]}(x)
\]

\[
+ V^{[2]T}(x)diag \left( s_2^{[2]}(x), i_2 = 1, \ldots, m[2] \right) V^{[2]}(x)
\]

\[
+ W^{[1]T}(x)diag \left( s_1^{[1]}(x), i_1 = 1, \ldots, m[1] \right) W^{[1]}
\]

\[
= \sum_{l=1}^{3} V^{[l]T}(x)S^{[l]}(x)V^{[l]}(x)
\]
2. Dimension Reduction in the input layer

In this chapter we are concerned with the case that $m^0 > m^1$, i.e. the number of neurons in the first hidden layer is smaller than the number of inputs. The main result will be that it is possible to efficiently construct a new neural network with $m^0 = m^1$, i.e. a dimensionality reduction is performed. Of particular interest is the case $m^0 >> m^1$, i.e. there are much less neurons in the first hidden layer than inputs, and the input dimension is rather large. High (input) dimension poses in general a severe problem in machine learning and statistics. On the one hand high dimensional data can have a very counter-intuitive geometry (see for example [97, Section 1.3] for an overview) which poses in itself a problem for some methods like nearest neighbour [17], on the other hand it might also pose computational problems and require large sample sizes. In general problems associated with high dimensional data are collected under the term curse-of-dimensionality, originally coined by Bellman in optimal control. It is hence desirable to reduce the dimensionality of a given problem and the method developed here is indeed one efficient way to reduce the dimensionality if $m^0 >> m^1$.

2.1. Dimension Reduction with approximate span of weights in first layer

The developments in this section rest on a simple observation: From the definition of $f(x)$ we see that for arbitrary $L \in \mathbb{N}$ all $y_i^{[l]}$, $l = 1, \ldots, L$, do not depend directly on $x$ but instead on $\langle a_i^{[1]}, x \rangle$. This means that effectively $f(x)$ is not a function of the $m^0$ inputs $x_1, \ldots, x_{m^0}$ but only of the $m^1$ (transformed) inputs $\langle a_1^{[1]}, x \rangle, \ldots, \langle a_{m^1}^{[1]}, x \rangle$. This heuristic will now be made precise and used to construct and analyse an efficient method for dimensionality reduction.

The following simple result is exactly the same as [34, Theorem 1, Remark 1], which can be used verbatimely since it does not rely on $L = 1$. The applicability for $L = 2$ has already been noted by [82].

**Proposition 2.1.** Dimensionality reduction for approximate span

Let $A = \text{span}_{\mathbb{R}^{m^0}} \left\{ a_1^{[1]}, \ldots, a_{m^1}^{[1]} \right\}$ and let $\tilde{A} \subseteq \mathbb{R}^{m^0}$ be a $m^1$ dimensional subspace.

18
Select an arbitrary orthonormal basis \( \tilde{a}_1, \ldots, \tilde{a}_{m[1]} \in \mathbb{R}^{m[0]} \) of this subspace and define

\[
\tilde{W}[1] = \begin{pmatrix} \tilde{a}_1^T \\ \vdots \\ \tilde{a}_{m[1]}^T \end{pmatrix}
\]

as well as \( \alpha_{i_1} = \tilde{W}[1]^T a_{i_1} \) for \( i_1 = 1, \ldots, m[1] \) and finally define \( \tilde{f}(\tilde{x}) = f(\tilde{W}[1]^T \tilde{x}) \). We then have

1. \( \tilde{f} \) is again a neural network function, identical to \( f \) apart from new weights \( \alpha_1, \ldots, \alpha_m \) in the first layer. Furthermore, for any other function \( \hat{f}(\hat{x}) \) one has

\[
\| f - \hat{f}(\tilde{W}[1]x) \|_\infty \leq \| f \|_{\text{Lip}} \| P_A - P_A \|_1 + \| \hat{f} - f \|_\infty
\]

2. For arbitrary \( \hat{a}_1, \ldots, \hat{a}_{m[1]} \in \mathbb{R}^{m[0]} \) we have

\[
\| a_{i_1} - \tilde{W}[1]^T \hat{a}_{i_1} \|_2 \leq \| P_A - P_A \|_F + \| \alpha_{i_1} - \hat{\alpha}_{i_1} \|_2, \quad i_1 = 1, \ldots, m[1]
\]

3. Assume \( S(a_1, \ldots, a_{m[1]}) \leq \epsilon \), then

\[
S(\alpha_1, \ldots, \alpha_{m[1]}) \leq \epsilon + \| P_A - P_A \|_F
\]

**Proof.** 1. The first part of the statement is clear from the definitions. For the second part, consider an arbitrary \( x \) and note that

\[
| f(x) - \hat{f}(\tilde{W}[1]x) | \leq | f(x) - \hat{f}(\tilde{W}[1]x) | + | \hat{f}(\tilde{W}[1]x) - \tilde{f}(\tilde{W}[1]x) |
\]

\[
\leq | f(x) - \hat{f}(\tilde{W}[1]x) | + \| \tilde{f} - \hat{f} \|_\infty
\]

\[
= | f(x) - f(\tilde{W}[1]x) | + \| \tilde{f} - \hat{f} \|_\infty
\]

\[
\leq \| f \|_{\text{Lip}} \| P_A x - P_A x \|_2 + \| \tilde{f} - \hat{f} \|_\infty
\]

Taking the supremum over \( x \) leads immediately to

\[
\| f - \hat{f}(\tilde{W}[1]x) \|_\infty \leq \| f \|_{\text{Lip}} \| P_A - P_A \|_1 + \| \tilde{f} - \hat{f} \|_\infty,
\]

the required statement.

2. For any \( i_1 = 1, \ldots, m[1] \) we simply have

\[
\| a_{i_1} - \tilde{W}[1]^T \hat{a}_{i_1} \|_2 \leq \| a_{i_1} - \tilde{W}[1]^T \alpha_{i_1} \|_2 + \| \tilde{W}[1]^T (\alpha_{i_1} - \hat{\alpha}_{i_1}) \|_2
\]

\[
= \| (P_A - P_A) a_{i_1} \|_2 + \| \tilde{W}[1]^T (\alpha_{i_1} - \hat{\alpha}_{i_1}) \|_2
\]

\[
\leq \| P_A - P_A \|_F + \| \alpha_{i_1} - \hat{\alpha}_{i_1} \|_2
\]
Assume now $S(a_1^{[1]}, \ldots, a_{m^{[1]}}^{[1]}) \leq \epsilon$. W.l.o.g we can assume that $\tilde{a}_1^{[1]}, \ldots, \tilde{a}_{m^{[1]}}^{[1]} \in A$, then we get

$$S(\alpha_1, \ldots, \alpha_{m^{[1]}}) = S(\tilde{W}^T a_1^{[1]}, \ldots, \tilde{W}^T a_{m^{[1]}}^{[1]})$$

$$= S(\tilde{W}^T \tilde{W}^T a_1^{[1]}, \ldots, \tilde{W}^T \tilde{W}^T a_{m^{[1]}}^{[1]}) = S(P_A a_1^{[1]}, \ldots, P_A a_{m^{[1]}}^{[1]})$$

$$\leq \sqrt{\sum_{i=1}^{m^{[1]}} \| P_A \tilde{a}_i^{[1]} - \tilde{a}_i^{[1]} \|^2_2}$$

$$\leq \sqrt{\sum_{i=1}^{m^{[1]}} \| P_A a_i^{[1]} - P_A \tilde{a}_i^{[1]} \|^2_2} + \sqrt{\sum_{i=1}^{m^{[1]}} \| P_A \tilde{a}_i^{[1]} - P_A \tilde{a}_i^{[1]} \|^2_2}$$

$$\leq \epsilon + \| P_A - P_A \|_F$$

\[\square\]

**Remark 2.2. Interpretation of Proposition 2.1**

We give a short interpretation of Proposition 2.1:

1. Part 1 of the preceding Proposition says that if we know approximately the span of the weights in the first layer, i.e. if we can find a space $\tilde{A}$ that is close to $A$ (in the sense that $\| P_A - P_A \|_F$ is small) and we can approximate $\tilde{f}$ uniformly (say by $\hat{f}$), then we already have a uniform approximation of $f$ via $\hat{f}(\tilde{W}^T \tilde{x})$. Note that if we can sample easily from $f$ we can do so also from $\tilde{f}$ due to $\tilde{f}(\tilde{x}) = f(\tilde{W}^T \tilde{x})$.

2. Part 2 means that if we can learn approximately the weights in the first layer, i.e. $\alpha_i$, for the reduced neural network representing $\tilde{f}$, say by $\hat{\alpha}_i$, then we can already approximate the weights in the first layer of $f$ via $\tilde{W}^T \hat{\alpha}_{i1}$.

3. Part 3 means that if the weights in the first layer are $\epsilon$-quasiorthogonal, also the weights in the reduced network be (almost) $\epsilon$-quasiorthogonal.

The Proposition hence implies that if we know approximately $A$ we can restrict ourselves to developing methods for the case $m^{[0]} = m^{[1]}$, unless of course we want to profit from very high dimensional input spaces.

**Remark 2.3.** The approach to dimensionality reduction in this section can be interpreted as a form of active subspace method [26]. In the context of neural networks a similar approach has been used by [109] for $L = 1$, based in turn on [108].

### 2.2. Approximating the span of first layer weights

In the preceding section we found that the dimensionality can be effectively reduced if an approximation of the space $A$ is available. Hence the task in this section is to approximate $A$, the span of the weights in the first layer. We will use the methodology
from [35, Section 2.2.1], which is based in turn on an observation in [33]: For all \( x \in D \) we have \( \nabla f(x) \in A \) (cf Corollary 1.4), i.e. using differentiation the weights in the first layer become accessible. Generically, we should have \( A = \text{span}_{\mathbb{R}^{m[0]}} \{ \nabla f(x_k), k = 1, \ldots, N \} \) if \( N \) is large enough and the query points \( x_1, \ldots, x_N \) are “well-spread”. Of course in general only \( f \) is available and not \( \nabla f \), but the latter can be easily approximated by finite differences of first order. Unfortunately, the approximated gradients won’t be in \( A \) in general, but intuitively they should be close to \( A \) and hence might be useful approximate \( A \). In the following these intuitive ideas will be made precise.

Note that we focus here on active sampling and a setting without noise: We assume that the neural network can be evaluated at arbitrary (admissible) inputs, i.e. it can be actively queried, and that the outputs returned are without noise. Note that in common scenarios in machine learning

Introducing the approximation algorithm

First, we have to set up the finite difference approximation. Consider points \( x_1, \ldots, x_N \in \mathbb{R}^d \) and a step size \( \epsilon > 0 \). Using the Mean Value Theorem we have for \( j = 1, \ldots, m^{[0]} \) and \( k = 1, \ldots, N \)

\[
\frac{\partial}{\partial x_j} f(x_k) = \frac{f(x_k + \epsilon e_j) - f(x_k)}{\epsilon} - \left( \frac{\partial}{\partial e_j} f(x_k + \eta_{j,k} e_j) - \frac{\partial}{\partial e_j} f(x_k) \right) \quad (2.1)
\]

with \( \eta_{j,k} \in [0, \epsilon] \). Define now

\[
X = \left( \frac{\partial}{\partial e_j} f(x_k) \right)_{j=1,\ldots,m^{[0]}}^{k=1,\ldots,N}
\]

\[
Y = \left( \frac{f(x_k + \epsilon e_j) - f(x_k)}{\epsilon} \right)_{j=1,\ldots,m^{[0]}}^{k=1,\ldots,N}
\]

\[
E = \left( \frac{\partial}{\partial e_j} f(x_k + \eta_{j,k} e_j) - \frac{\partial}{\partial e_j} f(x_k) \right)_{j=1,\ldots,m^{[0]}}^{k=1,\ldots,N}
\]

in order to rewrite (2.1) as

\[
Y = X + E,
\]

i.e. we interpret the finite differences (stored in \( Y \)) as derivates (stored in \( X \)) with perturbations (stored in \( E \)). If we had no perturbations, then the columns of \( Y \) should span \( A \) generically, if the \( x_k \) are “spread out enough”. A natural choice is hence to sample the \( x_k \) independently uniformly, say, from the unit sphere \( S^{m^{[0]}-1} \). Assuming that we reached all of \( A \) we could then simply apply an orthonormalization procedure (e.g. Gram-Schmidt or a numerically stable alternative) to get the \( \hat{a}_i^{[1]} \) from the previous discussion. However, since we have only perturbed derivates available due to the finite differences, we need to deal with the perturbation. A simple intuitive choice would be to find a \( m^{[1]} \)-dimensional subspace closest to the columns of \( Y \) in say the \( \ell_2 \)-sense and finding an orthonormal base of this space. Since this is equivalent to performing SVD on \( Y^T \) (see e.g. [19, Chapter 3]) we are lead immediately to our first algorithm.
Algorithm 1 Basic approximation algorithm for span of first layer weights

**Require:** Number of samples $N$, neural network function $f$, step size $\epsilon > 0$

1. Draw $x_1, \ldots, x_N$ i.i.d. uniformly from $\mathbb{S}^{m[0]-1}$
2. Build $Y$
3. Compute SVD of $Y^T$,
   
   $$Y^T = (\hat{U}_1 \quad \hat{U}_2) \begin{pmatrix} \hat{\Sigma}_1 & 0 \\ 0 & \hat{\Sigma}_2 \end{pmatrix} (\hat{V}_1^T \quad \hat{V}_2^T)$$

   where $\hat{\Sigma}_1 = \text{diag}(\sigma_1, \ldots, \sigma_{m[1]})$ contains the $m[1]$ largest singular values
4. return $\hat{V}_1$

If the algorithm returns vectors $\tilde{a}_1, \ldots, \tilde{a}_{m[1]}$, the approximation of $A$ will then be

$$\tilde{A} = \text{span} \{ \tilde{a}_1, \ldots, \tilde{a}_{m[1]} \}$$

As discussed previously, we will also set

$$\tilde{W}^{[1]} = \begin{pmatrix} \tilde{a}_1^T \\ \vdots \\ \tilde{a}_{m[1]}^T \end{pmatrix}$$

Analysis of the approximation algorithm

We now show that the space found by the preceding algorithm is in fact a good approximation to $A$. In order to measure how close $A$ and $\tilde{A}$ are the Frobenius norm of the difference of the projection operators will be used, $\|P_A - P_{\tilde{A}}\|_F$. If $A$ and $\tilde{A}$ have the same dimension this quantity describes intuitively how strongly the two spaces are rotated with respect to each other, see e.g. [100].

Intuitively, neglecting numerical errors, the smaller the step size $\epsilon > 0$, the better our approximation should be. However, since we are using a probabilistic algorithm only a probabilistic bound for $\|P_A - P_{\tilde{A}}\|_F$ can be expected. The next goal is hence to show that we have

$$\|P_A - P_{\tilde{A}}\|_F \leq B$$

with high probability for some term $B \geq 0$ that depends only on $\epsilon > 0$, $N \in \mathbb{N}$ and $f$ and should be decreasing with decreasing $\epsilon$ and increasing $N$.

We will use the strategy from [34, Section 2.2.1] which consists in applying the classical Wedin theorem [99] (cf. Section S.1.1) First, a bound on the Frobenius distance $\|X - Y\|_F$ is required which will be derived in the next lemma.

**Lemma 2.4. Upper bound on $\|X - Y\|_F$**

In the situation of Algorithm 1

$$\|X - Y\|_F \leq \sqrt{NC^{(2,4)}}\epsilon$$
with

\[ C^{[2.4]} = \sqrt{m[L]} \sqrt{\sum_{j=1}^{m[0]} (C_j^{[S.31]})^2}. \]

**Proof.** We have

\[
\|X - Y\|_F = \|E\|_F = \sqrt{\sum_{j=1}^{m[0]} \sum_{k=1}^{N} \left( \frac{\partial}{\partial x_j} f(x_k + \eta_j \epsilon_j) - \frac{\partial}{\partial x_j} f(x_k) \right)^2}
\]

\[
= \sqrt{\sum_{j=1}^{m[0]} \sum_{k=1}^{N} \sum_{i_L=1}^{m[L]} \left( \frac{\partial}{\partial x_j} y_{i_L}^{[L]}(x_k + \eta_j \epsilon_j) - \frac{\partial}{\partial x_j} y_{i_L}^{[L]}(x_k) \right)^2}
\]

\[
\leq \sqrt{m[L]} \sqrt{\sum_{j=1}^{m[0]} \sum_{k=1}^{N} \sum_{i_L=1}^{m[L]} \left( \frac{\partial}{\partial x_j} y_{i_L}^{[L]}(x_k + \eta_j \epsilon_j) - \frac{\partial}{\partial x_j} y_{i_L}^{[L]}(x_k) \right)^2}
\]

\[
= \sqrt{m[L]} \sqrt{\sum_{j=1}^{m[0]} \sum_{k=1}^{N} \sum_{i_L=1}^{m[L]} \left( \frac{\partial}{\partial x_j} y_{i_L}^{[L]}(x_k + \eta_j \epsilon_j) - \frac{\partial}{\partial x_j} y_{i_L}^{[L]}(x_k) \right)^2}
\]

\[
= \sqrt{m[L]} \sqrt{\sum_{j=1}^{m[0]} \sum_{k=1}^{N} (C_j^{[S.31]} (\epsilon)^2}
\]

\[
= \sqrt{m[L]} \sqrt{N \epsilon^2 \sum_{j=1}^{m[0]} (C_j^{[S.31]})^2}
\]

where we used the triangle inequality (for the absolute value) for \( \text{(1)} \), the fact that \( \| \cdot \|_1 \leq \sqrt{m[L]} \| \cdot \|_2 \) (assuming appropriate dimensions of course) for \( \text{(2)} \) and finally Lemma S.31 for \( \text{(3)} \).

Due to regularity considerations a bound of the form in the above lemma is to be expected.

**Example 2.5.** 1. Let’s consider the case \( L = 1 \) and assume that \( \|a_{i_1}^{[1]}\|_2 \leq 1 \) for \( i_1 = 1, \ldots, m[1] \). Specialising Lemma 2.4 to this setting we find that (note that
\[ \|a_{i_1}^{[1]}\|_2 \leq 1 \implies \|(a_{i_1}^{[1]})_i\| \leq 1 \quad \text{for all } i = 1, \ldots, m^{[0]} \]

\[ \|X - Y\|_F \leq \kappa_2^{[1]} c m^{[1]} \sqrt{N}, \]

which is exactly the result used in [35, Theorem 2.2] (cf. their equation (2.9)).

2. Now consider the case \( L = 2 \) and assume that \( \|a_{i_1}^{[0]}\|_2 \leq 1 \) for \( i_1 = 1, \ldots, m^{[0]}, \quad l = 1, 2 \). Specialising Lemma S.31 to this setting leads to

\[ \|X - Y\|_F \leq 2 \max\{\kappa_1^{[1]} \kappa_2^{[2]}, \kappa_1^{[2]} \kappa_2^{[1]}\} m^{[2]} m^{[1]} \sqrt{N} \epsilon \]

which is exactly the bound from [82, Lemma 15, Proposition 3].

Next, the smallest singular value of \( X \) has to be lower bounded. Unfortunately, in general no notrivial bound can be found, hence we have to make an additional assumption.

Defining

\[ J[f] = \int_{S^{m^{[0]}}} \nabla f(x) \nabla f(x)^T d\mu_{S^{m^{[0]}}}(x), \]

and making an assumption on the singular values of this matrix the next result can be derived.

**Lemma 2.6. Lower bounding smallest singular value**

Let \( s \in (0, 1) \) be arbitrary, \( X_1, \ldots, X_N \) be drawn i.i.d. uniformly from \( S^{m^{[0]-1}} \) and build the matrix \( X \). Additionally, assume that \( \|a_{i_l}^{[l]}\|_2 \leq 1 \) for \( i = 1, \ldots, m^{[l]}, \quad l = 1, \ldots, L \). Assuming that \( \sigma_{m^{[1]}}(J[f]) \geq \alpha \) for some \( \alpha > 0 \) we have

\[ \sigma_{m^{[1]}}(X) \geq \sqrt{N \alpha (1 - s)} \]

with probability at least \( 1 - m^{[1]} \exp \left( - \frac{N \alpha^2}{2 C^{[2,6]} m^{[1]}} \right) \), where

\[ C^{[2,6]} = \left( \prod_{l=1}^{L} m^{[l]} \right) \left( \prod_{l=1}^{L} \kappa_1^{[l]} \right)^2 m^{[L]} \]

**Remark 2.7.** This lemma and the following proof is a direct generalization of [35, Lemma 2.1], in particular setting \( L = 1 \) recovers exactly the latter result. One can generalize this lemma easily to the case of different norms of the weights by appropriately modifying Step 2 in the following proof and changing the constant \( C^{[2,6]} \). However, for simplicity we won’t use this option in the following.

**Proof.** The strategy is to use a matrix version of the classical Chernoff concentration inequality which is recalled in the appendix, cf. Theorem S.18.

**Step 1** First we have to transfer our setting from one random matrix \( X \) to a sum of i.i.d. random matrices. Let \( w_1, \ldots, w_{m^{[1]}} \) be an orthonormal basis of \( A \) and define

\[ \tilde{W} = \begin{pmatrix} w_1^T \\ \vdots \\ w_{m^{[1]}}^T \end{pmatrix} \]
Obviously

\[ \sigma_j(X) = \sigma_j(\bar{W}^{[1]}X) = \sqrt{\sigma_j(\bar{W}X X^T \bar{W}^T)}, \quad j = 1, \ldots, N \]

since the columns of \( X \) are vectors from \( A \) and singular values are invariant under unitary (here orthogonal) transformations. Furthermore

\[ \bar{W}X X^T \bar{W}^T = \bar{W} \left( \sum_{k=1}^{N} \nabla f(X_k) \nabla f(X_k)^T \right) \bar{W}^T = \sum_{k=1}^{N} \bar{W} \nabla f(X_k) \nabla f(X_k)^T \bar{W}^T. \]

Defining \( Z_k = \bar{W} \nabla f(X_k) \nabla f(X_k)^T \bar{W}^T \), which are i.i.d. random matrices, we see that \( \bar{W}X X^T \bar{W}^T \) is indeed a sum of i.i.d. random matrices.

**Step 2** We have to upper bound the singular values of the \( X_k \). For any \( x \in D^{[0]} \)

\begin{align*}
\sigma_1(\bar{W} \nabla f(x) \nabla f(x)^T \bar{W}^T) & \overset{\text{(1)}}{=} \sigma_1(\nabla f(x) \nabla f(x)^T) \\
& \overset{\text{(2)}}{=} \| \nabla f(x) \|_2^2 \\
& \overset{\text{(3)}}{=} \| W^{[1]} T G^{[1]}(x) \ldots W^{[L]} T (g^{[L]})'(x) \|_2^2 \\
& \overset{\text{(4)}}{=} \left( \prod_{l=1}^{L} \| W^{[l]} T \|_{2 \rightarrow 2} \right)^2 \left( \prod_{l=1}^{L-1} \| G^{[l]}(x) \|_{2 \rightarrow 2} \right)^2 \| (g^{[L]})'(x) \|_2^2 \\
& \overset{\text{(5)}}{=} \left( \prod_{l=1}^{L} \sqrt{m^{[l]}} \right)^2 \left( \prod_{l=1}^{L-1} \kappa^{[l]}_1 \right)^2 m^{[L]} \left( \kappa^{[L]}_1 \right)^2 \\
& = \left( \prod_{l=1}^{L} m^{[l]} \right) \left( \prod_{l=1}^{L-1} \kappa^{[l]}_1 \right)^2 m^{[L]} = C^{[2.6]} 
\end{align*}

where we used the orthonormality of the \( w_1, \ldots, w_{m^{[1]}} \) for (1), the definition of SVD for (2), Corollary 1.4 for (3), the submultiplicativity of the operator norm (and \( \| \cdot \|_{2 \rightarrow 2} \leq \| \cdot \|_F \)) for (4), the definition of the various norms for (5) and the definition of \( \kappa^{[l]}_1 \) for (6).

**Step 3** We can now use Matrix Chernoff. By assumption

\[ \mu_{\min} := \sigma_{m^{[1]}} \left( \sum_{j=1}^{N} \mathbb{E}[X_j] \right) = \sum_{j=1}^{N} \sigma_{m^{[1]}}(\mathbb{E}[X_j]) = NJ[f] \geq N\alpha, \]
hence

\[
P \left[ \sigma_{m[1]}(X) \leq \sqrt{N\alpha(1-s)} \right] = P \left[ \sqrt{\sigma_{m[1]}(\tilde{W}XX^T\tilde{W}^T)} \leq \sqrt{N\alpha(1-s)} \right] \\
= P \left[ \sigma_{m[1]}(\tilde{W}XX^T\tilde{W}^T) \leq N\alpha(1-s) \right] \\
\leq P \left[ \sigma_{m[1]}(\tilde{W}XX^T\tilde{W}^T) \leq N\mu_{\min}(1-s) \right] \\
\stackrel{(\dag)}{\leq} m^{[1]} \exp \left( -\frac{\mu_{\min} s^2}{2C^{[2.6]}} \right) \\
\leq m^{[1]} \exp \left( -\frac{N\alpha s^2}{2C^{[2.6]}} \right)
\]

where we used Theorem S.18 for (\dag). The result follows simply by taking the complementing event. \qed

Corresponding to [34, Remark 2] one can can improve the previous result if the weights are nearly orthogonal.

**Corollary 2.8. Nearly orthogonal case**

Consider the situation of Lemma 2.6. If the weights in layer \( l = 1, \ldots, L \) are \( \epsilon_{qo}^{[l]} \)-nearly orthogonal, then we change the constant \( C^{[2.6]} \) to

\[
C^{[2.8]} = \left( \prod_{l=1}^{L} (1 + \epsilon_{qo}^{[l]} \right)^2 \left( \prod_{l=1}^{L} \kappa_{1}^{[l]} \right)^2 m^{[L]}
\]

**Proof.** Using Proposition S.11 (i) simply bound the expression

\[
\left( \prod_{l=1}^{L} \| W^{[l]}T \|_{2\rightarrow2} \right)^2
\]

in the proof of Lemma 2.6 by

\[
\left( \prod_{l=1}^{L} (1 + \epsilon_{qo}^{[l]} \right)^2
\]

\qed

**Example 2.9.** For \( L = 1 \) we find that

\[
\sigma_{m[1]}(X) \geq \sqrt{N\alpha(1-s)}
\]

with probability at least \( 1 - m^{[1]} \exp \left( -\frac{N\alpha s^2}{2m^{[1]}(\kappa_{1}^{[1]})^2} \right) \), which is exactly the result from [34, Lemma 4].

If \( a_{1}^{[1]}, \ldots, a_{m[1]}^{[1]} \) are \( \epsilon_{qo}^{[1]} \)-nearly orthogonal, then according Corollary 2.8 the probability changes to at least

\[
1 - m^{[1]} \exp \left( -\frac{N\alpha s^2}{2(1 + \epsilon_{qo}^{[1]} m^{[1]}(\kappa_{1}^{[1]})^2} \right),
\]

which is exactly the result from [34, Remark 2].
Finally, with the previous two lemmata Wedin’s bound can be used to arrive at the following Theorem.

**Theorem 2.10. Approximation error bound**

Let $\epsilon > 0$, $N \in \mathbb{N}$, $s \in (0, 1)$ and a neural network function $f$ be given and assume that $\sigma_{m[1]}(J[f]) \geq \alpha$ for some $\alpha > 0$. Applying Algorithm 1 on $f$ with $N$ samples we have that

$$
\|P_A - P_{\tilde{A}}\|_F \leq P \frac{2C^{[S;31]}\epsilon}{\sqrt{\alpha(1-s)} - 2C^{[S;31]}\epsilon}
$$

with probability at least $1 - m^{[1]} \exp\left(-\frac{N\alpha s^2}{2C^{[2;3]}[1]}\right)$ where

**Proof.** We have

$$
\|P_A - P_{\tilde{A}}\|_F \leq \|V_1V_1^T - \tilde{V}_1\tilde{V}_1^T\|_F
$$

where the definition of the projections has been used for $\mathbb{1}$; Wedin’s bound in the form S.2 has been used for $\mathbb{2}$ and for $\mathbb{3}$ the invariance under transposition. Furthermore, Weyl’s inequality in the form

$$
|\sigma(X^T) - \sigma(Y^T)| \leq \|X - Y\|_F
$$

has been applied for $\mathbb{4}$.

Combining the deterministic upper bound on $\|X^T - Y^T\|_F$ from Lemma 2.4 and the probabilistic lower bound on $\sigma_{m[1]}(X)$ from Lemma 2.6 then shows that

$$
\|P_A - P_{\tilde{A}}\|_F \leq \frac{2\|X - Y\|_F}{\sigma_{m[1]}(X)} - \|X - Y\|_F
\leq \sqrt{N}\sqrt{\alpha(1-s)} - 2\sqrt{NC^{[S;31]}\epsilon}
\leq \frac{2\sqrt{N}\sqrt{\alpha(1-s)} - 2\sqrt{NC^{[S;31]}\epsilon}}{2C^{[S;31]}}
\leq \frac{\sqrt{\alpha(1-s)} - 2C^{[S;31]}\epsilon}{\alpha(1-s) - 2C^{[S;31]}\epsilon}
$$

with probability at least

$$
1 - m^{[1]} \exp\left(-\frac{N\alpha s^2}{2C^{[2;3]}[1]}\right).
$$

$\square$
Corollary 2.11. Consider the situation of Theorem 2.10. If the weights in layer \( l = 1, \ldots, L \) are \( \epsilon_q \)-nearly orthogonal, then the probability in this Theorem can be changed to at least

\[
1 - m^{[l]} \exp \left( - \frac{N \alpha s^2}{2(C^{[l,8]})^2} \right)
\]

Proof. Immediate from Theorem 2.10 and Corollary 2.8.

\[\square\]

2.3. Numerical experiments

We now illustrate the approach presented and analysed above with some numerical simulations. First, we consider networks with only a few layers. For this we sample for each fixed architecture 50 networks (with nearly orthogonal weights) and perform the subspace approximation with different number of samples. The results for one and two hidden layers are presented in Figure 2.1. The uniform approximation errors are estimated by sampling 500 points randomly from the unit ball and then taking the maximum absolute deviation between the approximated function and the real function value. Note that although the methods developed later on require that in a certain sense higher layers are thinner than lower layers the dimensionality reduction scheme presented here does not rely on it. Not surprisingly, applying the method on a network with a rather broad second hidden layer still leads to good results, cf. Figure 2.2. How does the method behave for deeper layers? Figure 2.3 shows the performance for networks with three hidden layers as well as even deeper networks. It is clear that the approach still works in this case. Note however that for deeper networks the finite difference approximation gets more problematic, cf. also Remark S.33.

Excursion: ReLU networks

Since the method described here relies on differentiation it is strictly speaking not applicable to ReLU networks. However, it is clear that as long as not too many of the samples are in the vicinity of the kinks of the ReLU and that all ReLUs are activated at least for one sample the method should still work. Indeed, as Figure 2.4 shows the performance is only mildly degraded. However, the situation changes for deeper ReLU networks, cf. Figure 2.5. One possible explanation is that the number of possible different affine-linear regions increases exponentially with depth [74] and hence also the chance that a sample point is close to a problematic kink. Furthermore, note from a certain number of samples on the performance gets worse with increasing sample size. This is not surprising since having more samples the likelihood of encountering a kink of a ReLU increases.
Figure 2.1.: Illustrating the subspace approximation method for networks with one and two hidden layers.

Figure 2.2.: Illustrating the subspace approximation method for a network with two hidden layers and a rather wide second layer (architecture $[100, 20, 150]$).
Figure 2.3.: Illustrating the subspace approximation method for a network with three hidden layers (architecture $[100, 20, 10, 5]$) and deeper networks (architecture $[100, 20, 10, \ldots, 10]$).
Figure 2.4.: Illustrating the subspace approximation method for a networks with one and two hidden layers and ReLU activations.

Figure 2.5.: Illustrating the subspace approximation method for deeper network with ReLU activations.
3. Learning the auxiliary matrix space

3.1. Introduction and setup

The following developments are direct generalizations of [82, Chapter 3] from \( L = 2 \) to \( L \geq 2 \). From Corollary 1.6 it is known that the Hessian for a \( L \)-layer network is of the form

\[
H[f](x) = \sum_{i_L=1}^{m_L} s_{i_L}^{[L]}(x) \left[ v_{i_L}^{[L]}(x) \otimes v_{i_L}^{[L]}(x) \right] + \ldots + \sum_{i_1=1}^{m_1} s_{i_1}^{[1]}(x) \left[ v_{i_1}^{[1]}(x) \otimes v_{i_1}^{[1]}(x) \right],
\]

i.e. for each \( x \) the Hessian is a linear combination of certain rank 1 matrices \( v_{i_l}^{[l]}(x) \otimes v_{i_l}^{[l]}(x) \). These matrices depend on \( x \), though the dependence is only through terms of the form \( z_{i_l}^{[1]}(x) = \langle a_{i_l}^{[1]}, x \rangle \). For an integrable random \( m^{[0]} \)-dimensional \( X \) with mean \( \mathbb{E}[X] = 0 \) we get that \( z_{i_l}^{[1]}(X) \) are integrable random variables with mean

\[
\mathbb{E}[\langle a_{i_l}^{[1]}, X \rangle] = \langle a_{i_l}^{[1]}, \mathbb{E}[X] \rangle = 0.
\]

This situation applies for example for \( X \) sampled uniformly and independently from a sphere or from a ball with center at 0. Intuitively, if \( X_1, \ldots, X_N \) are sampled independently and identically distributed in this manner the random matrices \( H[f](X_k) \) should concentrate around the space

\[
\mathcal{M} = \text{span}_{\mathbb{R}^{m^{[0]} \times m^{[0]}}} \{ v_{i_l}^{[l]}(0) \otimes v_{i_l}^{[l]}(0) \mid l = 1, \ldots, m^{[l]}, l = 1, \ldots, L \}.
\]

For convenience we define

\[
\begin{align*}
v_{i_l}^{[l]} &= v_{i_l}^{[l]}(0) \\
V^{[l]} &= V^{[l]}(0) \\
G^{[l]} &= G^{[l]}(0)
\end{align*}
\]

In the following we will approximate the space \( \mathcal{M} \), henceforth called auxiliary matrix space, which in turn will be used in the next section to approximate all the \( v_{i_l}^{[l]} \), which of course include also the \( a_{i_1}^{[1]} \). Based on the previous considerations a strategy similar to Step 1 can be derived for approximating \( \mathcal{M} \): Assume \( X_1, \ldots, X_N \) are distributed as described above. If \( H[f](X_k), k = 1, \ldots, N \) concentrate indeed around \( \mathcal{M} \) and all \( v_{i_l}^{[l]} \otimes v_{i_l}^{[l]} \) are suitably separated, then finding a \( m = m^{[1]} + \ldots + m^{[L]} \)-dimensional matrix...
subspace minimizing an appropriate error measure (e.g. \( \| \cdot \|_F \)) w.r.t. the sampled \( H[f](X_k) \) should lead to a space close to \( \mathcal{M} \).

In order to make these considerations precise we have to work with vectorizations. Define

\[
M = \text{span}_{\mathbb{R}^{(m_l)_0 \times 2}} \left\{ \text{vec}(v^{[l]}_i \otimes v^{[l]}_l) \mid i_l = 1, \ldots, m^{[l]}, l = 1, \ldots, L \right\},
\]

then vectorization is an isometric isomorphism between \( \mathcal{M}, \| \cdot \|_F \) and \( (M, \| \cdot \|_2) \). Finding a subspace of \( \mathbb{R}^{(m_0)_0 \times (m_0)_0} \) that is in some sense close to \( M \) should then be enough to find a subspace of \( \mathbb{R}^{m_0 \times m_0} \) that is close to \( \mathcal{M} \).

The heuristic considerations from above can now be made rigorous. Let \( x_1, \ldots, x_N \in \mathbb{R}^{m_0} \) be some input points and define

\[
Y = \left( \text{vec}(H[f](x_1)) \mid \ldots \mid \text{vec}(H[f](x_N)) \right)
\]

After vectorization the corresponding error measure is simply the Euclidean norm, and similar to Step 1 minimizing this norm is equivalent to performing an SVD, hence we are lead to the next algorithm. Following [82] the default choice of \( \mathcal{D} \) will be the uniform distribution from the sphere \( S^{m_0-1} \). Assume \( \hat{v}_1, \ldots, \hat{v}_m \) are the columns of \( \hat{V}_1 \). The

**Algorithm 2** Generic auxiliary space approximation scheme

**Require:** Number of samples \( N \), neural network function \( f \), sampling distribution \( \mathcal{D} \)

1. Draw \( x_1, \ldots, x_N \) i.i.d. uniformly from \( \mathcal{D} \)
2. Build \( Y \) from \( x_1, \ldots, x_N \)
3. Compute SVD of \( Y^T \),

\[
Y^T = \begin{pmatrix} \hat{U}_1 & \hat{U}_2 \end{pmatrix} \begin{pmatrix} \hat{\Sigma}_1 & 0 \\ 0 & \hat{\Sigma}_2 \end{pmatrix} \begin{pmatrix} \hat{V}_1^T \\ \hat{V}_2^T \end{pmatrix}
\]

where \( \hat{\Sigma}_1 = \text{diag}(\sigma_1, \ldots, \sigma_m) \) contains the \( m = m^{[1]} + \ldots + m^{[L]} \) largest singular values and \( \hat{V}_1 = (v_1, \ldots, v_m) \) the corresponding right singular vectors

4. **return** \( \hat{V}_1 \)

space

\[
\hat{M} = \text{span}_{\mathbb{R}^{(m_0)_0 \times 2}} \{ \hat{v}_1, \ldots, \hat{v}_m \}
\]

should then be close to \( M \) and \( \hat{v}_1, \ldots, \hat{v}_m \) is an ONB for \( \hat{M} \). Therefore

\[
\hat{M} = \text{span}_{\mathbb{R}^{m_0 \times m_0}} \{ \text{unvec}(\hat{v}_1), \ldots, \text{unvec}(\hat{v}_m) \}
\]

should then be close to \( M \) and \( \text{unvec}(\hat{v}_1), \ldots, \text{unvec}(\hat{v}_m) \) is an ONB for \( \hat{M} \).

Of course \( H[f](x) \) is not available, so a finite difference approximation will be used later on. However, to avoid technicalities the concentration behaviour will be first investigated in the setting of exactly known Hessians.
3.2. Concentration of Hessians

The goal of this section is to show that the space $\tilde{\mathcal{M}}$ is close to $\mathcal{M}$, or equivalently that $\tilde{\mathcal{M}}$ is close to $\mathcal{M}$, and the distance between the spaces will be measured by $\|P_{\tilde{\mathcal{M}}} - P_{\mathcal{M}}\|_F$, cf. Chapter 2.

Before tackling this task we briefly give an overview of the approach since the technical details might obfuscate the general strategy. Note that the developments in this section are directly adapted from [82, Chapter 3] and will culminate in a generalization of [82, Theorem 4]. Our goal is to find a good upper bound

$$\|P_{\tilde{\mathcal{M}}} - P_{\mathcal{M}}\|_F \leq B,$$  \hfill (3.2)

where $B$ is a constant that ideally depends only on general properties of $f$ (like the constants $\kappa_{[l]}$, layer widths $m^{[l]}$ and quasiorthogonality constants $\epsilon_{qo}^{[l]}$) as well as the number of samples in Algorithm 3.1. Since sampling is used only a probabilistic bound can be expected, hence we want that (3.2) holds with high probability.

Note that $\tilde{\mathcal{M}}$ is simply a singular subspace and similar to the situation in Chapter 2 Wedin’s bound might be useful. For this we need the following ingredients:

- A matrix $\hat{Y}$ of the same shape as $Y$ and $B_1 \geq 0$ such that

  $$\|Y - \hat{Y}\|_F \leq B_1$$

  with high probability (recall that $Y$ is a random matrix).

- Given the matrices $Y$ and $\hat{Y}$ from above, consider their SVDs

  $$Y = (U_1 U_2) \begin{pmatrix} \Sigma_1 & 0 \\ 0 & \Sigma_2 \end{pmatrix} \begin{pmatrix} V_1^T \\ V_2^T \end{pmatrix}$$

  $$\hat{Y} = (\hat{U}_1 \hat{U}_2) \begin{pmatrix} \hat{\Sigma}_1 & 0 \\ 0 & \hat{\Sigma}_2 \end{pmatrix} \begin{pmatrix} \hat{V}_1^T \\ \hat{V}_2^T \end{pmatrix}$$

  where $\Sigma_1$ and $\hat{\Sigma}_1$ are diagonal matrices with the largest $m = m^{[1]} + \ldots + m^{[L]}$ singular values of $Y$ and $\hat{Y}$, respectively. We need a number $B_2 > 0$ such that with high probability

  $$\min_{i=1,\ldots,m} |\sigma_i - \hat{\sigma}_j| \geq B_2 \quad \min_{j=m+1,\ldots,(m[0]^2)} |\sigma_i| \geq B_2$$

  where $\sigma_i$ and $\hat{\sigma}_i$ is the $i$-th largest singular value of $Y$ and $\hat{Y}$, respectively.

Using Wedin’s bound (together with a theorem of Weyl) these two ingredients lead immediately to a bound of the form

$$\|P_{\tilde{\mathcal{M}}} - P_{im(\hat{V}_1)}\|_F \leq \hat{B}$$
for some $\hat{B} > 0$ that holds with high probability. What matrix $\hat{Y}$ should we choose? Recall the following important trick: In Chapter 2 finding a number $B_2$ was easy since there we had $\hat{\Sigma}_2 = 0$ (in the notation of the current chapter), since in this case the bound involving $B_2$ reduces to

$$\min_{i=1,\ldots,m} |\sigma_i| \geq B_2$$

(again with high probability since $Y$ is random). In turn, $\hat{\Sigma}_2 = 0$ was achieved by simply using a matrix $\hat{Y}$ with rank less or equal to $m$ (in the notation of the current chapter). A good way to ensure this is to choose a $\hat{Y}$ such that $\text{im}(\hat{Y}) \subseteq M$. An obvious choice of $\hat{Y}$ that fulfills all the requirements above is then

$$\hat{Y} = \left(\text{vec}(P_M H[f](x_1)) \cdots \text{vec}(P_M H[f](x_N))\right).$$

Intuitively, with this setup $\|Y - \hat{Y}\|_F$ measures how far away the actual realizations of the Hessians are from $\mathcal{M}$ (recall that in general $H[f](x) \notin \mathcal{M}$ since $v_i^{[l]}$ depend on $x$ for $l \geq 2$). In other words, it measures the error introduced by the projections. The second ingredient

$$\min_{i=1,\ldots,m} |\sigma_i| \geq B_2,$$

(with high probability) can be interpreted as requiring that $H[f](X_k)$ are well-spread (and e.g. not concentrating on a very low-dimensional linear subspace, which could make $\sigma_m$ rather small), but not too much.

Finally, we have to get from (3.2) to (3.2). Intuitively, the richer the image of $\hat{Y}$ is (recall that the range of $\hat{Y}$ is contained in $\mathcal{M}$, hence there’s no risk of contamination from the outside of $\mathcal{M}$), the closer (3.2) is to (3.2). If $\text{im}(\hat{Y}) = \mathcal{M}$ then the two bounds even coincide. A bootstrap-type argument will show that this indeed happens if (3.2) is good enough.

In this section we will work through the strategy outlined above. First, a deterministic bound on $\|Y - \hat{Y}\|_F$ is derived (i.e. it will explicitly involve arbitrary but fixed query points $x_1,\ldots,x_N$) in Lemma 3.7. Using properties of the known distribution of the query points this bound is then used in Proposition 3.9 to derive a bound of the form

$$\|Y - \hat{Y}\|_F \leq B_1,$$

that holds with high probability. Next, a lower bound on the $m$-th largest singular value of $Y$ is presented (with high probability). Note that in contrast to the situation in Chapter 2 a generalization of Matrix Chernoff has to be used since in general the $m$-th largest singular value will be an inner singular value. Combining everything using Wedin’s theorem leads finally to a bound of the form (3.2).

**Technical preparations**

We now start to work through the approach just outlined. The first goal is to derive the deterministic bound alluded to above, which is done using the following intermediate results.
Lemma 3.1. Consider a neural network with $L \geq 2$ layers and assume that the weights in layers $2 \leq l \leq L$ are $\epsilon_{q_0}^{[l]}$-quasiorthogonal. With the notation from Corollary 1.6 we have

$$\|S^{[l]}(x)\|_F \leq \kappa_2^{[l]} \sqrt{m^{[l]}} \left( \prod_{k=l+1}^{L} (1 + \epsilon^{[k]}) \right) \left( \prod_{k=l+1}^{L} \kappa_1^{[k]} \right) =: C_l^{[3,1]}$$

for all $1 \leq l \leq L$.

Example 3.2. 1. For all $L \geq 2$ we get for $l = L$ that

$$\|S^{[l]}(x)\|_F \leq \sqrt{m^{[l]}} \kappa_2^{[L]}$$

Note that for $L = 2$ this is the same bound used in the proof of [82, Theorem 4] via [82, Lemma 12].

2. For $l = L - 1$ we get

$$\|S^{[l]}(x)\|_F \leq \kappa_2^{[L-1]} \sqrt{m^{[L]}} \kappa_1^{[L]} (1 + \epsilon^{[L]})$$

Note that for $L = 2$ this is exactly the bound used in the proof of [82, Lemma 14] (where we have $\|t_x\|_2 = \|S^{[2]}(x)\|_F$, using the notation of [82]).

Proof. For all $1 \leq l \leq L$ we find

$$\|S^{[l]}(x)\|_F = \|\text{diag}(s^{[l]}_{i_l}(x), i_l = 1, \ldots, m^{[l]})\|_F$$

$$= \|(s^{[l]}_{i_l}(x))_{i_l=1,\ldots,m^{[l]}}\|_2$$

If $l = L$ one can use the bound

$$\|S^{[l]}(x)\|_F = \sqrt{m^{[L]}} \left( \sum_{i_L=1}^{m^{[L]}} \left( g^{[L]}_{i_L} \right)^{\#} \left[ z^{[L]}_{i_L}(x) \right]^2 \right)$$

$$\leq \kappa_2^{[L]} \sqrt{m^{[L]}}$$

and the claim of the Lemma follows by the usual convention $\prod_{k=l+1}^{L} = 1$. Assume now $l \leq L - 1$, then we get

$$\|S^{[l]}(x)\|_F = \left\| \left( g^{[L]}_{i_L} \right)^{\#} \left[ z^{[L]}_{i_L}(x) \right] \sum_{i_{L+1}} \cdots \sum_{i_L} \left( g^{[L+1]}_{i_{L+1}} \right)^{\#} \left[ z^{[L+1]}_{i_{L+1}}(x) \right] \cdots \left( g^{[L]}_{i_L} \right)^{\#} \left[ z^{[L]}_{i_L}(x) \right] \left( a^{[L]}_{i_{L-1}} \right)_{i_{L-1}} \cdots \left( a^{[L]}_{i_{L-1}} \right)_{i_{L-1}} \right\|_2$$

$$\leq \kappa_2^{[L]} \left\| \left( \sum_{i_{L+1}} \cdots \sum_{i_L} \left( g^{[L+1]}_{i_{L+1}} \right)^{\#} \left[ z^{[L+1]}_{i_{L+1}}(x) \right] \cdots \left( g^{[L]}_{i_L} \right)^{\#} \left[ z^{[L]}_{i_L}(x) \right] \left( a^{[L]}_{i_{L-1}} \right)_{i_{L-1}} \cdots \left( a^{[L]}_{i_{L-1}} \right)_{i_{L-1}} \right\|_2$$

(3.3)
by definition of $\kappa_{l}^{[I]}$. Now, for $l = L - 1$

$$
\left\| \sum_{i_{L} = 1}^{m_{[L]}} \left( g_{L}^{[i_{L}]} \right)' \left[ z_{L}^{[i_{L}]}(x) \right] \left( a_{L}^{[i_{L}]} \right)_{i_{L} - 1} \right\|_{2} 
\leq \| W^{[L]} \|_{2 \rightarrow 2} \left\| \sum_{i_{L} = 1}^{m_{[L]}} \left( g_{L}^{[i_{L}]} \right)' \left[ z_{L}^{[i_{L}]}(x) \right] \left( a_{L}^{[i_{L}]} \right)_{i_{L} - 1} \right\|_{2}
\leq \| W^{[L]} \|_{2 \rightarrow 2} \left( 1 + \epsilon_{q_{o}}^{[L]} \right) \sqrt{m_{[L]} \kappa_{l}^{[L]}}
$$

where the definition of matrix-vector multiplication was used for (1), the definition of the operator norm for (2) and Proposition S.11 (i) together with the boundedness of $\left( g_{L}^{[i_{L}]} \right)'$ for (3). Similarly, for $1 \leq l < L - 1$ (assuming of course that $L \geq 3$) we get

$$
\left\| \sum_{i_{L} = 1}^{m_{[L]}} \sum_{i_{L+1} = 1}^{m_{[L+1]}} \left( g_{L+1}^{[i_{L+1}]} \right)' \left[ z_{L+1}^{[i_{L+1}]}(x) \right] \left( a_{L+1}^{[i_{L+1}]} \right)_{i_{L+2} \ldots i_{L+1}} \right\|_{2}
\leq \| W^{[L]} \|_{2 \rightarrow 2} \left\| \sum_{i_{L} = 1}^{m_{[L]}} \left( g_{L}^{[i_{L}]} \right)' \left[ z_{L}^{[i_{L}]}(x) \right] \left( a_{L}^{[i_{L}]} \right)_{i_{L} - 1} \right\|_{2}
\leq \| W^{[L]} \|_{2 \rightarrow 2} \left( 1 + \epsilon_{q_{o}}^{[L]} \right) \sqrt{m_{[L]} \kappa_{l}^{[L]}}
$$

where (1) is a simply rearrangement of the terms, for (2) the definition of matrix-vector multiplication was used (same as for the case $l = L - 1$, see above), and for (3) the definition of the operator norm as well as the definition of $\kappa_{l}^{[I]}$ was used and these three steps were repeated several time. Finally, for (4) the bound from above for the case $l = L - 1$ was used and for (5) Proposition S.11 (i) was used to bound the terms $\| W^{[l]} \|_{2 \rightarrow 2}$. We can now continue with (3.3) to get

$$
\| S^{[l]}(x) \|_{F} \leq \kappa_{l}^{[I]} \left( \sum_{i_{L} = 1}^{m_{[L]}} \left( g_{L}^{[i_{L}]} \right)' \left[ z_{L}^{[i_{L}]}(x) \right] \left( a_{L}^{[i_{L}]} \right)_{i_{L} - 1} \right)_{i_{L} = 1, \ldots, m_{[L]}}
\leq \kappa_{l}^{[I]} \left( 1 + \epsilon_{q_{o}}^{[L]} \right) \kappa_{l}^{[I]} \left( 1 + \epsilon_{q_{o}}^{[L]} \right) \sqrt{m_{[L]} \kappa_{l}^{[L]}}
= \kappa_{l}^{[I]} \sqrt{m_{[L]}} \left( \prod_{k = l+1}^{L} \left( 1 + \epsilon_{q_{o}}^{[k]} \right) \right) \left( \prod_{k = l+1}^{L} \kappa_{k}^{[k]} \right)
$$

In the following we work frequently with products of certain matrices. Given a neural
network with $L$ layers we define for convenience

$$
\Pi^{[l_1,l_2]}(x) = G^{[l_1]}(x)W^{[l_1+1]T}G^{[l_1+1]}(x) \cdots W^{[l_2]T}G^{[l_2]}(x)
$$

$$
\Pi^{[l_1]}(x) = G^{[l_1]}(x)
$$

where $1 \leq l_1 < l_2 \leq L$.

**Lemma 3.3.** Consider a neural network with $L$ layers and quasiorthogonal weights. We have for all $1 \leq l_1 \leq l_2 \leq L$ that

$$
\|\Pi^{[l_1,l_2]}(x)\|_{2 \rightarrow 2} \leq \left( \prod_{k=l_1}^{l_2} \kappa_1^{[k]} \right) \left( \prod_{k=l_1+1}^{l_2} (1 + \epsilon^{[k]}_{q_0}) \right) =: C^{[3,3]}_{l_1,l_2}
$$

Note that for $l_1 = l_2 = l$ we get the simple bound

$$
\|\Pi^{[l]}(x)\|_{2 \rightarrow 2} \leq \kappa_1^{[l]}
$$

**Proof.** The case $l_1 = l_2$ is simple,

$$
\|\Pi^{[l_1,l_2]}(x)\|_{2 \rightarrow 2} = \|G^{[l_2]}(x)\|_{2 \rightarrow 2} \leq \kappa_1^{[l_2]}
$$

For $l_1 < l_2$ we find

$$
\|\Pi^{[l_1,l_2]}(x)\|_{2 \rightarrow 2} = \|G^{[l_1]}(x)W^{[l_1+1]T}G^{[l_1+1]}(x) \cdots W^{[l_2]T}G^{[l_2]}(x)\|_{2 \rightarrow 2}
$$

$$
\overset{1}{\leq} \|G^{[l_1]}(x)\|_{2 \rightarrow 2} \|W^{[l_1+1]}\|_{2 \rightarrow 2} \|\Pi^{[l_1+1,l_2]}(x)\|_{2 \rightarrow 2}
$$

$$
\overset{2}{\leq} \kappa_1^{[l_1]} \left( 1 + \epsilon^{[l_1+1]}_{q_0} \right) \|\Pi^{[l_1+1,l_2]}(x)\|_{2 \rightarrow 2}
$$

$$
\overset{3}{\leq} \cdots \left( \prod_{k=l_1}^{l_2-1} \kappa_1^{[k]} \right) \left( \prod_{k=l_1+1}^{l_2} (1 + \epsilon^{[k]}_{q_0}) \right) \|G^{[l_2]}(x)\|_{2 \rightarrow 2}
$$

$$
\overset{4}{=} \left( \prod_{k=l_1}^{l_2} \kappa_1^{[k]} \right) \left( \prod_{k=l_1+1}^{l_2} (1 + \epsilon^{[k]}_{q_0}) \right)
$$

where the submultiplicativity of $\cdot \|_{2 \rightarrow 2}$ and the definition of $\Pi^{[l_1+1,l_2]}$ was used for $\overset{1}{\leq}$, the definition of $\kappa_1^{[l_1]}$ and the quasiorthogonality of the weights was used for $\overset{2}{\leq}$, steps $\overset{\circ}{\leq}$ and $\overset{\circ}{\leq}$ were simply repeated for $\overset{3}{\leq}$ and finally we used again the definition of $\kappa_1^{[l_2]}$ for $\overset{4}{=}$. \hfill \Box

The following Lemma generalizes the bound $\|\Delta_x\|_{2 \rightarrow 2}$ (using the notation from [82]) used in the proof of [82, Theorem 4].

**Lemma 3.4.** Consider a neural network where each of the $L$ layers has $\epsilon_{q_0}^{[l]}$-quasiorthogonal weights, and $\|\epsilon_{q_0}^{[l]}\|_2 \leq 1$, $l = 1, \ldots, L$. For each layer $l$ we then have

$$
\|G^{[l]}(x) - G^{[l]}(0)\|_{2 \rightarrow 2} \leq C^{[3,4]}_l \|W^{[1]}x\|_\infty
$$
with
\[ C^{[3,4]}_1 =: \kappa_2^{[1]} \]
and for \( l = 2, \ldots, L \)
\[ C^{[3,4]}_l =: \kappa_2^{[l]} \left( \prod_{k=1}^{l-1} \kappa_1^{[k]} \right) \left( \prod_{k=2}^{l-1} (1 + \epsilon_0^{[k]}) \right) \sqrt{\lambda^{[1]}} \]

**Proof.** First recall that for diagonal matrices
\[ \| \text{diag}(x) \|_{2 \rightarrow 2} = \| x \|_{\infty} \]
for any \( x \in \mathbb{R}^n \). The case \( l = 1 \) is simple:
\[ \| G^{[1]}(x) - G^{[1]}(0) \|_{2 \rightarrow 2} = \max_{i_1,1,\ldots,m^{[1]}} \left| \left( g_{i_1}^{[1]} \right)' \left[ \langle a_{i_1}^{[1]}, x \rangle \right] - \left( g_{i_1}^{[1]} \right)' \left[ \langle a_{i_1}^{[1]}, 0 \rangle \right] \right| \]
\[ \leq \kappa_2^{[1]} \left( \max_{i_1,1,\ldots,m^{[1]}} |\langle a_{i_1}^{[1]}, x \rangle| \right) \]
\[ = \kappa_2^{[1]} \| W^{[1]} x \|_{\infty} \]
where we used the Lipschitz continuity of all \( \left( g_{i_1}^{[1]} \right)' \). Similarly, for \( 2 \leq l \leq L \) we get
\[ \| G^{[l]}(x) - G^{[l]}(0) \|_{2 \rightarrow 2} = \max_{i_1,1,\ldots,m^{[l]}} \left| \left( g_{i_1}^{[l]} \right)' \left[ \langle a_{i_1}^{[l]}, y^{[l-1]}(x) \rangle \right] - \left( g_{i_1}^{[l]} \right)' \left[ \langle a_{i_1}^{[l]}, y^{[l-1]}(0) \rangle \right] \right| \]
\[ \leq \kappa_2^{[l]} \left( \max_{i_1,1,\ldots,m^{[l]}} |\langle a_{i_1}^{[l]}, y^{[l-1]}(x) - y^{[l-1]}(0) \rangle| \right) \]
\[ \leq \kappa_2^{[l]} \| y^{[l-1]}(x) - y^{[l-1]}(0) \|_{2} \]
\[ = \kappa_2^{[l]} \sum_{i_{l-1}=1}^{m^{[l-1]}} \left| g_{i_{l-1}}^{[l-1]} \left[ \langle a_{i_{l-1}}^{[l-1]}, y^{[l-2]}(x) \rangle \right] - g_{i_{l-1}}^{[l-1]} \left[ \langle a_{i_{l-1}}^{[l-1]}, y^{[l-2]}(0) \rangle \right] \right| \]
\[ = \kappa_2^{[l]} \| y^{[l-1]}(x) - y^{[l-1]}(0) \|_{2} \]
where Lipschitz continuity of all \( \left( g_{i_1}^{[l]} \right)' \) has been used for \( 1 \) and Cauchy-Schwarz for \( 2 \) together with \( \| a_{i_1}^{[l]} \|_2 \leq 1 \) for all \( i_1 = 1, \ldots, m^{[l]} \). For \( l = 2 \) we can continue with
\[ \| G^{[2]}(x) - G^{[2]}(0) \|_{2 \rightarrow 2} \leq \kappa_2^{[2]} \sum_{i_1=1}^{m^{[1]}} \left| g_{i_1}^{[1]} \left[ \langle a_{i_1}^{[1]}, x \rangle \right] - g_{i_1}^{[1]} \left[ \langle a_{i_1}^{[1]}, 0 \rangle \right] \right| ^2 \]
\[ \leq \kappa_2^{[2]} \kappa_1^{[1]} \sqrt{\lambda^{[1]}} \left( \max_{i_1,1,\ldots,m^{[1]}} |\langle a_{i_1}^{[1]}, x \rangle| \right) \]
\[ = \kappa_2^{[2]} \kappa_1^{[1]} \sqrt{\lambda^{[1]}} \| W^{[1]} x \|_{\infty} \]
by using the Lipschitz continuity of all $g_{i_l}$ and $\| \cdot \|_2 \leq \sqrt{m[1]} \cdot \| \cdot \|_\infty$ (in $\mathbb{R}^{m[1]}$). For $l \geq 3$ (assuming of course that $L \geq 3$) we can continue with

$$\|G^{l}[x] - G^{l}(0)\|_{2 \to 2} \leq \kappa_2 \kappa_1 \|W^{l-1}(y^{l-2}(x) - y^{l-2}(0))\|_{2}$$

(3.4)

$$\leq \sum_{j=1}^{l-1} \left( \prod_{j=1}^{k-1} \kappa_1^{[j]} (1 + \epsilon_{qo}^{[j+1]}) \right) C^{[3.4]}(1 + \epsilon_{qo}^{[k+1]}) C^{[3.3]}_{k+1,l} + \sum_{j=1}^{l-1} \left( \prod_{j=1}^{k-1} \kappa_1^{[j]} (1 + \epsilon_{qo}^{[j+1]}) \right) C^{[3.4]}$$

Lemma 3.5. Consider a neural network with $L$ layers, each having $\epsilon_{qo}$-quasiorthogonal weights $l = 1, \ldots, L$. Let $1 \leq l \leq L - 1$ be arbitrary. We have

$$\|\Pi^{[1,l]}(x) - \Pi^{[1,l]}(0)\|_{2 \to 2} \leq C^{[3.5]}_{l} \|W^{[1]}|_{\infty}$$

with

$$C^{[3.5]}_{l} = \sum_{k=1}^{l-1} \left( \prod_{j=1}^{k-1} \kappa_1^{[j]} (1 + \epsilon_{qo}^{[j+1]}) \right) C^{[3.4]}(1 + \epsilon_{qo}^{[k+1]}) C^{[3.3]}_{k+1,l} + \sum_{j=1}^{l-1} \left( \prod_{j=1}^{k-1} \kappa_1^{[j]} (1 + \epsilon_{qo}^{[j+1]}) \right) C^{[3.4]}$$

Proof. We have for all $1 \leq l_1 < l_2 \leq L$

$$\|\Pi^{[l_1,l_2]}(x) - \Pi^{[l_1,l_2]}(0)\|_{2 \to 2}$$

(3.4)

where we used the elementary identity $xy - \bar{x}\bar{y} = (x - \bar{x})\bar{y} + x(y - \bar{y})$ for (1), the triangle inequality and submultiplicativity of $\| \cdot \|_{2 \to 2}$ as well as the definition of $\Pi^{[l_1,l_2]}$ for (2).
We can now continue with

\[
\| \Pi^{[1,l]}(x) - \Pi^{[1,l]}(0) \|_{2-2} \\
\oplus \| G^{[1]}(x) - G^{[1]}(0) \|_{2-2} \| W^{[2]} \|_{2-2} \| \Pi^{[2,l]}(x) \|_{2-2} \\
+ \| G^{[1]}(0) \|_{2-2} \| W^{[2]} \|_{2-2} \| \Pi^{[2,l]}(x) - \Pi^{[2,l]}(0) \|_{2-2}
\]

\[
\oplus \sum_{k=1}^{l-1} \left( \prod_{j=1}^{k-1} \| G^{[j]}(0) \|_{2-2} \| W^{[j+1]} \|_{2-2} \right) \| G^{[k]}(x) - G^{[k]}(0) \|_{2-2} \| W^{[k+1]} \|_{2-2} \| \Pi^{[k+1,l]}(x) \|_{2-2} \\
+ \left( \prod_{j=1}^{k-1} \| G^{[j]}(0) \|_{2-2} \| W^{[j+1]} \|_{2-2} \right) \| \Pi^{[k,l]}(x) - \Pi^{[k,l]}(0) \|_{2-2}
\]

\[
\oplus \sum_{k=1}^{l-1} \left( \prod_{j=1}^{k-1} \| G^{[j]}(0) \|_{2-2} \| W^{[j+1]} \|_{2-2} \right) \| G^{[k]}(x) - G^{[k]}(0) \|_{2-2} \| W^{[k+1]} \|_{2-2} \| \Pi^{[k+1,l]}(x) \|_{2-2} \\
+ \left( \prod_{j=1}^{k-1} \| G^{[j]}(0) \|_{2-2} \| W^{[j+1]} \|_{2-2} \right) \| G^{[l]}(x) - G^{[l]}(0) \|_{2-2}
\]

where we used (3.4) with \( l_1 = 1, l_2 = l \) for (1). For (2) we used (3.4) in a recursive manner for \( l_1 = 2, l_2 = l, \ldots, l_1 = l - 1, l_2 = l \) and collected all the terms. For (3) we used that

\[
\| \Pi^{[l,l]}(x) - \Pi^{[l,l]}(0) \|_{2-2} = \| G^{[l]}(x) - G^{[l]}(0) \|_{2-2}
\]
by definition. Finally, we can bound each factor in the sum,
\[
\|\Pi^{[1,l]}(x) - \Pi^{[1,l]}(0)\|_{2\rightarrow 2} = \\
\kappa(\bigoplus_{k=1}^{l-1} \left( \prod_{j=1}^{k-1} \|G^{[j]}(0)\|_{2\rightarrow 2} \|W^{[j+1]}\|_{2\rightarrow 2} \right) \|G^{[k]}(x) - G^{[k]}(0)\|_{2\rightarrow 2} \|W^{[k+1]}\|_{2\rightarrow 2} \|\Pi^{[k+1,l]}(x)\|_{2\rightarrow 2} \\
+ \left( \prod_{j=1}^{l-1} \|G^{[j]}(0)\|_{2\rightarrow 2} \|W^{[j+1]}\|_{2\rightarrow 2} \right) \|G^{[l]}(x) - G^{[l]}(0)\|_{2\rightarrow 2}
\]
\[
\kappa(\bigoplus_{k=1}^{l-1} \left( \prod_{j=1}^{k-1} \kappa^{[j]}(1 + \epsilon^{[j+1]}_{q_0}) \right) \|G^{[k]}(x) - G^{[k]}(0)\|_{2\rightarrow 2} (1 + \epsilon^{[k+1]}_{q_0}) \|\Pi^{[k+1,l]}(x)\|_{2\rightarrow 2} \\
+ \left( \prod_{j=1}^{l-1} \kappa^{[j]}(1 + \epsilon^{[j+1]}_{q_0}) \right) \|G^{[l]}(x) - G^{[l]}(0)\|_{2\rightarrow 2}
\]
\[
\kappa(\bigoplus_{k=1}^{l-1} \left( \prod_{j=1}^{k-1} \kappa^{[j]}(1 + \epsilon^{[j+1]}_{q_0}) \right) C^{[3,4]}_k (1 + \epsilon^{[k+1]}_{q_0}) C^{[3,3]}_{k+1,l} \\
+ \left( \prod_{j=1}^{l-1} \kappa^{[j]}(1 + \epsilon^{[j+1]}_{q_0}) \right) C^{[3,4]}_l \|W^{[1]}x\|_{\infty}
\]
\[
= \left( \prod_{k=1}^{l-1} \kappa^{[j]}(1 + \epsilon^{[j+1]}_{q_0}) \right) C^{[3,4]}_k (1 + \epsilon^{[k+1]}_{q_0}) C^{[3,3]}_{k+1,l} + \left( \prod_{j=1}^{l-1} \kappa^{[j]}(1 + \epsilon^{[j+1]}_{q_0}) \right) C^{[3,4]}_l \|W^{[1]}x\|_{\infty}
\]
We started with (3.5) for ① and continued using (cf. Proposition S.11 (i) and the previous Lemma)
\[
\|G^{[j]}(x)\|_{2\rightarrow 2} \leq \kappa^{[j]}_1 \\
\|W^{[j]}\|_{2\rightarrow 2} \leq 1 + \epsilon^{[j]}_{q_0}
\]
with appropriate j for ②. Using Lemma 3.4 and 3.3 for ③ and rewriting leads to the result. □

**Example 3.6.** 1. For any \( L \geq 2 \) we get for \( l = 1 \) from the definition of \( \Pi^{[1,1]} \) and Lemma 3.5 and 3.4
\[
\|\Pi^{[1,1]}(x) - \Pi^{[1,1]}(0)\|_{2\rightarrow 2} = \|G^{[1]}(x) - G^{[1]}(0)\|_{2\rightarrow 2} \leq \kappa^{[1]}_2 \|W^{[1]}x\|_{\infty},
\]
i.e. Lemma 3.5 reduces to Lemma 3.4. Note that for \( L = 2 \) we get exactly the bound used in [82, Theorem 4] (using the notation \( \Delta_x \|_{2\rightarrow 2} \leq \kappa_2 \|ATx\|_{\infty} \)).

2. For \( L = 3 \) we get for \( l = 2 \)
\[
\|\Pi^{[1,2]}(x) - \Pi^{[1,2]}(0)\|_{\infty} \leq \left( C^{[3,4]}_1 (1 + \epsilon^{[2]}_{q_0}) C^{[3,3]}_{2,2} + \kappa^{[1]}_1 (1 + \epsilon^{[2]}_{q_0}) C^{[3,4]}_2 \right) \|W^{[1]}x\|_{\infty} \\
= \left( \kappa^{[1]}_2 \kappa^{[2]} + \kappa^{[2]}_2 (\kappa^{[1]}_1)^2 \sqrt{m^{[1]}} \right) (1 + \epsilon^{[2]}_{q_0}) \|W^{[1]}x\|_{\infty}
\]
42
The next lemma combines our intermediate results in order to arrive at the deterministic bound announced earlier. Note that it generalizes the variant of [82, Lemma 12] used for [82, Theorem 4] to arbitrary $L \geq 2$.

**Lemma 3.7.** Consider a neural network with $L \geq 2$ layers. Assume the weights in layer $1 \leq l \leq L$ are $\epsilon_q[l]$-quasiorthogonal with $\|a[l]_i\|_2 \leq 1$ and let $f$ be the function represented by the network. Let $x_1, \ldots, x_N \in B^{R_{\text{inst}}}_R$ for some $R > 0$ and define

$$Y = (\text{vec}(H[f](x_1)) \cdots \text{vec}(H[f](x_N)))$$

$$\hat{Y} = (\text{vec}(P_M H[f](x_1)) \cdots \text{vec}(P_M H[f](x_N))).$$

We have then

$$\|Y - \hat{Y}\|_F^2 \leq (C^{(3.7)})^2 \sum_{k=1}^N \|W^{[1]}_k\|_F^2$$

where

$$C^{(3.7)} = \sum_{l=2}^L (1 + \epsilon_q[l])^2 (1 + 2\epsilon_q[l]) C^{[3.1]}_l \left( (C^{[3.5]}_l)^2 + 2C^{[3.5]}_l C^{[3.3]}_l \right)$$

**Proof.** First,

$$\|Y - \hat{Y}\|_F^2 = \sum_{k=1}^N \|\text{vec}(H[f](x_k)) - \text{vec}(P_M H[f](x_k))\|_F^2$$

$$= \sum_{k=1}^N \|H[f](x_k) - P_M H[f](x_k)\|_F^2$$

by the definition of the norms and vec. Now, for any $\|x\|_2 \leq R$

$$\|H[f](x) - P_M H[f](x)\|_F \overset{1}{=} \left\| \sum_{l=1}^L V^{[l]}^T(x)S^{[l]}(x)V^{[l]}(x) - P_M \left( V^{[l]}^T(x)S^{[l]}(x)V^{[l]}(x) \right) \right\|_F$$

$$\overset{2}{=} \sum_{l=1}^L \|V^{[l]}^T(x)S^{[l]}(x)V^{[l]}(x) - P_M \left( V^{[l]}^T(x)S^{[l]}(x)V^{[l]}(x) \right)\|_F$$

$$\overset{3}{=} \sum_{l=2}^L \|V^{[l]^T(x)}S^{[l]}(x)V^{[l]}(x) - P_M \left( V^{[l]^T(x)}S^{[l]}(x)V^{[l]}(x) \right) \|_F$$

$$\overset{4}{=} \sum_{l=2}^L \|V^{[l]^T(x)}S^{[l]}(x)V^{[l]}(x) - V^{[l]^T}S^{[l]}(x)V^{[l]}\|_F$$

where we used Corollary 1.6 and the linearity of $P_M$ for $1$, the triangle inequality (w.r.t. $\| \cdot \|_F$) for $2$, the fact that $V^{[l]^T(x)}S^{[l]}(x)V^{[l]}(x) \in \mathcal{M}$ for $3$ and finally that $P_M$ is an orthogonal projection together with $V^{[l]^T}S^{[l]}(x)V^{[l]} \in \mathcal{M}$ for $4$. 

43
Let now $2 \leq l \leq L$, then
\[
\|V[l]^{T}(x)S[l](x)V[l]^{T}(x) - V[l]^{T}S[l](x)V[l]\|_{F} \\
\leq \|W[l]^{T}T[l](x)W[l]^{T}S[l](x)T[l](x) - W[l]^{T}S[l](x)T[l](x)\|_{F} \\
\leq \|W[l]^{T}\Pi[l-1]T[l](x)S[l](x)W[l]^{T}\Pi[l-1]T[l](x) - W[l]^{T}\Pi[l-1]T[l](x)S[l](x)W[l]^{T}\Pi[l-1]T[l](x)\|_{F} \\
\leq \|W[l]^{T}\Pi[l-1]T[l](x) - W[l]^{T}\Pi[l-1]T[l](x)\|_{F} \\
\leq \|W[l]^{T}\Pi[l-1]T[l](x) - \Pi[l-1]T[l](x)\|_{F} \\
\leq \|W[l]^{T}\Pi[l-1]T[l](x) - \Pi[l-1]T[l](x)\|_{F} \\
\leq \|W[l]^{T}(x)W[l]^{T}(x)\|_{F} \leq (1 + 2\epsilon[l]) \|S[l]\|_{F} \leq (1 + 2\epsilon[l])C[l]^{3,1}
\]
(3.6)
due to Proposition S.11 (ii) and Lemma 3.1. We can now combine our previous results to get
\[
\|V[l]^{T}(x)S[l](x)V[l]^{T}(x) - V[l]^{T}(0)S[l](x)V[l](0)\|_{F} \\
\leq \|W[l]^{T}\Pi[l-1]T[l](x) - \Pi[l-1]T[l](x)\|_{F} \\
\leq \|W[l]^{T}\Pi[l-1]T[l](x) - \Pi[l-1]T[l](x)\|_{F} \\
\leq \|W[l]^{T}\Pi[l-1]T[l](x) - \Pi[l-1]T[l](x)\|_{F} \\
\leq \|W[l]^{T}(x)W[l]^{T}(x)\|_{F} \leq (1 + 2\epsilon[l]) \|S[l]\|_{F} \leq (1 + 2\epsilon[l])C[l]^{3,1}
\]
where we used Cauchy-Schwarz and $x \in B_R^{R[m]}$. We get the simple estimate
\[
\|W[l]^{T}(x)\|_{\infty} = \max_{i=1,\ldots,m[l]} \left| \langle a[l]^{[1]}_{i}, x \rangle \right| \\
\leq \max_{i=1,\ldots,m[l]} \|a[l]^{[1]}_{i}\|_2 \|x\|_2 \\
\leq R,
\]
which was used for (2). We can now sum over $l = 2, \ldots, L$ to get
\[
\|H[f](x) - P_MH[f](x)\|_{F} \leq \sum_{l=2}^{L} \|V[l]^{T}(x)S[l](x)V[l]^{T}(x) - V[l]^{T}(0)S[l](x)V[l](0)\|_{F} \\
\leq \sum_{l=2}^{L} (1 + \epsilon[l])^2 (1 + 2\epsilon[l])C[l]^{3,1} \left( R(C[l]^{3,1})^2 + 2C[l]^{3,1}C[l]^{3,3} \right) \|W[l]^{T}(x)\|_{\infty}
\]
and use this estimate for all $x_1, \ldots, x_N$ to get
\[
\|Y - \hat{Y}\|_{F}^2 \leq \sum_{k=1}^{N} \left( \sum_{l=2}^{L} (1 + \epsilon[l])^2 (1 + 2\epsilon[l])C[l]^{3,1} \left( R(C[l]^{3,1})^2 + 2C[l]^{3,1}C[l]^{3,3} \right) \|W[l]^{T}(x_k)\|_{\infty} \right)^2 \\
= (C[l]^{3,7})^2 \sum_{k=1}^{N} \|W[l]^{T}(x_k)\|_{\infty}^2
\]
44
by using

\[
C^{[3.7]} = \sum_{l=2}^{L} (1 + \epsilon_{qo}^{[1]})(1 + 2\epsilon_{qo}^{[l]})C^{[3.1]} \left( R(C_{l-1}^{[3.5]})^2 + 2C_{l-1}^{[3.5]}C_{1,l-1}^{[3.3]} \right)
\]

Example 3.8. 1. For \( L = 2 \) and \( R = 1 \) we have

\[
C^{[3.7]} = (1 + \epsilon_{qo}^{[1]})(1 + 2\epsilon_{qo}^{[2]})\kappa_2^2 \sqrt{m[2]} \left( \kappa_1^{[1]} + 2\kappa_1^{[1]} \kappa_2^{[1]} \right).
\]

Note that this is exactly the bound used in [82, Theorem 4] (constant \( \tilde{C} \) used for [82, (3.3.15)]).

2. For \( L = 3 \) and \( R > 0 \) we get (using Examples 3.2 and 3.6)

\[
C^{[3.7]} = (1 + \epsilon_{qo}^{[1]})(1 + 2\epsilon_{qo}^{[3]})\kappa_2^2 \kappa_1^{[3]} \left( R(\kappa_2^{[1]} \kappa_1^{[1]}) + \kappa_2^{[2]}(\kappa_1^{[1]})^2 \sqrt{m[1]} \right) (1 + \epsilon_{qo}^{[2]})^2
\]

\[
+ 2\kappa_2^{[1]} \kappa_1^{[1]}(1 + \epsilon_{qo}^{[2]})\kappa_1^{[1]} \kappa_2^{[1]} \right) \right)
\]

In the next proposition, which is a direct generalization of [82, Lemma 13], a probabilistic bound on \( \|\bar{Y} - \bar{Y}\|_F \) is presented. Since this result is formulated in a rather general manner it can be used later on for a variety of different sampling schemes.

Proposition 3.9. Let \( X_1, \ldots, X_N \) be i.i.d. random vectors from \( B_R^{[0]} \) for some \( R > 0 \). Furthermore, assume that for all row-orthonormal matrices \( A \in \mathbb{R}^{m[1] \times m[0]} \) the i.i.d. scalar random variables \( \|AX_i\|_2^2 \) are subgaussian with

\[
\|AX_i\|_2^2 - \mathbb{E}[\|AX_i\|_2^2] \leq C,
\]

where \( C \geq 0 \) is independent of \( A \). Furthermore, define

\[
Y = (\text{vec}(H[f](X_1)) \cdots \text{vec}(H[f](X_N))
\]

\[
\bar{Y} = (\text{vec}(P_MH[f](X_1)) \cdots \text{vec}(P_MH[f](X_N)))
\]

and \( \mu = \mathbb{E}[\|\tilde{W}^{[1]}X_1\|_2^2] \), where \( \tilde{W}^{[1]} \) is the best row-orthonormal matrix for \( W^{[1]} \). We have for any \( t \geq 0 \) that

\[
\|Y - \bar{Y}\|_F \leq C^{[3.7]}(1 + \epsilon_{qo}^{[1]})\sqrt{m[1]}(1 + \epsilon_{qo}^{[2]})\sqrt{(1 + t)N\mu}
\]

with probability at least

\[
1 - 2 \exp \left( - \frac{C^{[8.17]}Nt^2\mu^2}{C^2} \right).
\]

45
Proof. For any $x \in \mathbb{R}^{m[0]}$

\[
\|W^{[1]}x\|_{\infty} \overset{\text{1}}{\leq} \|W^{[1]}x\|_2 \\
\overset{\text{2}}{\leq} (1 + \epsilon_{[0]}) \|\hat{W}^{[1]}x\|_2 \\
\overset{\text{3}}{\leq} (1 + \epsilon_{[0]}) \sqrt{m[1]}\|\hat{W}^{[1]}x\|_{\infty}
\]

where $\|\cdot\|_{\infty} \leq \|\cdot\|_2$ was used for $\text{1}$, Proposition S.11 (i) for $\text{2}$ (with $\hat{W}^{[1]}$ the optimal column-orthonormal matrix for $W^{[1]}$, cf. Proposition S.10), and finally $\|\cdot\|_2 \leq \sqrt{m[1]}\|\cdot\|_{\infty}$ (w.r.t. $\mathbb{R}^{m[1]}$) for $\text{3}$. Combining this with Lemma 3.7 results in

\[
\|Y - \hat{Y}\|_F^2 \leq (C^{[3,7]})(1 + \epsilon_{[0]})^2 m[1] \sum_{k=1}^{N} \|\hat{W}^{[1]}X_k\|_{\infty}^2 = \tilde{C}^2 \sum_{k=1}^{N} \|\hat{W}^{[1]}X_k\|_{\infty}^2, \tag{3.7}
\]

where we defined for brevity $\tilde{C} = C^{[3,7]}(1 + \epsilon_{[0]})\sqrt{m[1]}$. Now

\[
P\left[ \|Y - \hat{Y}\|_F \leq \tilde{C}\sqrt{(1 + t)N\mu} \right]
\]

\[
\geq 1 - P\left[ \|Y - \hat{Y}\|_F \geq \tilde{C}(1 + t)N\mu \right]
\]

\[
\overset{\text{1}}{\geq} 1 - P\left[ \tilde{C}^2 \sum_{k=1}^{N} \|\hat{W}^{[1]}X_k\|_{\infty}^2 \geq \tilde{C}(1 + t)N\mu \right]
\]

\[
= 1 - P\left[ \sum_{k=1}^{N} \|\hat{W}^{[1]}X_k\|_{\infty}^2 \geq (1 + t)N\mu \right]
\]

\[
\overset{\text{2}}{\geq} 1 - \exp\left(-\frac{C^{[S,17]}(tN\mu)^2}{\sum_{k=1}^{N} \|\hat{W}^{[1]}X_k\|_{\infty}^2 - \mathbb{E}[\|\hat{W}^{[1]}X_k\|_{\infty}^2]}\right)
\]

\[
\overset{\text{3}}{\geq} 1 - \exp\left(-\frac{C^{[S,17]}(tN\mu)^2}{NC^2}\right)
\]

\[
= 1 - \exp\left(-\frac{C^{[S,17]}N\mu^2}{C^2}\right)
\]

where we used (3.7) for $\text{1}$, the Hoeffding inequality for subgaussian random variables S.17 for $\text{2}$ (with $C^{[S,17]} > 0$ some absolute constant independent of the concrete subgaussian random variables) and for $\text{3}$ the facts that all $X_k$ are identically distributed and that $\|\hat{W}^{[1]}X_k\|_{\infty}^2 - \mu_{\psi_2} \leq C$ by assumption. \hfill \Box

Remark 3.10. Note that the constant $C^{[3,7]}$ from Lemma 3.7 and used in the preceding proposition depends both on characteristics of the neural network (which are independent of the sampling scheme as long as no adaptive strategy is used) as well as the radius
$R > 0$ of the ball the inputs are taken from (which depends on the sampling scheme). To make this dependence explicit we will sometimes write $C^{[3.7]} = C^{[3.7]}(R)$.

The next task is to lower bound the $m$-th largest singular value. This is done in the following proposition which is a direct adaption of [82, Lemma 14] for $L \geq 2$.

**Proposition 3.11.** Let $f$ be represented by a neural network with $L \geq 2$ layers and each layer has $\epsilon_{q_0}^{[l]}$-nearly orthogonal weights. Furthermore, let $X_1, \ldots, X_N$ be i.i.d. with $X_k \in D^{[0]}$ for $k = 1, \ldots, N$ and define the random matrix

$$Y = \left( \text{vec}(H[f](X_1)) \cdots \text{vec}(H[f](X_N)) \right)$$

Define $m = m^{[1]} + m^{[2]} + \ldots + m^{[L]}$ and suppose that

$$\sigma_m \left( \mathbb{E} \left[ \text{vec}(H[f](X_1))^\otimes 2 \right] \right) \geq \alpha$$

for some $\alpha > 0$. Then we have

$$\sigma_m(Y) \geq \sqrt{(1-t)N\alpha}$$

with probability at least

$$1 - m \exp \left( - \frac{t^2 N \alpha}{2(C^{[3.11]^2})} \right),$$

where

$$C^{[3.11]} = \sum_{l=1}^{L} \left( \prod_{k=1}^{l-1} (1 + \epsilon_{q_0}^{[k]}) \right)^2 \left( \prod_{k=1}^{l-1} \kappa_1^{[k]} \right)^2 \left( 1 + 2\epsilon_{q_0}^{[l]} \right) C^{[3.11]}_l$$

**Proof.** Obviously $\text{vec}(H[f](X_1)), \ldots, \text{vec}(H[f](X_N))$ are i.i.d. random vectors of dimension $m^{[0]}$. Since

$$\sigma_m(Y) = \sqrt{\sigma_m(Y Y^T)}$$

$$= \sqrt{\sigma_m \left( \sum_{k=1}^{N} \text{vec}(H[f](x_n))^\otimes 2 \right)}$$

we are actually in the situation of Theorem S.19 and only an upper bound $\| \text{vec}(H[f](X_1)) \|_2$ is missing.

We have for each $x \in D$

$$\| \text{vec}(H[f](x)) \|_2 = \| H[f](x) \|_F$$

$$= \left\| \sum_{l=1}^{L} V^{[l]T}(x) S^{[l]}(x) V^{[l]}(x) \right\|_F$$

$$\leq \sum_{l=1}^{L} \left\| V^{[l]T}(x) S^{[l]}(x) V^{[l]}(x) \right\|_F$$
using Corollary 1.6 and the triangle inequality for $\| \cdot \|_F$. Now,
\[
\| W^{[1]} T S^{[1]}(x) W^{[1]} \|_F \leq (1 + 2 \epsilon_{q_0}^{[1]}) \| S^{[1]}(x) \|_F \leq (1 + 2 \epsilon_{q_0}^{[1]}) C_1^{[3.1]}
\]

using Proposition S.11 (ii) for the first step and Lemma 3.1 for the second. For $L \geq l \geq 2$ we find
\[
\| V^{[l]} T (x) S^{[l]}(x) V^{[l]}(x) \|_F = \| W^{[1]} T G^{[1]}(x) \ldots G^{[l-1]}(x) W^{[l]} T S^{[l]}(x) W^{[l]} G^{[l-1]}(x) \ldots G^{[1]}(x) W^{[1]} \|_F
\]
\[
\leq \left( \prod_{k=1}^{l-1} (1 + \epsilon_{q_0}^{[k]}) \right)^2 \left( \prod_{k=1}^{l-1} \kappa_1^{[k]} \right)^2 (1 + 2 \epsilon_{q_0}^{[l]}) C_1^{[3.1]}
\]

where we used for (1) submultiplicativity and Proposition S.11 (ii) repeatedly. For (2) we used Proposition S.11 (i) (repeatedly), Lemma 3.1 together with
\[
\| G^{[k]}(x) \|_F = \left\| \left( g^{[k]}_{i_k} \right)^T \left[ z^{[k]}_{i_k}(x) \right], i_k = 1, \ldots, m^{[k]} \right\|_F \leq \kappa_1^{[k]}
\]

Combining the previous two estimates we find
\[
\| \text{vec}(H[f](x)) \|_2 \leq \sum_{l=1}^{L} \| V^{[l]} T (x) S^{[l]}(x) V^{[l]}(x) \|_F
\]
\[
\leq \sum_{l=1}^{L} \left( \prod_{k=1}^{l-1} (1 + \epsilon_{q_0}^{[k]}) \right)^2 \left( \prod_{k=1}^{l-1} \kappa_1^{[k]} \right)^2 (1 + 2 \epsilon_{q_0}^{[l]}) C_1^{[3.1]} = C^{[3.11]}
\]

which immediately leads to an upper bound on the operator norm of $\text{vec}(H[f](x))^{\otimes 2}$,
\[
\| \text{vec}(H[f](x))^{\otimes 2} \|_{2 \rightarrow 2} \leq \| \text{vec}(H[f](x))^{\otimes 2} \|_F
\]
\[
\leq \| \text{vec}(H[f](x))^{\otimes 2} \|_2 \leq C^{[3.11]}^2
\]

Having all necessary ingredients, we find that
\[
P \left[ \sigma_m(Y) \geq \sqrt{(1 - t)N\alpha} \right] = P \left[ \sqrt{\sigma_m \left( \sum_{k=1}^{N} \text{vec}(H[f](x_n))^{\otimes 2} \right) \geq \sqrt{(1 - t)N\alpha} \right]
\]
\[
= P \left[ \sigma_m \left( \sum_{k=1}^{N} \text{vec}(H[f](x_n))^{\otimes 2} \right) \geq (1 - t)N\alpha \right]
\]
\[
\geq 1 - P \left[ \sigma_m \left( \sum_{k=1}^{N} \text{vec}(H[f](x_n))^{\otimes 2} \right) \leq (1 - t)N\alpha \right]
\]
\[
\geq 1 - m \exp \left( - \frac{t^2 N\alpha}{2C^{[3.11]^2}} \right)
\]
Remark 3.12. A simple observation regarding the previous result can be made: The derivation does not make use directly of the magnitude of the input samples \(X_1, \ldots, X_N\) (in contrast to Proposition 3.9) since only properties of the weights (which are independent of the inputs) and activation functions are used (the latter properties obviously have to hold for all inputs). As long as the samples are in the input domain the norm constraints from [82, Lemma 14] are actually not necessary and the preceding proposition can be directly used for a variety of different sampling schemes. Note however that the magnitudes of the inputs do play a role, though implicitly. Allowing larger norms of the inputs means that the input range of the various activation functions within the network gets larger and hence this might influence the constants \(\kappa_k\). This is relevant e.g. when consider unbounded activation functions since then one has to carefully analyse how the input domain (of the network as a whole) is changed (i.e. the input domains of each activation function within the network) when the input is propagated through the neural network.

Example 3.13. For \(L = 2\) we get that
\[
\sigma_{m[0],m[1]}(Y) \geq \sqrt{(1-t)N\alpha}
\]
with probability at least
\[
1 - (m[1] + m[2]) \exp\left(\frac{t^2N\alpha}{2C_2m[2]}\right)
\]
where
\[
C_2 = (1 + \epsilon_{q_0})^4(1 + 2\epsilon_{q_0})^2(\kappa_2^2\kappa_1^2 + \kappa_1^2\kappa_2^2),
\]
which is exactly the result from [82, Lemma 14].

**Probabilistic preparations**

In order to use the previously derived deterministic bound for finding concentration inequalities in addition some probabilistic ingredients are necessary. Here we present only the case of uniform sampling from a sphere.

**Proposition 3.14. Sampling from spheres**

Let \(X \sim \mathcal{U}(S_R^{d-1})\) for \(R > 0\) and \(A \in \mathbb{R}^{n \times d}\) with \(d \geq n > 0\) be row-orthonormal (i.e. \(AA^T = I_n\)). We then have

1. \[
\mathbb{E}[\|AX\|_2^2] \leq \sqrt{\frac{R}{d+2}} \log d + \frac{R^2}{d}
\]
Proof. Let $\tilde{X} \sim \mathcal{U}(S_{1}^{d-1})$, then $X \sim R\tilde{X}$ and hence $E[\|AX\|_{\infty}^{2}] = R^{2}E[\|A\tilde{X}\|_{\infty}^{2}]$. Furthermore,
\[
\|A\tilde{X}\|_{\infty}^{2} = \max_{i=1,\ldots,n}(A_{i}\tilde{X})^{2}
\]
and for all $i = 1, \ldots, n$ $(A_{i}\tilde{X})^{2} \sim Beta(\frac{1}{2}, \frac{d-1}{2})$ from Proposition S.25. Since a $Beta$-distributed random variable is subgaussian we get
\[
\exp \left( \theta ((A_{i}\tilde{X})^{2} - E[(A_{i}\tilde{X})^{2}]) \right) \leq \exp \left( \theta^{2} \sigma^{2} \left( \frac{1}{2}, \frac{d-1}{2} \right) \right)
\]
for all $\theta \in \mathbb{R}$, where we used Proposition S.26 2. in the second step. We can now use Proposition S.16 to arrive at the first claim (recall that $E[(A_{i}\tilde{X})^{2}] = \frac{1}{d}$)
\[
E[\|AX\|_{\infty}^{2}] = R^{2}E[\|A\tilde{X}\|_{\infty}^{2}]
\]
\[
= R^{2}E \left[ \max_{i=1,\ldots,n}(A_{i}\tilde{X})^{2} - \frac{1}{d} \right] + R^{2} \frac{1}{d}
\]
\[
\leq R^{2} \sqrt{\frac{1}{4(d+2)} \log n} + R^{2} \frac{1}{d}
\]
\[
= \sqrt{\frac{R}{d+2} \log n + R^{2} \frac{1}{d}}
\]
For the second claim,
\[
\|AX\|_{\infty}^{2} - E[AX_{\infty}^{2}] \overset{1}{=} R^{2}\|A\tilde{X}\|_{\infty}^{2} - E[\|A\tilde{X}\|_{\infty}^{2}] \overset{2}{=} R^{2} \frac{1}{\sqrt{\log 2}} \|A\tilde{X}\|_{\infty}^{2} - E[\|A\tilde{X}\|_{\infty}^{2}] \overset{3}{=} R^{2} \frac{1}{\sqrt{\log 2}}
\]
where we used that $\|AX\|_{\infty}^{2}$ is subgaussian (since it is bounded by definition) and $\|\cdot\|_{\psi_{2}}$ is a norm (as well as the linearity of expectation) for (1), Proposition S.15 for (2) and finally that $0 \leq \|A\tilde{X}\|_{\infty}^{2} \leq 1$ and therefore $\|A\tilde{X}\|_{\infty}^{2} - E[\|A\tilde{X}\|_{\infty}^{2}] \leq 1$ for (3).

Analysis of the approximation algorithm

We can present now an approximation error bound for sampling from spheres. Note that the following result is a direct generalization of [82, Theorem 4] (going from $L = 2$ to $L \geq 2$).

50
In this section (and Section 3.3) we assume that the weights are \( \epsilon qo \) nearly orthogonal and normed.

**Theorem 3.15. Bound on approximation error**

Consider the situation of Algorithm 3.1 with \( D = U(\mathbb{S}^{m-1}) \). We have then

\[
\|P_M - P_{\hat{M}}\|_F \leq \frac{2C[3.7](1 + \epsilon qo}\sqrt{m} \sqrt{(1 + t) \sqrt{\frac{R}{d+2}} \log d}}{\sqrt{(1 - t)\bar{\alpha}}}
\]

with probability at least

\[
1 - 2 \exp\left(-C[8.17] \frac{N t^2 (\sqrt{\frac{R}{d+2}} \log d + \frac{R^2}{d})^2}{R^4 \log 2}\right) - m \exp\left(\frac{t^2 N \bar{\alpha}}{2 \left(C[3.11]\right)^2}\right)
\]

**Proof.** From Wedin bound we get

\[
\|P_M - P_{\hat{M}}\|_F \leq \frac{2\|Y - \hat{Y}\|_F}{\bar{\alpha}}
\]

with

\[
\bar{\alpha} \geq \min_{i=1,...,m, j=m+1,...,d^2} |\sigma_i - \hat{\sigma}_j|,
\]

\[
\bar{\alpha} \geq \min_{i=1,...,m} |\sigma_i|
\]

By construction

\[
\min_{i=1,...,m, j=m+1,...,d^2} |\sigma_i - \hat{\sigma}_j| = 0
\]

and

\[
\min_{i=1,...,m} |\sigma_i| = \sigma_m,
\]

hence we can use \( \bar{\alpha} = \sigma_m \). Combining now Proposition 3.9 and Proposition 3.14 we find that

\[
\|Y - \hat{Y}\| \leq C[3.7](1 + \epsilon qo)\sqrt{m} \sqrt{(1 + t)N \left(\sqrt{\frac{R}{d+2}} \log d + \frac{R^2}{d}\right)}
\]

with probability at least

\[
1 - 2 \exp\left(-C[8.17] \frac{N t^2 (\sqrt{\frac{R}{d+2}} \log d + \frac{R^2}{d})^2}{R^4 \log 2}\right).
\]

Furthermore, using Proposition 3.11 ensures that

\[
\sigma_m \geq \sqrt{(1 - t)N\bar{\alpha}}
\]
with probability at least
\[ 1 - m \exp \left( \frac{t^2 N \bar{\alpha}}{2 \left( C[3.11]\right)^2} \right). \]

Combining everything using a simple union bound we arrive at
\[ \| P_M - \hat{P}_M \|_F \leq \frac{2C[3.7](1 + \ell_{ps}^{[1]}) \sqrt{m^{[1]}}} {\sqrt{(1 - t) \bar{\alpha}}} \sqrt{(1 + t)N \left( \sqrt{\frac{R}{d+2}} \log d + \frac{R^2}{d} \right)} \]

with probability at least
\[ 1 - 2 \exp \left( - \frac{C[8.17] N t^2 \left( \sqrt{\frac{R}{d+2}} \log d + \frac{R^2}{d} \right)^2}{R^4 \log^2 2} \right) - m \exp \left( \frac{t^2 N \bar{\alpha}}{2 \left( C[3.11]\right)^2} \right) \]

3.3. Approximation by finite differences

As already discussed in the introduction to this chapter, in general the Hessians of the network under consideration are not available. However, similar to Chapter 2 we can use a finite difference approximation of the Hessian,
\[ \Delta^2(x) := \left( \frac{f(x + \epsilon_i e_i + \epsilon_j e_j) - f(x + \epsilon_i e_i) - f(x + \epsilon_j e_j) + f(x)}{\epsilon^2} \right)_{i=1,...,m^{[0]}}, \]
see also [35, Section 3.1] and [82, Chapter 4]. This leads to the following variation of Algorithm 3.1. If all activation functions are twice differentiable with Lipschitz contin-

**Algorithm 3**  Generic auxiliary space approximation scheme with finite differences

**Require:** Number of samples $N$, neural network function $f$, sampling distribution $D$

1: Draw $x_1, \ldots, x_N$ i.i.d. uniformly from $D$
2: Build
\[ \tilde{Y} = \left( vec(\Delta^2[f](x_1)) \ldots vec(\Delta^2[f](x_N)) \right) \]
3: Compute SVD of $\tilde{Y}^T$,
\[ \tilde{Y}^T = (\tilde{U}_1 \tilde{U}_2) \begin{pmatrix} \tilde{\Sigma}_1 & 0 \\ 0 & \tilde{\Sigma}_2 \end{pmatrix} \begin{pmatrix} \tilde{V}_{1}^T \\ \tilde{V}_{2}^T \end{pmatrix} \]

where $\tilde{\Sigma}_1 = diag(\sigma_1, \ldots, \sigma_m)$ contains the $m = m^{[1]} + \ldots + m^{[L]}$ largest singular values and $V_1 = (v_1, \ldots, v_m)$ the corresponding right singular vectors
4: return $\tilde{V}_1$
uous second derivatives (e.g. if all activation functions are three times differentiable), it is clear that for all $\epsilon > 0$ there exists a constant $C_{f,\epsilon}^{[FD_2]}$ (depending only on $f$ and $\epsilon$) such that for all $x \in D$

$$\|H[f](x) - \Delta^2_\epsilon(x)\|_F \leq C_{f,\epsilon}^{[FD_2]} \epsilon.$$ One can derive such a constant explicitly, essentially using a result like Lemma S.31 combined with an argument from the proof of [35, Lemma 3.1]. For example, this calculation has been done explicitly for $L^2$ in [82, Section 4.1]. For simplicity we do not perform the explicit calculation for arbitrary $L$, instead we assume the constant $C_{f,\epsilon}^{[FD_2]}$ as given, which enables us to adapt the results from the previous section to the more realistic setting of using finite difference approximations instead of known exact Hessians. This is essentially a direct generalization of [82, Theorem 5].

**Theorem 3.16.** In the situation of Algorithm 3.3 assume that

$$\sigma_m(\mathbb{E}[\text{vec}(H[f](X))^{\otimes_2}]) \geq \tilde{\alpha},$$

where $X$ is sampled according to $\mathcal{D}$. Furthermore, let $s_0 > 0$ such that

$$\sqrt{(1-s)\tilde{\alpha}} > C_{f,\epsilon}^{[FD_2]} \epsilon.$$ Then we have for all $t > 0, s \in (0, s_0)$ that

$$\|P_M - P_M\|_F \leq \frac{2C_{f,\epsilon}^{[FD_2]} + 2C^{[3.7]}(1 + \epsilon^{[1]}_q)\sqrt{m}N \left(\frac{R}{d+2} \log d + \frac{R^2}{d}\right)}{(1-t)\tilde{\alpha} - C_{f,\epsilon}^{[FD_2]} \epsilon}$$

with probability at least

$$1 - 2\exp\left(-\frac{C^{[5.17]}N t^2 \left(\frac{R}{d+2} \log d + \frac{R^2}{d}\right)}{m^2} - m \exp\left(-\frac{t^2 N \tilde{\alpha}}{2 \left(C^{[3.11]}\right)^2}\right)\right).$$

**Proof.** Similar to Theorem 3.15 Wedin’s bound will be used here. We start with the numerator in this result: We have (using the triangle inequality and the definition of $\|\cdot\|_F$)

$$\|\hat{Y} - \hat{\bar{Y}}\|_F \leq \|\hat{Y} - Y\|_F + \|Y - \hat{\bar{Y}}\|_F$$

$$= \sqrt{\sum_{k=1}^{N} \|H[f](x_k) - \Delta^2_\epsilon[f](x_k)\|^2_F} + \|Y - \hat{\bar{Y}}\|_F$$

$$\leq \sqrt{NC_{f,\epsilon}^{[FD_2]}} \epsilon + \|Y - \hat{\bar{Y}}\|_F$$

$$\leq \sqrt{NC_{f,\epsilon}^{[FD_2]}} \epsilon + C^{[3.7]}(1 + \epsilon^{[1]}_q)\sqrt{m}N \left(\frac{R}{d+2} \log d + \frac{R^2}{d}\right)$$

$$\leq \sqrt{NC_{f,\epsilon}^{[FD_2]}} \epsilon + C^{[3.7]}(1 + \epsilon^{[1]}_q)\sqrt{m}N \left(\frac{R}{d+2} \log d + \frac{R^2}{d}\right).$$
with probability at least

\[
1 - 2 \exp \left( - \frac{C^{[S,17]} N t^2 \left( \frac{R}{d+2} \log d + \frac{R^2}{d} \right)^2}{\frac{R^4}{\log^2 2}} \right)
\]

where we used Proposition 3.9 in the last inequality. Next, we deal with the denominator in the right hand side of Wedin’s bound. Obviously

\[
\sigma_m(\tilde{Y}) \geq \sigma_m(Y) - |\sigma_m(\tilde{Y}) - \sigma_m(Y)|,
\]

and

\[
\sigma_m(Y) \geq \sqrt{(1 - s)N \hat{\alpha}}
\]

with probability at least

\[
1 - m \exp \left( - \frac{s^2 N \hat{\alpha}}{2(C^{[3,11]}))^2} \right)
\]

by Proposition 3.11. Furthermore, using Weyl’s inequality

\[
|\sigma_m(\tilde{Y}) - \sigma_m(Y)| \leq \|Y - \hat{Y}\|_F \leq \sqrt{NC}_{f,\epsilon} \epsilon,
\]

so altogether

\[
\sigma_m(\tilde{Y}) \geq \sqrt{(1 - t)N \hat{\alpha}} - \sqrt{NC}_{f,\epsilon} \epsilon.
\]

Since by construction the range of \( \tilde{Y} \) has always dimension less or equal to \( m \) we finally arrive at

\[
\|P_M - \hat{P}_M\|_F \leq \frac{2\|\hat{Y} - \tilde{Y}\|_F}{\sigma_m(\tilde{Y})}
\]

\[
\leq \frac{2\sqrt{NC}_{f,\epsilon} \epsilon + 2C^{[3,7]}(1 + \epsilon_q[1])\sqrt{m[1]}(1 + t)N \left( \sqrt{\frac{R}{d+2} \log d + \frac{R^2}{d}} \right)}{\sqrt{(1 - t)N \hat{\alpha}} - \sqrt{NC}_{f,\epsilon} \epsilon}
\]

\[
= \frac{2C^{[FD,\epsilon]} \epsilon + 2C^{[3,7]}(1 + \epsilon_q[1])\sqrt{m[1]}\sqrt{(1 + t)N \left( \sqrt{\frac{R}{d+2} \log d + \frac{R^2}{d}} \right)}}{\sqrt{(1 - t)\hat{\alpha}} - C^{[FD,\epsilon]} \epsilon}
\]

with probability at least

\[
1 - 2 \exp \left( - \frac{C^{[S,17]} N t^2 \left( \frac{R}{d+2} \log d + \frac{R^2}{d} \right)^2}{\frac{R^4}{\log^2 2}} \right) - m \exp \left( - \frac{t^2 N \hat{\alpha}}{2(C^{[3,11]}))^2} \right)
\]
3.4. Alternative approach

Unfortunately the originally proposed procedure (using a direct generalization of the auxiliary space approximation from [82] together with Algorithm [35, Algorithm 3.4]) is not entirely satisfying. First, the approximation of the auxiliary space (measured by $\|P_M - P_{\tilde{M}}\|_F$) seems to be rather imprecise despite a huge number of samples. This was confirmed by inspecting the approximation error in the numerical experiments in Chapter 4. Second, spurious vectors occurred, i.e. vectors that are neither weights nor mixed weights, cf. in particular Section 4.4. Third, it is not clear how to distinguish weights and mixed weights as already observed in [82] for two hidden layers. For this case two methods were proposed in [82] (one for general activation functions, one for sigmoidal functions), cf. also [32, Section 6.1]. Originally we proposed an alternative method for distinguishing weights and mixed weights (Iteratively Projections and Resampling of Hessians) that enjoys empirical success (very few to no failures, i.e. wrong classification of vectors), however, its performance when used with spurious vectors is unclear and it seems to require much more additional samples than e.g. the saturation method in [32, Section 6.1].

Due to these problems we present a new method here for which we cannot give theoretical guarantees so far. However, experiments indicate good empirical success.

We start with some important preliminary observations that will motivate the alternative method.

Preliminary observations

First, we inspect the approximated space that result from methods like Algorithm 3.1. For this purpose we consider the angles between the one-dimensional spaces $\text{span}\{v_{i_1}^{[l]} \otimes v_{l_1}^{[l]}\}$ spanned by the tensor products of the true weights (respective mixed) weights $v_{i_1}^{[l]}$, for $i_1 = 1, \ldots, m^{[l]}$, $l = 1, \ldots, L$. Recall that $v_{i_1}^{[l]} = a_{i_1}^{[l]}$ for $i_1 = 1, \ldots, m^{[l]}$. Let $\tilde{\theta}_{i_1}^{[l]}$ denote the angle between the approximated auxiliary space and the space $\text{span}\{v_{i_1}^{[l]} \otimes v_{l_1}^{[l]}\}$ and let $P_{\tilde{M}}$ be the corresponding orthogonal projection onto the former of these spaces. Elementary considerations show that

$$\cos \tilde{\theta}_{i_1}^{[l]} = \frac{P_{\tilde{M}} v_{i_1}^{[l]} \otimes v_{l_1}^{[l]}}{\|v_{i_1}^{[l]} \otimes v_{l_1}^{[l]}\|_F}$$

which can be easily computed. Figure 3.4 illustrates the situation. Algorithm 3.1 allows for some flexibility in the sampling distribution $D$ and here we focus on simple distributions with mean 0 which exhibit certain symmetries, like sampling uniformly from a sphere of a given radius or sampling uniformly from a hypercube with given sidelengths, compare also with the discussions at the beginning of this chapter.

If we inspect the angles $\theta_{i_1}$ for a particular network one finds that some sampling schemes lead to results such as the one in Figure 3.4, namely a marked separation between the angles corresponding to weights $a_{i_1}^{[l]}$ and mixed weights $v_{i_1}^{[l]}$, $l \geq 2$. More
Figure 3.1.: Illustration of the angle between the one dimensional subspaces (in matrix space) span\{v_i^{[l]} \otimes v_i^{[l]}\} and the approximated auxiliary space.

precisely, the angles belonging to weights are smaller than the angles corresponding to mixed weights. In Figure 3.4 we demonstrate this for a network of architecture [20, 20, 10] (two hidden layers) and [20, 20, 10, 5] (three hidden layers) and using uniform sampling from the sphere of a certain radius and uniform sampling from the hypercube of a certain sidelength. Heuristically, this can be interpreted as the approximated auxiliary space containing a decent approximation of the matrix space spanned by the tensor products of the first layer weights. For convinence we define

$$\mathcal{A} = \text{span}\{a_1^{[1]} \otimes a_1^{[1]}, \ldots, a_m^{[1]} \otimes a_m^{[1]}\}.$$ 

and call this matrix subspace alternative auxiliary space. The above intuition can be then formulated as follows: Chosing an appropriate sampling distribution in Algorithm 3.3 leads to an approximated auxiliary space \(\tilde{\mathcal{M}}\) that contains a decent approximation \(\tilde{\mathcal{A}}\) to the alternative auxiliary space \(\mathcal{A}\). Note that for the particular case displayed in Figure 3.4 this resulted from sampling uniformly from the hypercube with moderately large sidelength and sampling uniformly from the sphere with moderately large radius, respectively. A similar behaviour was found in many of our experiments.

As a sidenote, in many numerical experiments one can find also a significant gap in the singular value distribution of the SVD used for Algorithm 3.1 around the \(m^{[1]}\)-th singular value for some sampling schemes, in particular sampling from the hypercube with independent coordinates, cf. Figure 3.3. Note that this phenomenon seems to be not a saturation effect since it also occurs when using e.g. sine as an activation function.
Figure 3.2.: Acute angles $\tilde{\theta}_{ri}$ (ordered as $\tilde{\theta}_{r1s1}, \ldots, \tilde{\theta}_{r2s1}, \ldots, \tilde{\theta}_{ri, \ldots}$) arising from using Algorithm 3.3 with sampling uniformly from unit sphere (upper row) and hypercube (lower row) for a network with two (left) and three (right) hidden layers.

Figure 3.3.: Singular values for the matrix $\tilde{Y}^T$ from Algorithm 3.3 (log scale) for a two hidden layer neural network. On the left $tanh$ has been used as an activation function, on the right $sine$. 

57
Figure 3.4.: Acute angles $\tilde{\theta}_{ij}^{[l]}$ between the approximate auxiliary space $\tilde{\mathcal{M}}$ from Algorithm 3.3 using uniform sampling from the hypercube with different sidelengths for networks with two hidden layers. The angles are sorted as in Figure 3.4. Shown are the average angles, i.e. we generated 20 random networks with two hidden layers, applied Algorithm 3.3 to get an approximation $\tilde{\mathcal{M}}$, calculated the acute angles $\tilde{\theta}_{ij}^{[l]}$ and averaged all angles (for the 20 networks) for each (mixed) weight.

Second, when changing the sampling distribution used in Algorithm 3.1 one can find an interesting change in the angles $\tilde{\theta}_{ij}^{[l]}$. To be precise, fix a parametrized family of distributions on $\mathbb{R}^{m_{[0]}}$ with mean zero and some symmetry (e.g. radially symmetric) and run Algorithm 3.3 for different parameters. For example, we tested the uniform distribution on the hypercube with the sidelength of the hypercube as a parameter and the uniform distribution on the sphere with the radius of the sphere as a parameter. We generated several networks of a certain architecture, run Algorithm 3.1 for each parameter setting and then averaged the angle values over all networks. Figure 3.4 shows selected results for two hidden layers and using the uniform distribution on the hypercube.

Note that we tested much more sampling parameters in between, but for brevity we display only four selected values. One can see clearly that the separation of the angles corresponding to weights and mixed weights depends on the sampling parameter. In particular, with increasing parameter (here sidelength) the angles corresponding to
weights get slightly better, the angles corresponding to mixed weights get significantly worse. It can be interpreted in the following way: The approximated auxiliary spaces resulting from different sampling parameters contain decent approximations to $\mathcal{A}$ for a large range of sampling parameter. At the same time they might contain a decent or a rather bad approximation to the spaces spanned by mixed weights.

This behaviour can be also observed for three and four hidden layers as well as sampling uniformly from the sphere for different radii.

Note that in rare case we found a different behaviour: For small parameter values (i.e. small hypercube or sphere) the angles corresponding to a subset of the mixed weights are rather small and all other angles (including those of the weights) are rather large. Increasing the sampling parameter then leads to a situation like the one just described. In the following developments we ignore these special cases at least for now.

**An alternative two-stage approximation method**

The preliminary observations lead us to the following idea: It seems that using Algorithm 3.3 with different sampling parameters (e.g. uniform sampling from the hypercube of different side lengths) we can systematically generate different subspace approximations that all contain a good approximation to $\mathcal{A}$ and most contain no good approximation of the one dimensional subspaces corresponding to mixed weights. Extracting from these different subspaces the “most stable” $m^{[b]}$-dimensional subspace should then lead to a good approximation of $\mathcal{A}$.

Based on the considerations above we arrive at the following two stage method to approximate $\mathcal{A}$: First we approximate a range of different $N_2$-dimensional subspaces $\mathcal{A}_1, \ldots, \mathcal{A}_{N_1}$ using Algorithm 3.3 with different sampling schemes $D_1, \ldots, D_{N_1}$. Second, we find the “mean subspace” of $\mathcal{A}_1, \ldots, \mathcal{A}_{N_1}$ which should be close to $\mathcal{A}$. The idea is illustrated in Figure 3.4 and made precise in Algorithm 3.4.

**Numerical experiments**

We now test the heuristically derived method from above. First, we look at a simple scenario: We consider neural networks with 2, 3, 4 layers and having widths $[20, 20, 10]$, $[20, 20, 10, 10]$ and $[20, 20, 10, 10, 5]$, the weights in each layer are nearly orthogonal with $\epsilon_{no} = 0.5$ and we consider activation functions of the form $g(s) = \tanh(c \cdot (s + b))$ with $c = 1$. We sample in each setting 40 networks and apply Algorithm 3.4 sampling from the hypercube with sidelengths $0.5, 1, \ldots, 20$ uniformly 100 times for each sidelength. In all cases the approximation error of the auxiliary space (i.e. $\|P_{\tilde{A}} - P_{\mathcal{A}}\|_F$) has been less than 0.2. Repeating this experiment with uniform sampling from the sphere did not work as well as sampling from the hypercube. The reason seems to be a more sensitive dependence on the sampling parameter. Changing the latter (i.e. the radius of the sphere) to $15, 15.5, \ldots, 25$ lead to decent results for sampling from the sphere (75% quantiles of approximation error of 0.27, 0.42, 0.34 and maximum error of 0.4, 0.98, 0.93 for two, three and four hidden layers, respectively).
Algorithm 4 Two stage alternative auxiliary space approximation

1: procedure APPROXMATRIXWEIGHTSPACE(f, $D_1, \ldots, D_{N_1}, N_2$)
2:   for $i = 1, \ldots, N_1$ do
3:     Sample $x_1, \ldots, x_{N_2}$ i.i.d. according to $D_i$
4:     Build
5:       $\hat{Y} = (\text{vec}(\Delta_2^2[f](x_1)) \ldots \text{vec}(\Delta_2^2[f](x_{N_2})))$
6:     Compute SVD
7:       $\hat{Y}^T = \begin{pmatrix} \hat{U}_1 & \hat{U}_2 \end{pmatrix} \begin{pmatrix} \hat{\Sigma}_1 & 0 \\ 0 & \hat{\Sigma}_2 \end{pmatrix} \begin{pmatrix} \hat{V}_1^T \\ \hat{V}_2^T \end{pmatrix}$
8:   end for
9:   for $i = 1, \ldots, N_1$ do
10:     Get ONB $B = (b_1 \ldots b_{N_2})$ for space $\mathcal{A}_i$,
11:     $Z \leftarrow (Z \ B)$
12:   end for
13: Compute SVD of $Z$
14: $Z = \begin{pmatrix} U_1 & U_2 \end{pmatrix} \begin{pmatrix} \Sigma_1 & 0 \\ 0 & \Sigma_2 \end{pmatrix} \begin{pmatrix} V_1^T \\ V_2^T \end{pmatrix}$
15:     where $\Sigma_1 = \text{diag}(\sigma_1, \ldots, \sigma_{N_2})$ contains the $N_2$ largest singular values and $V_1 = (v_1, \ldots, v_{N_2})$ the corresponding right singular vectors
16: return $\text{im}(V_1)$
17: end procedure

60
Next, we repeat this experiment (using sampling from hypercube again), but this time the scalings of the activation functions are chosen randomly uniformly from $[0.1, 3]$. Interesting, the performance deteriorates a bit: In 90% of the test cases (for 2, 3, 4 layers each) the approximation error of the auxiliary space is less than 0.3. However, in ca 10% of the test cases the approximation error has gone up dramatically very close to 1. It seems that the alternative approximation approach is sensitive to the scalings, at least to a certain extent.

Let us repeat the first experiment, but this time with widths $[20, 20, 30], [20, 20, 30, 5]$ and $[20, 20, 30, 30, 5]$. Obviously, for the second, second, second and third hidden layer, respectively, we cannot require near-orthogonality anymore. Not surprisingly, we find again excellent approximation performance with an approximation error of less than 0.2 in all cases. Note that this indicates that the alternative approximation approach seems to be suitable for general architectures and not only pyramidal shaped architectures (at least as long as the wider hidden layers are not much wider than the input and first hidden layer).
4. Recovering weights from the auxiliary matrix space

In the last chapter we saw that one can approximate a certain auxiliary matrix space via a non-adaptiv sampling strategy. This auxiliary space is spanned (approximately) from dyadic products of weights from the first layer as well as dyadic products of linear combinations of weights from the first layer. If the network has only one hidden layer the underlying weights can be extracted from this space using a simple non-linear program [34]. One can use the approach for one hidden layer also for the auxiliary matrix space for two hidden layers. The recovery guarantees from [34] do not apply directly anymore, however, empirical experience showed that this strategy can still be successful.

In this chapter we continue the work from [82] and use the original extraction methodology for the auxiliary matrix space resulting from two or more weights. We won’t be able to give recovery guarantees, so instead we perform numerical experiments to investigate the success of this approach.

4.1. Introduction and overview

Given an approximation \( \tilde{\mathcal{M}} \) to the auxiliary matrix space \( \mathcal{M} \) from last chapter, consider the following nonlinear program.

\[
\max_{\tilde{\mathcal{M}}, \mathcal{M}} \frac{\|M\|_{2\to2}}{\|M\|_{F\leq1}}.
\] (4.1)

This optimization problem is non-convex, but it was shown in [34] that if in the case of one hidden layer \( \tilde{\mathcal{M}} \) is close enough to \( \mathcal{M} \) and the weights are nearly orthogonal\(^1\) then all maximizers of (4.1) are nearly dyadic products of weights, i.e. the first singular direction of the maximizers will be close to a weight direction.

Consider now the situation for two or more layers. The space \( \mathcal{M} \) is spanned by matrices of the form \( a_{i_1}^{[1]} \otimes a_{i_1}^{[1]} \) for \( i_1 = 1, \ldots, n^{[1]} \) as well as \( (W^{[1]}\beta)^T \otimes (W^{[1]}\beta)^T \) for certain \( \beta \in \mathbb{R}^{n^{[1]}} \). Note that the \( \beta \) are unknown, but their form is known, cf. Section 1.4. Assume now that all the weights from the first layer as well as the mixed weights \( W^{[1]}\beta \) have unit norm. Obviously the weights and mixed weights cannot be orthogonal anymore. For example, if the weights from the first layer are orthogonal or nearly orthogonal and \( \beta \) is compressible (i.e. having only a few significantly non-zero coordinates), then the weights and mixed weights will be far from orthogonality.

\(^1\)Note that this can be relaxed using whiting [35]
Nevertheless, one can assume that in a generic situation (e.g. weights from first layer are nearly orthogonal and the mixing weights $\beta$ are not compressible) a certain degree of near-orthogonality between weights from first layer and mixed weights are retained. To illustrate this discussion the Gramian matrix (i.e. the pairwise scalar products) for all normalized $v_{il}^{[l]}$ is displayed in Figure 4.1 for a two hidden layer network of widths $[30, 15, 7, 3]$. In the generic situation just described it is reasonable that the extraction method outlined above might still work. Indeed, [82] followed this strategy for networks with two hidden layers with some success, retrieving the weights and mixed weights with some empirical success. We will follow suit and apply exactly this strategy for two or more layers, evaluating its success empirically.

4.2. Weight recovery using an approximate projected gradient algorithm

Of course the question remains how to find maximizers of (4.1). A simple algorithm was proposed and a basic convergence analysis provided in [34]. This algorithm is recalled here as Algorithm 4.2. Note that [32] proposed a slightly modified version and complemented it with a detailed analysis.

To simplify the presentation we introduce the following notation. Given a matrix $X \in \mathbb{R}^d$, let $X = U\Sigma V^T$ be its SVD and let $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_d$ be its ordered singular values. Given $\gamma > 0$ define now

$$\Pi_\gamma(X) = \frac{1}{\sqrt{\gamma^2\sigma_1^2 + \sigma_2^2 + \ldots + \sigma_d^2}} U \begin{pmatrix} \gamma \sigma_1 & 0 & \cdots & 0 \\ 0 & \sigma_2 & 0 & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & \cdots & \sigma_d \end{pmatrix} V^T$$

Note that computing $\Pi_\gamma$ requires performing a full SVD of its input argument.

<table>
<thead>
<tr>
<th>Algorithm 5 Iterative operator norm maximization</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>procedure</strong> RunMaximization($P_M$, $X_0$, $\gamma$, StopCriterion)</td>
</tr>
<tr>
<td>for $i = 0, 1, 2 \ldots$ do</td>
</tr>
<tr>
<td>$X_{i+1} = P_M \Pi_{\gamma}(X_i)$</td>
</tr>
<tr>
<td>if StopCriterion then</td>
</tr>
<tr>
<td>return $X_{i+1}$</td>
</tr>
<tr>
<td>end if</td>
</tr>
<tr>
<td>end for</td>
</tr>
</tbody>
</table>

The convergence analysis in [34] relies on near-orthogonality of the underlying weights as well as a good initial value. Again, this does not immediately apply to the situation at hand, however, empirical evidence in [34] and [82] suggests that this algorithm works.
Figure 4.1.: Illustration of loosing near-orthogonality for more than one hidden layer. Shown are all pairwise scalar products between normalized weights and mixed weights $\frac{v_r^{[l]}}{|v_r^{[l]}|_2}$, $l = 1, 2, 3$, for $m = [30, 15, 7, 3]$ and the weights in each layer are 0.1 near-orthogonal. The values of the scalar products are encoded with a graymap, white corresponds to 1, black to 0. The weights are ordered as $a_1^{[1]}, \ldots, a_{15}^{[1]}, v_1^{[2]}, \ldots, v_7^{[2]}, v_1^{[3]}, \ldots, v_3^{[3]}$ and numbering starts from 0.
very well for rather non-orthogonal weights as well as more than one layer. We will therefore also use this algorithm in the following.

**Excursion: Derivation of Algorithm 4.2 as an approximated projected subgradient method**

For convenience we present now a detailed derivation of Algorithm 4.2 as an approximate projected subgradient method. For simplicity we work in this section with general subspaces $\mathcal{M} \subseteq \mathbb{R}^{d \times d}$. Defining $\mathcal{B} =: B_{\mathbb{R}^{d \times d}}(0)$ the goal is to solve the nonlinear program

$$\max_{M \in \mathcal{B} \cap \mathcal{M}} \|M\|_{2\rightarrow 2},$$

i.e. we want to find a maximzer of the function $M \mapsto \|M\|_{2\rightarrow 2}$, which is in general not differentiable, on the constraint set $\mathcal{B} \cap \mathcal{M}$.

First the notion of subgradient is recalled.

**Definition 4.1. Subgradient**

Let $X$ be a real (pre-)Hilbert space, $D \subseteq X$ a convex set and $f : D \to \mathbb{R}$ a convex function. We define the *subdifferential of* $f$ *at* $x \in D$ by

$$\partial f(x) = \{ u \in X \mid \forall y \in X : f(x) + \langle x - y, y \rangle \leq f(y) \}.$$  

Each element of the subdifferential is called a *subgradient*.

Note that the subdifferential is a set which can be a singleton. For example, it is well-known in the case of a function on $\mathbb{R}^n$ (which is a Hilbert space) if a function is differentiable at a point $x$ in the usual sense, then the subdifferential of $f$ at $x$ is singleton containing the usual gradient (of $f$ at $x$).

We now give the subgradient step of the approximate projected subgradient method. Choosing the fixed step-size strategy (cf. [21, Section 1]) with step size $\gamma > 0$ the subgradient step is of the form

$$M^{(n+1)} = M^{(n)} + \gamma G^{(n)}$$

where $G^{(n)} \in \partial \|M^{(n)}\|_{2\rightarrow 2}$ is an arbitrary subgradient of $\|\cdot\|_{2\rightarrow 2}$ at the point $M^{(n)}$.

Therefore we need an element from the subdifferential of the function $M \mapsto \|M\|_{2\rightarrow 2}$, preferably one that is easy to calculate. The next result presents a simple choice.

**Lemma 4.2. Element from subdifferential**

Let $A \in \mathbb{R}^{d \times d}$ be arbitrary and consider its SVD $A = U\Sigma V^T$. Then

$$U_{:,1}V_{1,1} = U\text{diag}(1,0,\ldots,0)V^T \in \partial \|A\|_{2\rightarrow 2}.$$  

**Proof.** The claim follows immediately from [98, Example 3] (based on [98, Theorem 4]).
Using the subgradient from Lemma 4.2 we arrive at the subgradient step
\[ M^{(n+1)} = M^{(n)} + \gamma U^{(n)} \text{diag}(\sigma_1^{(n)} + \gamma, \sigma_2^{(n)}, \ldots, \sigma_d^{(n)}) V^{(n)T}, \]
where \( M^{(n)} = U^{(n)} \Sigma^{(n)} V^{(n)T} \) is the usual SVD and \( \sigma_i^{(n)}, i = 1, \ldots, d \) are the ordered singular values of \( M^{(n)} \), and we start at some \( M^{(0)} \in \mathbb{R}^{d \times d}. \)

Since the constraint set is \( \mathcal{B} \cap \mathcal{M} \) the projected gradient step is then
\[ M^{(n+1)} = P_{\mathcal{B} \cap \mathcal{M}} \left( M^{(n)} + \gamma U^{(n)} \text{diag}(1, 0, \ldots, 0) V^{(n)T} \right). \]

Now, the projection \( P_{\mathcal{B} \cap \mathcal{M}} \) is potentially complicated, so we use an approximate projection \( P \) (cf. [21, Section 3.1]) and arrive at the approximate projected subgradient step
\[ M^{(n+1)} = P_{\mathcal{M}} \left( P_{\mathcal{B}} \left( M^{(n)} + \gamma U^{(n)} \text{diag}(1, 0, \ldots, 0) V^{(n)T} \right) \right). \]

Obviously \( \mathcal{B} \cap \mathcal{M} \) is convex (as the intersection of a convex set \( \mathcal{B} \) and a subspace \( \mathcal{M} \)) and \( \mathcal{B} \cap \mathcal{M} \subseteq \mathcal{B} \). Therefore, we can use the approximate projection \( P = P_\mathcal{M} \circ P_\mathcal{B} \) as a substitute for \( P_{\mathcal{B} \cap \mathcal{M}} \).

It is clear that the projection of \( \mathbb{R}^{d \times d} \) onto \( \mathcal{B} \) (w.r.t. to the \( \| \cdot \|_F \)-norm) is given by
\[
P_{\mathcal{B}}(M) = \begin{cases} \frac{M}{\| M \|_F} & \| M \|_F \geq 1 \\ M & \text{otherwise} \end{cases}
\]
If \( \gamma \geq 1 \) then obviously \( \| U^{(n)} \text{diag}(\sigma_1^{(n)} + \gamma, \sigma_2^{(n)}, \ldots, \sigma_d^{(n)}) V^{(n)T} \|_F \geq \gamma \geq 1 \) and therefore using \( P_{\mathcal{B}} \) in the subgradient step leads to
\[
P_{\mathcal{B}}(M^{(n+1)}) = P_{\mathcal{B}} \left( U^{(n)} \text{diag}(\sigma_1^{(n)} + \gamma, \sigma_2^{(n)}, \ldots, \sigma_d^{(n)}) V^{(n)T} \right)
= \frac{1}{\sqrt{(\sigma_1^{(n)} + \gamma)^2 + \sigma_2^{(n)} + \ldots + \sigma_d^{(n)}}} U^{(n)} \text{diag}(\sigma_1^{(n)} + \gamma, \sigma_2^{(n)}, \ldots, \sigma_d^{(n)}) V^{(n)T},
\]
i.e. for \( M^{(n)} \) we have \( P_{\mathcal{B}}(M^{(n+1)}) = \Pi_\gamma(M^{(n)}) \) in the notation of Algorithm 4.2. Using the projection \( P_\mathcal{M} \) finally results in the complete update step
\[ M^{(n+1)} = P_{\mathcal{M}} \Pi_\gamma(M^{(n)}) \]
and we realize that we have arrived at Algorithm 4.2.

### 4.3. Weight recovery using gradient ascent on the unit sphere

The approximate projected subgradient algorithm presented in the last section might pose some problems.

- In each iteration of each restart a full SVD has to be performed. Although fast algorithms are available this can be still an expensive operation in high dimensions.
• In order to recover all weights (assuming that the auxiliary matrix space $\mathcal{M}$ or an approximation thereof is sufficiently well-behaved) one might need many restarts, for a detailed discussion cf. [35, Section 5.2].

Here we present an alternative strategy to recover weights from the auxiliary space $\mathcal{M}$. To explain the idea and derive the algorithm we return to the setting of [35], i.e. we assume

$$\mathcal{M} = \text{span}\{a_1 \otimes a_1, \ldots, a_n \otimes a_n\}$$

for nearly orthonormal vectors $a_i \in \mathbb{R}^d$ (or at least we have an approximation $\tilde{\mathcal{M}}$).

First, we explain the intuition behind our approach, in particular our choice to perform an optimization in weight space instead of matrix space. The next subsection is concerned with a formal derivation of a gradient based algorithm (Algorithm 4.3) that allows to retrieve (an approximation of) one weight from the auxiliary matrix space given an initial vector. Note that this algorithm does not require an SVD in its inner iteration. Finally, we present a simple adaptive initialization strategy (Algorithm 4.3) that could drastically reduce the number of runs (i.e. restarts of the respective optimization algorithm).

**Introduction and motivation**

In order to find one weight (or rather an approximation thereof) Algorithm 4.2 used in [35] and [82] starts with a random matrix contained in the intersection of the unit sphere in matrix space $\mathbb{R}^{d \times d}$ (w.r.t. $\| \cdot \|_F$-norm) and the auxiliary space $\mathcal{M}$ (or an approximation thereof), and while staying in this intersection it moves iteratively to get towards an approximately rank one matrix (and according to [35, Section 3.4] the first singular vector is guaranteed to be close to a weight).

Our proposed alternative works differently: We start with a random vector in weight space $\mathbb{R}^d$ and we move this vector iteratively to get closer to a weight. The key observation is that if a vector $u$ in weight space stays on the unit sphere (in weight space), then $u \otimes u$ stays on the unit sphere in matrix space (w.r.t. $\| \cdot \|_F$-norm). If we now move the vector on the unit sphere in weight space such that the corresponding tensor product is close to the intersection of the unit sphere in matrix space with the auxiliary space, then the underlying vector should be close to a weight (up to sign of course). This is the case since its tensor product is certainly a rank one matrix and as such it is a maximizer of the $\| \cdot \|_{2 \rightarrow 2}$-norm on the unit sphere (w.r.t. $\| \cdot \|_F$-norm). If it is also very close to the auxiliary space (or a good approximation), then it is close to a local maximizer of [35, (3.23)] and according to [35, Theorem 3.12] it should be close to a weight (up to sign). Note that of course for a rigorous usage of of the latter result one would have to perform an additional stability analysis since the vector might be only close to the approximate space. Figure 4.2 explains the basic setting. Starting with a random vector on the unit sphere in weight space we now want to move on the unit sphere in weight space such that the corresponding tensor product (on the unit sphere in matrix space) gets closer to the auxiliary space $\mathcal{M}$. In other words we want to minimize the distance of the tensor product to the auxiliary space which is the same as maximizing the orthogonal
projection of the tensor product onto the auxiliary space. This leads to the following simple optimization problem

$$\max_{u \in S^{d-1}} \| P_M(u \otimes u) \|_F$$

(4.2)

Note that problem (4.2) is an optimization problem on a manifold (the unit sphere in weight space) and powerful algorithms exist for this situation. In the next subsection we will derive a gradient-based algorithm for finding local maximizers of (4.2), cf. Algorithm 4.3. Note that in each iteration of this algorithm only matrix-vector and matrix-matrix-multiplications are used and no SVD is required. Of course, in order to recover all weights from the matrix space several restarts are necessary. However, since we work now in weight space an efficient initialization is possible as is explained later (cf. Algorithm 4.3).

Derivation of the gradient ascent algorithm

We now derive a simple gradient ascent algorithm on the sphere to find local maximizers of (4.2). To simplify calculations we use the equivalent problem

$$\max_{u \in S^{d-1}} \| P_M(u \otimes u) \|_F^2$$

(4.3)

The objective function is

$$f : S^{d-1} \rightarrow \mathbb{R}, \ u \mapsto \| P_M(u \otimes u) \|_F^2$$

and we need its gradient. This is quite a standard task and so we simply follow [1, Example 3.6.1] and [1, Section 4.6.1]. First, we need the tangent and normal space at a point $u$ on the sphere (interpreted as a Riemannian manifold with the standard
Riemannian metric, cf. e.g. [1, Section 3.6]). The tangent space $T_u S^d$ can be identified with $\mathbb{R}^{d-1}$ and the normal space is just

$$(T_u S^d)^\perp = \{ \alpha u \mid \alpha \in \mathbb{R} \}.$$  

The (orthogonal) projections onto the tangent space $T_u S^d$ and onto the normal space $(T_u S^d)^\perp$ are simply given by

$$P_u \xi =: P_{T_u S^d} \xi = (id_{\mathbb{R}^d} - uu^T) \xi, \quad P_u^\perp \xi =: P_{(T_u S^d)^\perp} \xi = uu^T \xi$$

for $\xi \in \mathbb{R}^d$, respectively.

Since the unit sphere in $\mathbb{R}^d$ is an embedded manifold having the gradient $\nabla \bar{f}$ of the unconstrained objective function $\bar{f} : \mathbb{R}^d \to \mathbb{R}, \ u \mapsto \|P_M(u \otimes u)\|_F^2$, the gradient of the actual objective function $f$ (whose domain is the unit sphere) is simply $\nabla f(u) = P_u(\nabla \bar{f}(u))$, cf. [1, (3.37) on page 48]. Therefore, let’s calculate $\nabla \bar{f}$, the gradient of the unconstrained objective function. We have

$$\bar{f}(x) = (\phi_1 \circ \phi_2 \circ \phi_3)(x)$$

with

$$\phi_1 : \mathbb{R}^{d\times d} \to \mathbb{R}, \ X \mapsto \|X\|_F^2,$$
$$\phi_2 : \mathbb{R}^{d\times d} \to \mathbb{R}^{d\times d}, \ X \mapsto P_MX$$
$$\phi_3 : \mathbb{R}^d \to \mathbb{R}^{d\times d}, \ u \mapsto u \otimes u,$$

hence using the chain rule for Frechet derivatives (e.g. [5, Theorem VII.3.3]) we find

$$D \bar{f}(x) = D\phi_1((\phi_2 \circ \phi_3)(x))D\phi_2(\phi_1(x))D\phi_1(x).$$

Working componentwise we find that the Frechet derivative of $\phi_1$ at $X \in \mathbb{R}^{d\times d}$ is the linear form

$$\mathbb{R}^{d\times d} \to \mathbb{R}, \ Y \mapsto \sum_{i,j=1}^d X_{ij} Y_{ij}$$

and the derivative of $P_M$ (a linear map) at $X$ is simply (cf. [5, Example VII.2.3a])

$$\mathbb{R}^{d\times d} \to \mathbb{R}^{d\times d}, \ Y \mapsto P_M Y.$$  

Furthermore, a simple calculation shows that the partial derivatives of $\phi_3$ (these are of course matrix-valued and the Frechet derivative of $\phi_3$ can then be found by [5, Theorem VII.2.8(i)]) are given by

$$\frac{\partial}{\partial u_k}(u \otimes u) = \begin{pmatrix} 0 & \cdots & 0 & u_1 & 0 & \cdots & 0 \\ \vdots & & \vdots & & \vdots & \cdots & \vdots \\ 0 & \cdots & 0 & u_{k-1} & 0 & \cdots & 0 \\ u_1 & \cdots & u_{k-1} & 2u_k & u_{k+1} & \cdots & u_d \\ 0 & \cdots & 0 & u_{k+1} & 0 & \cdots & 0 \\ \vdots & & \vdots & & \vdots & \cdots & \vdots \\ 0 & \cdots & 0 & u_d & 0 & \cdots & 0 \end{pmatrix} =: B_k(u), \ k = 1, \ldots, d$$  

69
Putting everything together we find that

$$\nabla \bar{f}(u) = \left( \sum_{i=1}^{d} (P_M(u \otimes u))_{i,j} (P_M(B_k(u)))_{i,j} \right)_{k=1, \ldots, d}.$$

Furthermore, since we want to move an a sphere (an embedded manifold) we need a retraction (for a definition see e.g. [1, Section 4.1]). In order to arrive at an efficient algorithm we use the simple choice

$$R_u(\xi) = \frac{u + \xi}{\|u + \xi\|_2},$$

from [1, Section 4.6.2].

Finally, we determine the step width using a an Armijo-rule, cf. e.g. [1, Algorithm 2 (p76)], and arrive at Algorithm 4.3.

**Algorithm 6** Weight recovery by gradient ascent on the sphere

**Require:** Initial iterate $u_0$ with $\|u_0\|_2 = 1$, Matrix space $\mathcal{M}$, scalars $\bar{\alpha} > 0$, $\beta, \sigma \in (0, 1)$

1: for $k = 0, 1, 2, \ldots$ do
2: \hspace{0.5cm} Compute $\eta_k = P_M(u_k \nabla \bar{f}(u_k))$
3: \hspace{0.5cm} Compute the smallest integer $m \geq 0$ such that
4: \hspace{2cm} $P_M(R_{u_k}(\bar{\alpha}\beta^m\eta_k)) \geq P_M(u_k) + \sigma \bar{\alpha}\beta^m\eta_k \|\eta_k\|_2^2$
5: \hspace{0.5cm} Set $u_{k+1} = R_{u_k}(\bar{\alpha}\beta^m\eta_k)$
6: end for

**A simple adaptive initialization procedure**

Since there will be local maxima it is clear that also this algorithm requires restarts. However, as long the underlying weights are nearly orthogonal it is very likely that nearly orthogonal initial vectors result in distinct recovered weights. Therefore it should be enough to start the algorithm with a random vector and then restart in the orthogonal complement of the previously recovered vectors. This way it could be enough to start the algorithm only once for each weight. Figure 4.3 illustrates the idea: The first initial vector (i.e. the initial value for Algorithm 4.3 or a variant) is drawn randomly on the unit sphere in weight space (we will use the uniform distribution on the sphere). This initial vector is used by Algorithm 4.3 to retrieve a first approximation of a weight from the (approximated) auxiliary space. The next initial vector is sampled randomly from the intersection of the unit sphere (in weight space) with the orthogonal complement of the space spanned by the first approximated weight. Since the orthogonal complement of this one-dimensional space is a hyperplane the intersection is again a sphere, this time of lower dimension, and we use again the uniform distribution (on the new sphere). If the weights are nearly orthogonal then a gradient based algorithm starting from this orthogonal complement should lead to a new, distinct weight. Having found
Figure 4.3.: Illustration of the simple adaptive initialization strategy. The first initial vector is drawn randomly on the unit sphere (left part). Using this initial vector a first approximation of a weight (using Algorithm 4.3 or a variant thereof) is retrieved. The second initial vector is now drawn randomly from the intersection of the unit sphere with the orthogonal complement of the space spanned by the first approximated weight (thick ring on the sphere on the right part). This initial vector is used in turn by Algorithm 4.3 or variant thereof to approximate the next weight.

the second approximated weight the third initial vector is then drawn randomly from the intersection of the unit sphere with the orthogonal complement of the two dimensional space spanned by the first two weights. This procedure is repeated until all weights are approximated. If the weights are indeed nearly orthogonal then as explained above each approximation should belong to a distinct weight and hence to retrieve $m$ weights from the auxiliary space should need only $m$ starts of the optimization algorithm 4.3.

Algorithm 4.3 provides a more formal description of this strategy.

**Algorithm 7** Adaptive initialization strategy for gradient ascent on the sphere

**Require:** Number of weights to retrieve $m$
1: Sample $u_1$ uniformly from $\mathbb{S}^{d-1}$
2: Approximate the first weight $\tilde{a}_1$ using $u_1$ (i.e. start Algorithm 4.3 with $u_1$)
3: for $k = 2, \ldots, m$ do
4: Sample $u_k$ uniformly from $\mathbb{S}^{d-1} \cap \text{span}\{\tilde{a}_1, \ldots, \tilde{a}_{k-1}\}$
5: Approximate weight $\tilde{a}_k$ using $u_k$ (i.e. start Algorithm 4.3 with $u_k$)
6: end for

**Remark 4.3.** 1. We like to point out that the adaptive initialization strategy is rather generic and other gradient based algorithms for weight recovery, i.e. variants of Algorithm 4.3, should work with Algorithm 4.3.

2. Algorithm 4.3 is a rather simple adaptive initialization strategy based on the geometric intuition explained in this section and relies strongly on the near-orthogonality
assumption. In particular, if the weights are not nearly orthogonal then using the orthogonal complement could pose a problem. For example later initial values could (due to non-orthogonality) be close to already recovered weights. Note that this can be easily detected by checking whether an approximation has already been found earlier and additional restarts can be performed. Another option for dealing for less nearly orthogonal weights could be the usage of near orthogonal complements. For example, one could increase the allowed non-orthogonality of the initial vectors with each run or successful approximation of a new weight.

3. Our adaptive initialization strategy depends on the optimization in weight space (Algorithm 4.3) instead of matrix space (Algorithm 4.2). It might be possible to implement a similar strategy in matrix space, e.g. sampling from orthogonal complements of previous approximated weights within the auxiliary space.

4.4. Numerical experiments: Recovering weights from full auxiliary space

We now apply the numerical methods from the preceding section to various settings.

Preliminary: Testing implementation in one layer case

First, to check the correctness of our implementation we try to reproduce the results from [35, Section 5.1]. To be precise, networks with one hidden layer and \( m = [20, 20, 5] \) are considered and the activation functions are of the form

\[
g_{i_1}^{[1]}(s) = c_{i_1} \tanh(s + b_{i_1}).
\]

Networks of this type are generated by sampling \( c_1, \ldots, c_{m[1]} \) i.i.d. \( \mathcal{N}(1, 0.2) \) and \( b_1, \ldots, b_{m[1]} \) i.i.d. \( \mathcal{N}(0, 0.2) \). Altogether 60 such networks are generated, the auxiliary space is approximated using Algorithm 3.3 (sampling uniformly from the unit sphere) and Algorithm 4.2 is run (with \( \gamma = 2 \) and 100 iterations as STOP criterion) 180 times (with random initial matrices). A weight is considered to be recovered by a vector from Algorithm 4.2 if its distance (up to sign) is less or equal 0.05. The results are summerized in Figure 4.4. This seems to reproduce [35, Figure 4] and hence our implementation of Algorithms 3.3 and 4.2 seems to be correct.

Testing gradient ascent on the sphere for one hidden layer

Next we test the gradient based optimization in weight space for weight recovery, cf. Algorithm 4.3 and Algorithm 4.3. First we look at an exact auxiliary space. We sampled 20 0.4-nearly orthogonal vectors \( a_1, \ldots, a_{20} \) of unit length from \( \mathbb{R}^{20} \) and consider

\[
\mathcal{M} = \text{span}\{a_1 \otimes a_1, \ldots, a_{20} \otimes a_{20}\}
\]
Figure 4.4.: Reproducing [35, Figure 4]

Figure 4.5.: Running Algorithm 4.3 from a random initial value. Shown are the distance of tensor product of the current product to the auxiliary space (this is the shifted negative of the objective function) on the left, the norm of the gradient (of the objective function) in the middle and the distances of the current iterate to every underlying weight $a_i$ on the right.

We sample a vector uniformly from the unit sphere and run Algorithm 4.3 with $\bar{\alpha} = 1$, $\beta = 0.5$, $\sigma = 0.5$, starting from this random vector. For now we use as stopping criterion simply the number of iterations (here 50). The results of this are in Figure 4.5. Using the same example we now run Algorithm 4.3 (using Algorithm 4.3 with the same parameters as above for the weight approximation) to retrieve all weights. The results can be found in Figure 4.6. Finally we apply the method to an approximated auxiliary space, using the architecture from [35, Section 5] (albeit with constant scalings equal to one) and 100 samples, the results are reported in Figure 4.7.

**Revisiting the two layer case: Recovery from exact auxiliary space**

Next, we complement the numerical investigations from [82, Chapter 5] and revisit the case of two hidden layers. The question is whether the weights and mixed weights can be recovered in principle when the auxiliary space is perfectly known. We consider
<table>
<thead>
<tr>
<th>Start id</th>
<th>Closetest weight</th>
<th>Distance weight</th>
<th>Num iter</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>13</td>
<td>1.610960e-16</td>
<td>49</td>
</tr>
<tr>
<td>1</td>
<td>19</td>
<td>2.775178e-16</td>
<td>49</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1.514807e-16</td>
<td>49</td>
</tr>
<tr>
<td>3</td>
<td>16</td>
<td>2.277232e-15</td>
<td>49</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td>3.184841e-14</td>
<td>49</td>
</tr>
<tr>
<td>5</td>
<td>11</td>
<td>2.330467e-15</td>
<td>49</td>
</tr>
<tr>
<td>6</td>
<td>8</td>
<td>1.938316e-16</td>
<td>49</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>2.248609e-16</td>
<td>49</td>
</tr>
<tr>
<td>8</td>
<td>6</td>
<td>1.508511e-16</td>
<td>49</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>2.574180e-16</td>
<td>49</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>2.430645e-16</td>
<td>49</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>1.602344e-16</td>
<td>49</td>
</tr>
<tr>
<td>12</td>
<td>4</td>
<td>2.140117e-16</td>
<td>49</td>
</tr>
<tr>
<td>13</td>
<td>7</td>
<td>2.320967e-16</td>
<td>49</td>
</tr>
<tr>
<td>14</td>
<td>17</td>
<td>1.996967e-16</td>
<td>49</td>
</tr>
<tr>
<td>15</td>
<td>2</td>
<td>1.548866e-16</td>
<td>49</td>
</tr>
<tr>
<td>16</td>
<td>18</td>
<td>2.952956e-16</td>
<td>49</td>
</tr>
<tr>
<td>17</td>
<td>12</td>
<td>1.222211e-16</td>
<td>49</td>
</tr>
<tr>
<td>18</td>
<td>14</td>
<td>1.331111e-16</td>
<td>49</td>
</tr>
<tr>
<td>19</td>
<td>15</td>
<td>1.813097e-16</td>
<td>49</td>
</tr>
</tbody>
</table>

Figure 4.6.: The results of Algorithm 4.3 on a simple example. Shown are the results of all starts.
<table>
<thead>
<tr>
<th>Start id</th>
<th>Closetest weight</th>
<th>Distance weight</th>
<th>Num iter</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>16</td>
<td>0.002372</td>
<td>49</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>0.005366</td>
<td>49</td>
</tr>
<tr>
<td>2</td>
<td>17</td>
<td>0.001305</td>
<td>49</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
<td>0.001614</td>
<td>49</td>
</tr>
<tr>
<td>4</td>
<td>14</td>
<td>0.001311</td>
<td>49</td>
</tr>
<tr>
<td>5</td>
<td>11</td>
<td>0.000391</td>
<td>49</td>
</tr>
<tr>
<td>6</td>
<td>18</td>
<td>0.002988</td>
<td>49</td>
</tr>
<tr>
<td>7</td>
<td>15</td>
<td>0.001557</td>
<td>49</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
<td>0.003190</td>
<td>49</td>
</tr>
<tr>
<td>9</td>
<td>19</td>
<td>0.001344</td>
<td>49</td>
</tr>
<tr>
<td>10</td>
<td>3</td>
<td>0.001969</td>
<td>49</td>
</tr>
<tr>
<td>11</td>
<td>12</td>
<td>0.005067</td>
<td>49</td>
</tr>
<tr>
<td>12</td>
<td>6</td>
<td>0.005614</td>
<td>49</td>
</tr>
<tr>
<td>13</td>
<td>0</td>
<td>0.002976</td>
<td>49</td>
</tr>
<tr>
<td>14</td>
<td>1</td>
<td>0.008087</td>
<td>49</td>
</tr>
<tr>
<td>15</td>
<td>13</td>
<td>0.002292</td>
<td>49</td>
</tr>
<tr>
<td>16</td>
<td>7</td>
<td>0.001713</td>
<td>49</td>
</tr>
<tr>
<td>17</td>
<td>2</td>
<td>0.002228</td>
<td>49</td>
</tr>
<tr>
<td>18</td>
<td>8</td>
<td>0.000235</td>
<td>49</td>
</tr>
<tr>
<td>19</td>
<td>10</td>
<td>0.002328</td>
<td>49</td>
</tr>
</tbody>
</table>

Figure 4.7.: Retrieving all weights from approximated auxiliary space.
networks with various sizes \((m = [20, 10, 5], m = [20, 5, 5], m = [20, 20, 5])\) and levels of near-orthogonality \(\epsilon_{no} = 0, 0.25, 0.5, 1\). The activation functions are of the form

\[
g_{i_l}^{[l]}(s) = \tanh(s + b_{i_l}^{[l]}), \quad i_l = 1, \ldots, m^{[l]}, \ l = 1, 2
\]

and we generate different networks by sampling \(b_1^{[l]}, \ldots, b_{m^{[l]}}^{[l]}\) i.i.d. \(\mathcal{N}(0, 0.2), \ l = 1, 2\). All other settings are the same as in the previous experiment. For now we are only interested in the proportion of weights (and mixed weights) recovered as well as the proportion of wrong or spurious vectors, i.e. vectors from Algorithm 4.2 that are more than 0.05 (up to sign) away from the nearest weight or mixed weight. Note that this is a more stringent requirement than used in [82, Chapter 5], but recall that for now we use the exact auxiliary space. The results of this experiment are reported in Figure 4.8. Of course the question arises whether the situation improves with a larger input space. We repeat the experiment with different widths, the results are reported in Figure 4.9. The situation with input dimension 20 and 40 seems to be qualitatively similar, so this indicates that for now we can continue with rather small input dimensions to save computation time.
Figure 4.9.: Extracting vectors from exact auxiliary space for two hidden layers.
Three layer case: Recovery from exact auxiliary space

We now repeat the previous experiment, but this time for three hidden layers. Using otherwise identical settings we arrive at the results reported in Figure 4.10.

Recovery from approximated auxiliary space

We now proceed to use the approximated auxiliary space to recover the weights. Following [35] and [82] we assume that a potential dimensionality reduction has already been performed, i.e. only the case $m^{[0]} = m^{[1]}$ is considered here. We sampled randomly 40 networks of widths $[20, 20, 10, 5]$, $[20, 20, 10, 5, 3]$, respectively, all with $tanh$ activation functions, random biases (sampled i.i.d. normal with mean 0 and SD 0.1) and near-orthogonal weights with $\epsilon_{no} = 0.5$ (in each layer). For each network we approximated the auxiliary space with 500 finite difference approximation of the Hessians (using step size 0.00001) at points uniformly distributed on the unit sphere and ran 500 starts of Algorithm 4.2 with $\gamma = 2$. The results (averaged over 40 trials) are shown in Figure 4.4. For validation and comparison we performed the same experiment also for a two layer network of form $[20, 20, 10]$, its results are shown in Figure 4.4. The results for three and four hidden layers are promising, but definitely worse than the case.
Figure 4.11.: Performance of weight recovery using the full approximated auxiliary space for three, four and five hidden layers. The left column contains the distribution of distances of final iterates to the closest true weight (averaged over all for the particular setting). The right column shows the proportion of false positives (i.e. proportion of final iterates that are more than a certain threshold away from the closest true weight or mixed weight), the proportion of (mixed) weights recovered for each network considered (i.e. proportion of weights or mixed weights for which at least one final iterate is closer than a certain threshold) and proportion of first layer weights, depending on a prescribed threshold, respectively.
of two hidden layers. However, as already observed in [82, Chapter 6] the radius of the sphere used for sampling the Hessians seems to play an important role. We therefore repeat the experiment (now with only 20 repetitions to shorten the computation time) trying different radii \(2, 4, \ldots, 20\) of the sphere used to sample from, the results are shown in Figure 4.4 and the two layer case is displayed for this experiment in Figure 4.4 for comparison.

**Discussion**

It is clear from Figures 4.4 and 4.4 that the performance deteriorates for more than two hidden layers. In particular, though not excellent the performance for two hidden layers is already quite good with the standard settings. However, (cf. also the discussion in [82, Section 6]) the case of more than two hidden layers seems to strongly improve by choosing a better radius of the sphere used for sampling, Figures 4.4 and 4.4. Unfortunately, the recovery success of the mixed weights seems to be still mediocre. Interestingly, the performance dependence on the radius used is different for two and three hidden layers and four and five. The observations in this section indicate that the approximation quality can be improved by choosing a suitable sampling distribution. Note however that the recovery success cannot be used to determine such a distribution or even finding good parameters for a distribution since the underlying weights (and potentially even the activation functions) are not known. Inspecting the singular values during the subspace approximation might be helpful, cf. also [82, Section 6].

**4.5. Distinguishing weights and mixed weights**

As mentioned before the recovery methods presented here cannot distinguish between weights and mixed weights. Formally, having approximations \(\Omega = \{\omega_1, \ldots, \omega_N\} \subseteq \mathbb{R}^{n[0]}\) of weights and normalized mixed weights (or their negative) we need a method for deciding whether a particular \(\omega_n\) is a weight or mixed weight and if \(L \geq 3\) also which
Figure 4.13.: Performance of weight recovery using the full approximated auxiliary space for three, four and five hidden layers using different radii for sampling the Hessians. The left column the proportion of false positives, the proportion of (mixed) weights recovered for each network considered and the proportion of first layer weights recovered using a threshold of 0.2, depending on the radius used. The right column shows the distribution of distances (cf. also Figure 4.4) for a given radius.
For the case of two hidden layers and sigmoidal activation functions an empirical method for this task was proposed in [82], [32] based on the saturation property of the sigmoidals. Adapting this method to the case of three or more hidden layers and comparing it with the algorithm described in this section is left to future work.

Here we present preliminary work on a novel and different approach for deciding whether a particular $\omega_n$ is an approximation of a weight or a mixed weight that should not depend on a particular shape of the activation functions. Deciding which layer a mixed weight belongs to is left as future work. Furthermore, we restrict ourselves to investigating only the case of perfect recovery, i.e. using $\Omega = \{a_1^{[1]}, \ldots, a_{m[1]}^{[1]}, v_1^{[2]}, \ldots, v_{m[L]}^{[L]}\}$.

We first explain the intuition behind the approach, then present a concrete algorithm which is then evaluated empirically. Similar to [32, Section 6.1] we do not provide formal guarantees for now.

**Motivation and intuition**

Recall that the Hessians have the form

$$H[f](x) = \sum_{i_1=1}^{m[1]} s_{i_1}^{[1]}(x) a_{i_1}^{[1]} \otimes a_{i_1}^{[1]} + \sum_{l=2}^{L} \sum_{i_l=1}^{m[l]} s_{i_l}^{[l]}(x) v_{i_l}^{[l]}(x) \otimes v_{i_l}^{[l]}(x)$$

and having appropriate random inputs they should generically concentrate around the space

$$\mathcal{M} = \text{span}_{[\mathbb{R}^{m[1]} \times [0])} \{v_{i_l}^{[l]}(0) \otimes 2 | i_l = 1, \ldots, m[l], l = 1, \ldots, L\}.$$

Furthermore, we may assume that there exists a certain separation between $a_{i_1}^{[1]}$ and $v_{i_l}^{[l]}$ in an heuristic sense (or rather between $\text{span}\{a_{i_1}^{[1]}\}$ and $\text{span}\{v_{i_l}^{[l]}\}$), otherwise recovering weights from the auxiliary space would already be problematic. For example, if in the
nearly orthonormal case \( L = 2 \), \( g_1^{[1]} = \ldots = g_m^{[1]} \), \( \left(g_1^{[1]}ight)'(0) \approx 1 \), \( m^{[2]} \) large and all \( a_1^{[2]} \) are dense then \( a_1^{[1]} \) and \( v_i^{[l]} \) are also nearly orthogonal.

Note that the Hessian has two components: A linear combination of tensor products of the weights (which do not depend on the input) with coefficients that depend on the input and a linear combination of the tensor products of \( v_i^{[l]} \), i.e. vectors that depend on the input (again with coefficients that depend on the input). The key observation now is that \( a_1^{[1]} \otimes a_1^{[1]} \in \mathcal{M} \), but in general \( v_i^{[l]}(x) \otimes v_i^{[l]}(x) \notin \mathcal{M} \) (only \( v_i^{[l]}(0) \otimes v_i^{[l]}(0) \) are used in the span above). Let’s now use random inputs so that \( H[f](x) \) becomes a random variable and fluctuates (around \( \mathcal{M} \)). Using an appropriate distribution for the inputs (e.g. uniform distribution on a sphere) the variance of the Hessian is finite and we can interpret this as having a finite budget of fluctuation. Since the Hessians are linear combinations of certain tensor products (with coefficients depending on the input though) this fluctuation budget has to be shared among all components \( a_1^{[1]} \otimes a_1^{[1]} \) \( (i_1 = 1, \ldots, m^{[1]}), v_i^{[l]}(x) \otimes v_i^{[l]}(x) \) \( (i_1 = 1, \ldots, m^{[l]}, l = 2, \ldots, L) \). Consider now scalar products \( \langle H[f](X), \omega \rangle \), \( \omega \in \Omega \) which are scalar random variables when \( X \) is random. Note that they can be interpreted as one dimensional features of \( \omega \), compare with [32, Section 6] and the discussion there. Assuming now near-orthogonality, the generic separation between weights and mixed weights alluded to above and the concentration around \( \mathcal{M} \) we should have \( \langle H[f](X), a_1^{[1]} \rangle \approx s_1^{[1]}(X) \). This means that this scalar product can get its full share from the fluctuation budget for \( a_1^{[1]} \) through \( s^{[1]}(X)_{i_1} \). In contrast the scalar product \( \langle H[f](X), v_i^{[l]} \rangle \), \( l = 2, \ldots, L \), does not get its full share since some of the fluctuation gets lost since \( v_i^{[l]}(0) \otimes v_i^{[l]}(0) \in \mathcal{M} \) (for \( l = 2, \ldots, L \), but is not a component in the (input dependent) linear combination of \( H[f](X) \). Of course, since the situation is rather complicated (coefficients and tensor products that depend both on the input and are not independent) it is not to be expected that e.g.

\[
\text{Var}(\langle H[f](X), a_1^{[1]} \rangle) >> \text{Var}(\langle H[f](X), v_i^{[l]}(0) \rangle) \quad l = 2, \ldots, L, \ i_1 = 1, \ldots, m^{[l]}
\]

However, it should be extremely unlikely that \( \text{Var}(\langle H[f](X), v_i^{[l]}(0) \rangle) \) for a certain \( i_1 \) and \( l \geq 2 \) is much larger than all \( \text{Var}(\langle H[f](X), a_1^{[1]} \rangle) \). Indeed, this is what can be observed in almost all the cases we inspected numerically and is demonstrated in Figure 4.5. Summerizing our discussion, if we inspect the (empirical) variances of \( \langle H[f](X), \omega \rangle \) for \( X \) sampled appropriately, then rather large variances should belong to weights. This basic idea will now be made precise and used in a concrete algorithm.

**Iterative projection and resampling of Hessians**

Assume that

\[
\Omega = \{a_1^{[1]}, \ldots, a_m^{[1]}, v_1^{[2]}, \ldots, v_{m^{[l]}}^{[l]}\},
\]

and that all weights and mixed weights \( v_i^{[l]} \) are nearly orthogonal. Based on the previous considerations we propose the following strategy: Sample \( x_1, \ldots, x_N \) from a radially
Figure 4.15.: Typical examples of the variances of the projections for networks with $L = 2$, $d = 100$, $m = (40, 20, 10)$ (left), $L = 3$, $d = 100$, $m^{[1]} = 10$, $m^{[2]} = 5$ (center), $L = 3$, $m = (40, 30, 20, 10)$ (right). For each example $\epsilon_{qo} = 0.01$ and all activation functions are $\tanh$ functions with random biases. Horizontal axis shows the index of the vectors (blue dots corresponding to weights, all other colors to mixed vectors ordered according to $l$), vertical axis is the variance of the projections. Calculated using 100 random points chosen i.i.d. uniformly from the unit ball, finite difference approximation of the Hessians with $\epsilon_{FD} = 0.01$ and the Tensorflow float64 datatype.

symmetric distribution (in the following the uniform distribution from a ball of certain radius will be used) and calculate the empirical variances of $\langle H[f](X), \omega \rangle$ for all $\omega \in \Omega$. Label the $\omega$ with the largest variance a weight and remove it from $\Omega$. Repeat this procedure $m^{[1]} - 1$ times, but now sample from the orthogonal complement of (the span of) the previously selected $\omega$. The method is summarized in Algorithm 4.5, called Iterative Projection and Resampling of Hessians for short. The working of the algorithm

<table>
<thead>
<tr>
<th>Algorithm 8 Distinguishing weights with IPRH</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Require:</strong> Set $\Omega \subseteq \mathbb{R}^{m^{[0]}}$ of weights and mixed weights, neural network $f$</td>
</tr>
<tr>
<td>1:</td>
</tr>
<tr>
<td>2: <strong>for</strong> $i = 1, \ldots, m^{[1]}$ <strong>do</strong></td>
</tr>
<tr>
<td>3: Sample $N$ Hessians $H<a href="x_k">f</a>$ where $x_k$ i.i.d. uniformly from the unit ball</td>
</tr>
<tr>
<td>4: Calculate the sample variance for each $\langle H<a href="x">f</a>, \omega \rangle$ for all $\omega \in \Omega$</td>
</tr>
<tr>
<td>5: Let $\omega^<em>$ be the vector with the highest variance. Set $\Omega_w = \Omega_w \cup {\omega^</em>}$ and $\Omega = \Omega \setminus {\omega^*}$.</td>
</tr>
<tr>
<td>6: Set $f(x) = f(P_{(\text{span}\Omega_w)^\perp}x)$</td>
</tr>
<tr>
<td>7: <strong>end for</strong></td>
</tr>
<tr>
<td>8: <strong>return</strong> $\Omega_w$, $\Omega$</td>
</tr>
</tbody>
</table>

is illustrated in Figure 4.5 for a network with two hidden layers.

**Numerical experiments**

In order to assess the reliability some neural networks are generated for different configurations and Algorithm 4.5 is applied to the real weights and mixed weights. We
Figure 4.16: Example run of IPRH using true weights and mixed vectors. Shown are the projected variances for a $L = 2$, $d = 100$, $m = (40, 20, 10)$, $\epsilon_{q0} = 0.01$, blue points correspond to weights, green to mixed weights. For brevity only selected iterations are shown, though in this example all weights are correctly detected. Iteration 0 corresponds to the first iteration (before any resampling), Iteration 19 is the last iteration where the last missing weight is detected. Calculated using $N = 20$ samples in each run with finite difference approximation of the Hessians with $\epsilon_{FD} = 0.01$ and the Tensorflow float64 datatype.
are interested in the amount of failures, where a failure means that a mixed vector is mistaken for a weight in one iteration.

First, networks with tanh activation functions and randomly sampled biases are tested. Each iteration uses \( N = 20 \) samples, finite difference approximation of the Hessians with \( \epsilon_{\text{FD}} = 0.01 \) and the Tensorflow float64 datatype. The result is shown in Figure 4.5.

Next the previous experiment is repeated, but this time we use sine functions for the activations, the results are shown in Figure 4.5. Finally, to make sure that more than one oscillation of the sine is used we repeated the experiment with sines having a higher frecuence, i.e. we use \( \sin(10 \cdot (\bullet + \text{bias})) \) for the activation functions. The results are shown in Figure 4.5. It is clear that at least when using the exact weights and mixed weights Algorithm 4.5 can reliable distinguish weights and mixed weights, even when not using a sigmoidal activation function. Furthermore, since we used a rather small \( N \) and a constant radius of the ball for resampling its performance might be further improved by tuning these parameters. Note that the basic version presented here assumes that all weights are contained in \( \Omega \) (though not necessarily all mixed weights). Dealing with missing weights is left for future work.

---

<table>
<thead>
<tr>
<th>( L )</th>
<th>( m )</th>
<th>Number of runs</th>
<th>Failures</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>(20, 10, 5)</td>
<td>1000</td>
<td>3 (0.3%)</td>
</tr>
<tr>
<td>3</td>
<td>(40, 20, 10, 5)</td>
<td>1000</td>
<td>7 (0.7%)</td>
</tr>
<tr>
<td>4</td>
<td>(40, 30, 20, 10, 5)</td>
<td>2000</td>
<td>50 out of 2000 (2.5%)</td>
</tr>
</tbody>
</table>

Figure 4.17.: Results for the first IPRH experiment

<table>
<thead>
<tr>
<th>( L )</th>
<th>( m )</th>
<th>Number of runs</th>
<th>Failures</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>(20, 10, 5)</td>
<td>1000</td>
<td>5 (0.5%)</td>
</tr>
<tr>
<td>3</td>
<td>(40, 20, 10, 5)</td>
<td>1000</td>
<td>15 (1.5%)</td>
</tr>
<tr>
<td>4</td>
<td>(40, 30, 20, 10, 5)</td>
<td>2000</td>
<td>48 out of 2000 (2.4%)</td>
</tr>
</tbody>
</table>

Figure 4.18.: Results for the second IPRH experiment

<table>
<thead>
<tr>
<th>( L )</th>
<th>( m )</th>
<th>Number of runs</th>
<th>Failures</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>(20, 10, 5)</td>
<td>1000</td>
<td>8 (0.8%)</td>
</tr>
<tr>
<td>3</td>
<td>(40, 20, 10, 5)</td>
<td>1000</td>
<td>0 (0%)</td>
</tr>
<tr>
<td>4</td>
<td>(40, 30, 20, 10, 5)</td>
<td>2000</td>
<td>6 out of 2000 (0.6%)</td>
</tr>
</tbody>
</table>

Figure 4.19.: Results for the third IPRH experiment
4.6. Numerical experiments: Weight recovery from alternative auxiliary space

We now continue the numerical experiments from Section 3.4. Recall that for the first experiment 40 random networks with nearly orthogonal weights (of level 0.5) have been generated for 2, 3 and 4 hidden layers and fixed scalings of 1 within the tanh activation functions. Using Algorithm 4.2 with 200 starts and the approximated alternative auxiliary space all weights from the first layer have been recovered in all test cases with error less than 0.1

Repeating this experiment for random scalings leads to full recovery (of all first layer weights) in 90% of all testcases (for each setting with 2, 3, 4 hidden layers), not surprisingly these cases correspond to the ones with small approximation error of the auxiliary space. In ca 10% recovery success deteriorates, but interestingly still more than half of all first layer weights have been recovered in each of those cases.

To further evaluate this alternative strategy and allow for a better comparison with weight recovery from the full auxiliary space we now apply the alternative strategy to the setting from the last subsection (using uniform sampling from the hypercube of sidelengths 1, 2, . . . , 25 with 100 samples per side-length). In all cases all the first layer weights have been recovered (with threshold significantly less than 0.1) and no spurious results occurred. The distribution of distances of the final iterates are displayed in Figure 4.6.
Figure 4.20.: Distribution of distances of final iterates to closest (first layer) weights using the two stage approximation method for the alternative auxiliary space. For details on the histograms cf. Figure 4.4.
5. Conclusion

Summary

The present work was concerned with learning functions of the form

\[ f(x) = \sum_{i_L=1}^{m[L]} g^{[L]}(W^{[L]}g^{[L-1]}(W^{[L-1]} \ldots g^{[1]}(W^{[1]}x) \ldots), \]

i.e. functions represented by a feedforward neural network with multiple hidden layers and linear activation functions in the last layer. More precisely, we worked on the question of how to reconstruct such a function in a data-efficient (i.e. trying to use only few function evaluations) and computationally feasible manner when the function can be actively queried and no noise is present. Since it is well-known that in general this is a very difficult problem one has to make assumptions about the underlying neural network. Here we used assumptions analogous to the ones used in [35] and [82] and similarly to the latter restricted our focus to normalized weights (in particular first layer weights). In Chapter 1 we gave an overview of the setting and discussed the structure of first and second order derivatives of functions represented by a feedforward neural network. In Chapter 2 we used the structure of first order derivates to generalize the dimensionality reduction scheme from [35] and [82] to multiple hidden layers. In Chapter 3 we used the structure of second order derivatives to generalize the subspace approximation scheme from [82, Chapter 3] (including its analysis) to multiple hidden layers. Furthermore, we proposed an extended approximation scheme that enjoys empirical success and is easier to work with, though there’s no rigorous analysis available at the moment. In Chapter 4 we followed [82, Chapter 5] and used the weight recovery scheme from [35] to recover weights and mixed weights from the approximated auxiliary space. Furthermore we proposed a new algorithm for weight recovery from auxiliary spaces based on geometrical considerations. In addition, we investigated the weight recovery schemes using the approximated alternative auxiliary space from Chapter 3.

Discussion

Let us briefly discuss the results of this work and put it into context. We were able to generalize the dimensionality reduction scheme from [35] and [82] leading to a practical algorithm with guarantees. Note that the latest version of [35] also presented a dimensionality reduction scheme based on passive sampling (i.e. selecting the right inputs is not necessary anymore) allowing for noise. We conjecture that this approach can be
easily adapted to the setting of multiple hidden layers, however, implementing such an algorithm and adapting the analysis is beyond the scope of this work.

In Chapter 3 we were able to generalize the subspace approximation scheme from [82, Chapters 3,4] to multiple hidden layers. In particular, also the concentration results from this latter work were generalized to multiple hidden layers. However, since we closely followed the strategy used in [82, Chapter 3] our work suffers from the same drawbacks. In particular there seems be a certain bias of the approximated auxiliary space with respect to the true auxiliary space which limits the precision and success of the weight recovery. Furthermore, similar to [82] our analysis also does not discuss the choice of the sampling distribution.

As an alternative we proposed an extended approximation scheme focusing on a different (reduced) alternative auxiliary space. Though the preliminary numerical experiments indicate empirical success, so far this approach is based entirely on geometric intuition and numerical observations and no rigorous analysis is available yet. Some concrete questions regarding this extended approximation scheme have been collected below. We would also like to point out that many interesting questions should be investigated numerically and which are beyond the scope of the present work. In particular an important question is how many samples should be drawn and which sampling parameters should be used. More precisely (with the notation of Section 3.4), how large should be $N_1$ (number of different sampling distributions), $N_2$ (dimension of the intermediate auxiliary spaces $\tilde{A}_i$) and $N_x$ (number of samples for approximating each intermediate auxiliary space)?

Finally, Chapter 4 has been concerned with weight recovery from the approximated matrix subspaces. Similarly to [82] we did not perform a rigorous analysis and also used the methodology from [35] for multiple hidden layers without adaption to this more complicated situation. However, we introduced two novelties: First, a new weight recovery algorithm (for recovering weights from the auxiliary spaces) based on geometric considerations has been proposed. Again, so far no rigorous mathematical analysis is available, but numerical evidence indicates good success. Since the proposed algorithm is just a standard gradient algorithm on the sphere (interpreted as a Riemannian manifold) adapted to the present situation we conjecture that standard analysis tools (e.g. [1] combined with the analysis of [35]) should lead to rigorous guarantees for this algorithm. Concrete questions in this direction are again collected below. Finally, we used the weight recovery methods of this chapter on the approximated alternative auxiliary space from Section 3.4. On the one hand, as already mentioned above, since no rigorous analysis of the extended approximation scheme is available at the moment there are also no guarantees on the success of the combined method (i.e. using the weight recovery algorithms on the approximated alternative auxiliary space). On the other hand note that the alternative auxiliary space has the same structure as the auxiliary space from [35, Chapter 3], and hence as soon as guarantees on the approximation error of the alternative auxiliary space become available the analysis from the latter work can be used. Furthermore, since the alternative space should contain little to no components arising from the mixed weights the problems with the auxiliary space reported in Chapter 4 and [82, Chapters 5,6] should not arise here.
Let us summerize what has been achieved in this work and what is still missing. Chapter 2 presents an efficient dimensionality reduction scheme for the first hidden layer in the case of active querying without noise and also provides guarantees on the approximation. Chapter 3 provides a computationally tractable method to approximate a helpful subspace (the auxiliary space $\mathcal{M}$) and provides guarantees on the approximation. However, we were only able to generalize the approach from [82] to multiple hidden layers, not to improve it (or its analysis). We were able to develop a new computationally tractable (and probably data efficient) approximation scheme for a simpler subspace (the alternative auxiliary space) that allows higher approximation precision and is more amenable to weight recovery from it (cf. Chapter 4). However, at the moment neither a rigorous analysis of this approach is available nor a way to systematically extend it to weights from higher layers. Using the auxiliary space for weight recovery we are essentially in the situation of [82], but this time for multiple hidden layers: If the network is benign and the approximation of the auxiliary space good enough (cf. Chapter 4) we can recover weights and mixed weights in many cases. In principle this could be used to gain more information about the underlying network like weights in higher layers or information about the activation functions (e.g. biases if a shifted fixed nonlinearity is used), compare with the discussion in [82, pp 83,84]. However, no rigorous guarantees are available yet and the problem of distinguishing weights and mixed weights (compare also with [82, Section 5.2]) persists. Finally, in the situation of no mixed weights in the matrix space (i.e. the setting of [35] or using the alternative auxiliary space) an efficient weight recovery algorithm has been presented which might be able to speed up the retrieval of all weights using an adaptive initialization procedure. Note that we assumed sufficient regularity of the activation functions and therefore did not consider e.g. ReLUs (but cf. the observations in Section 2.3 and also the discussion in [82, pp 84,85]).

In short, we can effectively reduce the dimensionality in the first hidden layer (with guarantees) and (at least empirically, though without rigorous guarantees) recover the (normalized) weights from the first layer for networks with multiple hidden layers. Similarly to [82] we can gather valuable information about weights in higher layers (using the approximated auxiliary space and potentially recovered mixed weights), which can be used for example to initialize gradient descent methods, cf. e.g. [109].

Open questions

We would like to close with some interesting and concrete open questions arising from the present work.

- **Alternative auxiliary space** Under what conditions on the network can (nontrivial) bounds on the approximation error of the approximated auxiliary space $\tilde{\mathcal{A}}$ be derived? Which sampling scheme (uniform sampling from sphere, uniform sampling from hypercube or something different) is most efficient (w.r.t. number of samples)?

- **Alternative auxiliary space for mixed weights** Can be an approximation scheme
(similar to the one in Section 3.4) be devised for the mixed weights? The same questions as in the previous item would arise.

- **Active sampling for the auxiliary space** In [35] passive sampling schemes have been introduced (that rely only on knowledge of the sampling distribution but not the ability to freely choose the inputs). Can the approximation schemes from the last two items be transferred to this setting?

- What about non-differentiable activation functions: Can the methods and their analysis from Chapters 2, 3 and 4 be adapted to this setting? In particular, can the last two items be answered for this setting?

- **Weight recovery using gradient ascent on the unit sphere** Which guarantees (especially convergence, convergence rates and recovery success) can be derived for the gradient ascent algorithm from Section 4.3? Is the simple adaptive initialization procedure proposed in the same chapter reliable and efficient? In particular, can the analysis based on near-orthogonality from [35] be used for these questions?
S. Supplementary material

S.1. Background material from Linear Algebra and Matrix Analysis

S.1.1. Singular values and singular vectors

In this section the most important definitions and results related to the singular value decomposition are briefly recalled. For convenience we generalize the definition of a diagonal matrix: Let \( m, n \in \mathbb{N} \) and \( d \in \mathbb{R}^{\min\{m,n\}} \), then define the \( m \times n \) matrix by

\[
\text{diag}(m,n)(d) =: \text{diag}(m,n)(d_1, \ldots, d_{\min\{m,n\}}) = \begin{cases} d_i & \text{if } 1 \leq i \leq \min\{m,n\} \\ 0 & \text{otherwise} \end{cases}
\]

or using a case distinction,

\[
\text{diag}(m,n)(d) = \begin{cases} \left( \begin{array}{cc} \text{diag}(d) & 0_{m \times (n-m)} \\ 0_{(m-n) \times n} & \text{diag}(d) \end{array} \right) & m < n \\ \text{diag}(d) & m = n \\ \left( \begin{array}{c} \text{diag}(d) \\ 0_{(m-n) \times n} \end{array} \right) & m > n \end{cases}
\]

**Definition S.1. SVD**

Let \( A \in \mathbb{R}^{m \times n} \) be a matrix with \( m, n \in \mathbb{N} \). A decomposition of the form \( A = U \Sigma V^T \) with \( U \in \mathbb{O}(m) \), \( V \in \mathbb{O}(n) \), \( \Sigma = \text{diag}(\sigma_1, \ldots, \sigma_{\min\{m,n\}}) \) and \( \sigma_1 \geq \ldots \geq \sigma_{\min\{m,n\}} \) is called a [singular value decomposition (SVD)](https://en.wikipedia.org/wiki/Singular_value_decomposition) of \( A \). The numbers \( \sigma_1 \geq \ldots \geq \sigma_{\min\{m,n\}} \) are called [singular values](https://en.wikipedia.org/wiki/Singular_value_decomposition), the vectors \( U_1, \ldots, U_m \) left singular vectors, the vectors \( V_1, \ldots, V_n \) right singular vectors.

An essential part of the analysis will be Wedin’s bound [99] which is related to the stability of SVD under perturbations. Here we recall this result in the form [92, Section 7], [35, Theorem 6.6] that is used in the main text.

**Theorem S.2. Wedin bound**

Let \( B, \tilde{B} \in \mathbb{R}^{m \times n} \) with SVD

\[
B = \left( \begin{array}{cc} U_1 & U_2 \\ \Sigma_1 & 0 \end{array} \right) \left( \begin{array}{c} \Sigma_1 \\ 0 \end{array} \right) \left( \begin{array}{c} V_1^T \\ V_2^T \end{array} \right)
\]

\[
\tilde{B} = \left( \begin{array}{cc} \tilde{U}_1 & \tilde{U}_2 \\ \tilde{\Sigma}_1 & 0 \end{array} \right) \left( \begin{array}{c} \tilde{\Sigma}_1 \\ 0 \end{array} \right) \left( \begin{array}{c} \tilde{V}_1^T \\ \tilde{V}_2^T \end{array} \right)
\]

93
where the corresponding components of the SVD have the same sizes for $B$ (with $\Sigma$ a $k \times k$ matrix) and $\sigma_1 \geq \sigma_2 \geq \ldots$ and $\tilde{\sigma}_1 \geq \tilde{\sigma}_2 \geq \ldots$ are the corresponding singular values ordered from highest to lowest. Furthermore, let $\alpha > 0$ such that

$$\min_{1 \leq i \leq k} |\tilde{\sigma}_i - \sigma_j| \geq \alpha,$$

$$\min_{1 \leq i \leq k} |\tilde{\sigma}_i| \geq \alpha.$$ Then we have

$$\|V_1 V_1^T - \tilde{V}_1 \tilde{V}_1^T\|_F \leq \frac{2}{\alpha} \|B - \tilde{B}\|_F.$$ 

### S.1.2. Norms

For completeness we briefly review some norms commonly used in the main text.

**Definition S.3. $\ell_p$-norm**

For any $0 \leq p < \infty$ the $\ell_p$-norm on $\mathbb{R}^d$ (sometimes called $\ell_p^d$-norm or just $p$-norm) is defined for any $x \in \mathbb{R}^d$ by

$$\|x\|_p = \left( \sum_{i=1}^d |x_i|^p \right)^{\frac{1}{p}}.$$ 

In addition, the $\ell_\infty$-norm (sometimes called $\ell_\infty^d$-norm or maximum norm) is defined by

$$\|x\|_\infty = \max_{i=1,\ldots,d} |x_i|.$$ 

It is clear that the norms just defined are indeed norms and hence turn $\mathbb{R}^d$ into a normed vector space. We also need various norms defined for matrices.

**Definition S.4.** For a matrix $A \in \mathbb{R}^{m \times n}$ we define its $\ell_p$-$\ell_q$-operator norm with $0 < p, q < \infty$ by

$$\|A\|_{p \rightarrow q} = \sup_{\|x\|_p = 1} \|Ax\|_q.$$ 

Furthermore, its **Frobenius norm** is defined by

$$\|A\|_F = \sqrt{\sum_{i=1}^m \sum_{j=1}^n a_{ij}^2}.$$ 

It is clear that the norms just defined are indeed norms on $\mathbb{R}^{m \times n}$. Note that often the $\ell_2$-$\ell_2$-operator norm $\|\cdot\|_{2 \rightarrow 2}$ is denoted by $\|\cdot\|_\infty$. Here we follow [36] to avoid confusion with the $\|\cdot\|_\infty$ norm on $\mathbb{R}^d$.

The following simple properties are folklore and details can be found in any book on matrix analysis or related areas, cf. e.g. [56] or [48].

**Proposition S.5.** Let $A \in \mathbb{R}^{m \times n}$ be an arbitrary matrix.
1. Let \( \sigma_1 \geq \ldots \geq \sigma_{\min(m,n)} \geq 0 \) be the singular values of \( A \), then
\[
\|A\|_F = \sqrt{\sum_{i=1}^{\min(m,n)} \sigma_i^2}
\]
and
\[
\|A\|_{2 \rightarrow 2} = \max_{i=1, \ldots, \min(m,n)} \sigma_i.
\]
In particular \( \|A\|_{2 \rightarrow 2} \leq \|A\|_F \).

2. For any \( U \in \mathbb{O}(m) \), \( V \in \mathbb{O}(n) \) we have \( \|UAV\|_F = \|A\|_F \) and \( \|UAV\|_{2 \rightarrow 2} = \|A\|_{2 \rightarrow 2} \), i.e. \( \cdot \| \) and \( \cdot \|_{2 \rightarrow 2} \) are unitarily invariant.

3. For any \( B \in \mathbb{R}^{m \times k} \) we have
\[
\|AB\|_F \leq \|A\|_F \|B\|_{2 \rightarrow 2}
\]
and
\[
\|AB\|_F \leq \|A\|_{2 \rightarrow 2} \|B\|_F
\]
as well as
\[
\|AB\|_{2 \rightarrow 2} \leq \|A\|_{2 \rightarrow 2} \|B\|_{2 \rightarrow 2}
\]

### S.1.3. Vectorization

We frequently have to switch between vectors and matrices using vectorization and its inverse.

**Definition S.6.** Let \( A \in \mathbb{R}^{m \times n} \) be a matrix, then we define its vectorization by
\[
vec(A) = \left( a_{(i-1)n+1, (i-1)\mod n+1} \right)_{i=1, \ldots, mn} \in \mathbb{R}^{mn}
\]

The vectorization of a matrix is hence nothing else than all its concatenated together into a vector. We collect a few simple results on this operation.

**Lemma S.7.** Vectorization \( vec : \mathbb{R}^{m \times n} \rightarrow \mathbb{R}^{mn} \) is an isometric isomorphism between \( \mathbb{R}^{m \times n} \) (with the topology generated by \( \| \cdot \|_F \)) and \( \mathbb{R}^{mn} \) (with the topology generated by \( \| \cdot \|_2 \)). Its inverse is given by
\[
unvec ((a_i)_{i=1, \ldots, mn}) = (a_{(i-1)n+j})_{i=1, \ldots, m, j=1, \ldots, n}
\]
S.1.4. Near-orthogonality

An important concept in the analysis in the main text are vectors that are nearly orthogonal or orthonormal. The following definition has been taken from [35, Definition 1.1].

Definition S.8. Nearly orthonormal

1. We call vectors $a_1, \ldots, a_m \in \mathbb{R}^d$, $m \leq d$ $\epsilon$-nearly orthonormal if $S(a_1, \ldots, a_m) \leq \epsilon$, where

$$S(a_1, \ldots, a_m) =: \inf \left\{ \sqrt{\sum_{i=1}^{m} \| a_i - \bar{a}_i \|_2^2} \mid \bar{a}_1, \ldots, \bar{a}_m \in \mathbb{R}^d \text{ orthonormal} \right\}$$

2. A matrix $A \in \mathbb{R}^{d \times m}$, $m \leq d$, is called $\epsilon$-nearly orthonormal if $S(A) \leq \epsilon$, where

$$S(A) =: \inf_{\bar{A} \in \mathcal{O}_c(d \times m)} \| A - \bar{A} \|_F,$$

and $\mathcal{O}_c(d \times m)$ denotes all $d \times m$ matrices with orthonormal columns.

Remark S.9. The concepts of near-orthonormality for vectors and matrices, respectively, are basically equivalent. Indeed, let $a_1, \ldots, a_m \in \mathbb{R}^d$ and define $A = (a_1 \ \cdots \ a_m) \in \mathbb{R}^{d \times m}$. By definition of the Frobenius norm

$$S(a_1, \ldots, a_m) = \inf \left\{ \sqrt{\sum_{i=1}^{m} \| a_i - \bar{a}_i \|_2^2} \mid \bar{a}_1, \ldots, \bar{a}_m \in \mathbb{R}^d \right\} = \inf \left\{ \| A - \bar{A} \|_F \mid \bar{A} = (\bar{a}_1 \ \cdots \ \bar{a}_m), \bar{a}_1, \ldots, \bar{a}_m \in \mathbb{R}^d \text{ orthonormal} \right\} = \inf_{\bar{A} \in \mathcal{O}_c(d \times m)} \| A - \bar{A} \|_F = S(A).$$

If $a_1, \ldots, a_m$ are $\epsilon$-nearly orthonormal, then the corresponding matrix $A$ is also $\epsilon$-nearly orthonormal. Conversely, if a matrix $A \in \mathbb{R}^{d \times m}$ is $\epsilon$-nearly orthonormal, then its columns $A_1, \ldots, A_m$ are $\epsilon$-nearly orthonormal vectors.

We will use this equivalence in the following by working with vectors or matrices whenever one perspective is more convenient than the other.

In general it is easier to work with orthonormal vectors instead of just nearly orthonormal vectors. An important question is therefore which orthonormal vectors are closest to a given set of nearly orthonormal vectors, or equivalently, finding the closest (in $\| \cdot \|_F$-norm) column-orthonormal matrix for a given nearly orthonormal matrix. It turns out that using the SVD finding these orthonormal vectors and orthonormal matrix, respectively, is easy.

Proposition S.10. Optimal orthonomal vectors / matrix
1. Any set of vectors $a_1, \ldots, a_m \in \mathbb{R}^d$, $m \leq d$, is $S(a_1, \ldots, a_m)$-nearly orthogonal (or quasiorthogonal) with

$$S(a_1, \ldots, a_m) = \sqrt{\sum_{i=1}^{m} (\sigma_i - 1)^2},$$

where $A = U \Sigma V^T$ is the SVD of the matrix $A = (a_1 \cdots a_m) \in \mathbb{R}^{d \times m}$ and a set of orthonormal vectors closest to $a_1, \ldots, a_m$ in Euclidean distance is given by the columns of $UV^T$.

2. Any matrix $A \in \mathbb{R}^{m \times d}$, $m \leq d$, is $S(A)$-nearly orthonormal with

$$S(a_1, \ldots, a_m) = \sqrt{\sum_{i=1}^{m} (\sigma_i - 1)^2},$$

where $A = U \Sigma V^T$ (with $\Sigma = \text{diag}(\sigma)$) is the SVD of $A$. A column-orthonormal matrix closest to $A$ in Frobenius norm is given by $U V^T$.

Proof. For 1. see [35, Theorem 6.2(i)] and its proof, 2. follows immediately from Remark S.9.

Next some useful properties of nearl orthonormal vectors and matrices are collected.

**Proposition S.11. Properties of near-orthogonality**

Let $A \in \mathbb{R}^{d \times m}$, $d \geq m$, be a $\epsilon$-nearly orthonormal matrix.

(i) $(1 - \epsilon) \leq \|A\|_{2 \to 2} \leq (1 + \epsilon)$

(ii) $(1 + 2\epsilon_{qo}) \|D\|_F \leq \|A^T DA\|_F \leq (1 + 2\epsilon_{qo}) \|D\|_F$ for any diagonal matrix $D$

Proof. See e.g. [82, Lemma 5, 1. and 3.].

---

**S.2. Background material from probability theory**

**S.2.1. Measure concentration for scalar random variables**

In order to derive concentration inequalities we have to work with convenient classes of probability distributions. One such class is defined next.

**Definition S.12.** An integrable real random variable $X$ with mean $\mu$ is called subgaussian if there exists $\sigma > 0$ such that

$$\mathbb{E}[\exp(\theta(X - \mu))] \leq \exp \left( \theta^2 \frac{\sigma^2}{2} \right)$$

for all $\theta \in \mathbb{R}$. 

97
There are various equivalent definitions of subgaussian random variables, see e.g. [96, Proposition 2.5.2] for an overview. A particularly convenient characterization is presented next.

**Definition S.13. Subgaussian random variable**

Let $X$ be an integrable random variable with mean 0. Then the subgaussian norm of $X$ is defined by

$$
\|X\|_{\psi_2} =: \inf \left\{ t > 0 \mid \mathbb{E} \left[ \exp \left( \frac{X^2}{t^2} \right) \right] \leq 2 \right\}.
$$

The name is justified by the following result.

**Proposition S.14. Properties of subgaussian RV**

1. $\| \cdot \|_{\psi_2}^2$ is a norm on the space of subgaussian random variables.

2. Let $X$ be an integrable random variable with mean 0. Then $X$ is a subgaussian random variable if and only if $\|X\|_{\psi_2} < \infty$.

3. If $X$ is a subgaussian random variable with $\sigma > 0$ from Definition S.12, then

$$
\|X - \mathbb{E}[X]\|_{\psi_2}^2 \leq C\sigma^2
$$

for an absolute constant $C > 0$ (i.e. $C$ is independent of $X$).

*Proof. See e.g. [96, Section 2.5.2].

Many important types of random variables are subgaussian. For now, we contend ourselves with simple examples, one presented in the next result.

**Proposition S.15. Bounded RV are sub-gaussian**

Every bounded real random variable $X$ is sub-gaussian and

$$
\|X - \mathbb{E}[X]\|_{\psi_2} \leq \frac{1}{\sqrt{\log 2}} \|X\|_{\infty}
$$

*Proof. See [96, Example 2.5.8 3.].

The following proposition is a first example of the usefulness of the notion of subgaussianity.

**Proposition S.16. Mean of maximum of subgaussian RV**

Let $X_1, \ldots, X_N$ be subgaussian random variables with mean zero. Let $C > 0$ such that

$$
\mathbb{E}[\exp(\theta X_i)] \leq \exp(\theta^2 C)
$$

for all $i = 1, \ldots, N$ and $\theta \in \mathbb{R}$. We then have

$$
\mathbb{E} \left[ \max_{i=1,\ldots,N} X_i \right] \leq \sqrt{4C \log N}.
$$
Note that in the preceding proposition the random variables $X_1, \ldots, X_N$ do not have to be i.i.d.

Proof. See [36, Proposition 7.29].

The main reason for working with subgaussian random variables is that powerful concentration results are available. For the most relevant for our context is given by the Hoeffding inequality (for subgaussian random variables). Note that there exist various versions including special cases (e.g. for weighted sums of binary random variables or weighted sums of bounded random variables). We use the following variant, cf. [96, Theorem 2.6.2].

**Theorem S.17. Hoeffding inequality**

Let $X_1, \ldots, X_N$ be independent, zero mean, subgaussian random variables. Then there exists an absolute constant $C^{(S.17)} > 0$ such that for any $t \geq 0$

$$
P\left[ \left| \sum_{k=1}^N X_k \right| \geq t \right] \leq 2 \exp \left( -\frac{C^{(S.17)}t^2}{\sum_{k=1}^N \|X_k\|_{\psi_2}^2} \right)
$$

**S.2.2. Measure concentration for random matrices**

The following theorem ([95, Remark 5.3]) generalizes the well-known classical Chernoff bound. Note that it is actually a simplification of the more general [95, Theorem 5.1].

**Theorem S.18. Matrix Chernoff (Upper bound on smallest EVaL)**

Let $X_1, \ldots, X_N$ be independent, positive semidefinite random matrices, all of dimension $m \times m$. Assume that

$$
\sigma_1(X_j) \leq C \quad \mathbb{P}\text{-a.s.}
$$

for a certain $C \geq 0$ and all $j = 1, \ldots, N$ and define

$$
\mu_{\text{min}} = \sigma_m \left( \sum_{j=1}^N \mathbb{E}[X_j] \right),
$$

It follows for all $s \in (0, 1)$ that

$$
P \left[ \sigma_m \left( \sum_{j=1}^N X_j \right) \leq (1 - s)\mu_{\text{min}} \right] \leq m \exp \left( \frac{\mu_{\text{min}}s^2}{2C} \right)
$$

We need also bounds for inner singular values of sums of dyadic random matrices. The following result is due to M. Rauchensteiner and contained in his proof of [82, Lemma 14].

**Theorem S.19. Inner singular values** Let $X_1, \ldots, X_N$ be i.i.d. $d$-dimensional random vectors such that

$$
\|X_1\|_2 \leq C \quad \mathbb{P}\text{-a.s.}
$$
(so that in particular \(X_1\) is integrable) and set \(\alpha = \sigma_{d_0}(\mathbb{E}[X_1^{\otimes 2}])\) for \(2 \leq d_0 \leq d\). We then have for all \(t \in (0, 1)\) that

\[
P \left[ \sigma_{d_0} \left( \sum_{n=1}^{N} X_n^{\otimes 2} \right) \leq (1 - t)N\alpha \right] \leq d_0 \exp \left( -\frac{t^2N\alpha}{2C} \right)
\]

The proof is based on the next theorem ([40, Theorem 4.1], we omitted one bound for brevity) which generalizes the basic matrix Chernoff theorem to inner eigenvalues.

**Theorem S.20.** *Matrix Chernoff for inner eigenvalues*

Let \(X_1, \ldots, X_N\) be independent, positive semidefinite random matrices of sizes \(d \times d\). Let \(1 \leq n_0 \leq d\) be an integer and define

\[
\mu_{n_0} = \lambda \left( \sum_{k=1}^{N} \mathbb{E}[X_k] \right).
\]

Furthermore, let

\[
\Psi : \bigcup_{1 \leq n \leq d} \mathbb{O}_c(d \times n) \to \mathbb{R}
\]

be a function such that

\[
\max_{k=1, \ldots, N} \lambda_{\max}(V^T X_j V) \leq \Psi(V) \quad \text{a.s. for all } V \in \bigcup_{1 \leq n \leq d} \mathbb{O}_c(d \times n).
\]

and \(V_- \in \mathbb{O}_c(d \times d - n_0 + 1)\) and \(V_+ \in \mathbb{O}_c(d \times n_0)\) given matrices such that

\[
\mu_{d_0} = \lambda_{\max} \left( \sum_{k=1}^{N} V_+^T \mathbb{E}[X_k] V_+ \right) = \lambda_{\min} \left( \sum_{k=1}^{N} V_-^T \mathbb{E}[X_k] V_- \right).
\]

Then for all \(0 \leq t < 1\)

\[
P \left[ \lambda_{n_0} \left( \sum_{k=1}^{N} X_k \right) \leq (1 - t)\mu_{n_0} \right] \leq n_0 \left( \frac{e^t}{(1 - t)(1 - t)} \right)^{\mu_{n_0} \Psi(V_-)}.
\]

**Proof. of Theorem S.19** Obviously \(X_1^{\otimes 2}\) are independent, identically distributed and symmetric random matrices of size \(d \times d\). In addition, since for each \(z \in \mathbb{R}^{d^2}\) and \(x \in \mathbb{B}^d\)

\[
z^T X^{\otimes 2} z = (x^T z)^2 \geq 0
\]

all of these matrices are also positive semidefinite. Furthermore, since \(\|X_1\|_2\) and hence all \(\|X_k^{\otimes 2}\|_F, k = 1, \ldots, N\), are bounded the random matrices \(X_k^{\otimes 2}\) are integrable. Since each realization of these matrices is symmetric, positive definite this is also the case for the expectation and an eigenvalue decomposition exists,

\[
\mathbb{E}[X_1^{\otimes}] = U \Lambda U^T.
\]

100
Assume that the eigenvalues are ordered, i.e.

$$\lambda = \text{diag}(\lambda_1, \ldots, \lambda_d), \quad \lambda_1 \geq \ldots \geq \lambda_d \geq 0$$

and that \(u_1, \ldots, u_d \in \mathbb{R}^d\) are the corresponding (orthonormal) eigenvectors. Define now

$$U_- = (u_1 \ldots u_d), \quad U_+ = (u_d \ldots u_d),$$

and note that \(U_- \in \mathbb{O}_c(d^2 \times d_0), U_+ \in \mathbb{O}_c(d^2 \times d^2 - d_0 + 1)\). By construction we have

$$\lambda_{\text{max}} \left( \sum_{n=1}^{N} U^T \mathbb{E}[X_1^{\otimes 2}] U_+ \right) = N \alpha$$
$$\lambda_{\text{min}} \left( \sum_{n=1}^{N} U^T \mathbb{E}[X_1^{\otimes 2}] U_- \right) = N \alpha$$

and almost surely

$$\|X_1^{\otimes 2}\|_{2 \rightarrow 2} \leq \|X_1^{\otimes 2}\|_F = \|X_1\|_2^2 \leq C^2.$$

Since all \(X_1^{\otimes 2}\) (and their expectations) are symmetric positive semidefinite the singular values and eigenvalues are the same. Theorem S.20 can be used. The claim follows by the usual simplication.

**S.2.3. Technical results for selected distributions**

For different sampling strategies various technical results on certain distribution are needed which we collect here. By default it is assumed that the distributions are defined w.r.t. to the appropriate Borel \(\sigma\)-algebra. The first simple, but very useful result from [82, Chapter 3.2] applies to a broad range of distributions, in particular uniform distributions on spheres and balls.

**Proposition S.21. Rotation-invariants dists and dists of components**

Let \(X = (X_1, \ldots, X_d)^T\) be a \(d\)-dimensional multivariate random variable (i.e. a \(d\)-dimensional random vector) whose distribution is rotationally invariant. Furthermore, let \(A \in \mathbb{R}^{n \times d}\) with \(d > n > 0\) be a row-orthonormal matrix. Then

$$AX \sim \begin{pmatrix} X_1 \\ \vdots \\ X_n \end{pmatrix},$$

i.e. \(AX\) (a \(n\)-dimensional random vector) and the first \(n\) components of \(X\) have the same distribution.

**Proof.** Let \(B \in \mathbb{R}^{(d-n) \times d}\) be a row-orthonormal matrix such that

$$\bar{A} = \begin{pmatrix} A \\ B \end{pmatrix} \in \mathbb{O}(d).$$
Since by assumption the distribution of $X$ is rotationally invariant we have
\[ AX \sim A(A^TX) = \begin{pmatrix} X_1 \\ \vdots \\ X_n \end{pmatrix}, \]
and the claim follows.

Next we recall the definition of important distributions and some of their properties.

**Definition S.22. Beta distribution**

We say that a scalar real random variable $X$ has *Beta*($\alpha, \beta$)-distribution (with shape parameters $\alpha, \beta > 0$) if the corresponding image measure w.r.t. to the Lebesgue measure has density
\[ \text{Beta}(\alpha, \beta)(x) = \frac{\Gamma(\alpha + \beta)}{\Gamma(\alpha)\Gamma(\beta)} x^{\alpha-1} (1-x)^{\beta-1}, \]
where $\Gamma$ is the usual Gamma function.

**Definition S.23. $\chi^2$ distribution**

Let $Z_1, \ldots, Z_d \overset{i.i.d.}{\sim} N(0,1)$. The distribution of the random variable
\[ X = Z_1^2 + \ldots + Z_d^2 \]
is called *$\chi^2$-distribution (with $d$ degrees of freedom)* and its density is denoted by $\chi^2(d)$.

**Lemma S.24. Some properties of Beta distribution**

1. **Mean of Beta-distribution** Let $X \sim \text{Beta}(\alpha, \beta)$, then $\mathbb{E}[X] = \frac{\alpha}{\alpha + \beta}$.

2. **Beta from $\chi^2$** Let $X_1 \sim \chi^2(n)$, $X_2 \sim \chi^2(m)$ be independent with $n, m \in \mathbb{N}$, then
\[ \frac{X_1}{X_1 + X_2} \sim \text{Beta} \left( \frac{m}{2}, \frac{n}{2} \right) \]

*Proof.* 1. See e.g. [31, p. 55]

2. The $\chi^2$ density with $d$ degrees of freedom is the same as twice the so-called Gamma density with scale parameter 1 and shape parameter $\frac{d}{2}$, see e.g. [31, p. 71, Item 1]. By a well-known formula (e.g. [31, p. 59, Item 7] the claim follows.

The *Beta*-distribution is closely related to uniform distributions on spheres. For convenience we recall the next result ([82, Lemma 9]) and its proof here.

**Proposition S.25. Uniform and Beta distributions**

Let $X \sim U(S^{d-1})$ and $A \in \mathbb{R}^{n \times d}$ be row-orthonormal for $d > n > 0$. We then have
\[ \|AX\|_2^2 \sim \text{Beta} \left( \frac{n}{2}, \frac{d-n}{2} \right) \]
Proof. Define

\[ X_{[n]} = \begin{pmatrix} X_1 \\ \vdots \\ X_n \end{pmatrix}, \]

then \( AX \sim X_{[n]} \) since the uniform distribution on the unit sphere is rotationally invariant, cf. Proposition S.21. Let \( Z_1, \ldots, Z_d \overset{i.i.d.}{\sim} \mathcal{N}(0, 1) \) and define \( Z = (Z_1, \ldots, Z_d)^T \), then it is well-known that (e.g.

\[ \frac{Z}{\|Z\|_2} \sim \mathcal{U}(S^{d-1}) \]

and therefore

\[ X_i \sim \frac{Z_i}{\|Z\|_2}. \]

Combining the results so far we get

\[
\|AX\|_2^2 \sim \|X_{[n]}\|_2^2 \\
= X_1^2 + \ldots + X_n^2 \\
\sim \frac{Z_1^2 + \ldots + Z_n^2}{Z_1^2 + \ldots + Z_n^2 + \left(Z_{n+1}^2 + \ldots + Z_d^2\right)} \\
= \frac{Y_1}{Y_1 + Y_2},
\]

where we defined \( Y_1 = Z_1^2 + \ldots + Z_n^2 \) and \( Z_{n+1}^2 + \ldots + Z_d^2 \). By definition \( Y_1 \sim \chi^2(n) \) and \( Y_2 \sim \chi^2(d - n) \) and \( Y_1, Y_2 \) are independent. According to Lemma S.24 2. we finally get

\[
\|AX\|_2^2 \sim \frac{Y_1}{Y_1 + Y_2} \sim \text{Beta} \left( \frac{n}{2}, \frac{d - n}{2} \right).
\]

Measure concentration using subgaussianity will play an important role in the main text. Conveniently, Beta-distributed random variables are subgaussian and their properties have been thoroughly investigated. The next proposition collects the most relevant properties in this context.

**Proposition S.26.** Beta RV are subgaussian

Let \( X \sim \text{Beta}(\alpha, \beta) \) for some \( \alpha, \beta > 0 \).

1. \( X \) is sub-gaussian.

2. We have for the smallest \( \sigma = \sigma(\alpha, \beta) \) from Definition S.12 that

\[
\sigma(\alpha, \beta) \leq \frac{1}{4(\alpha + \beta + 1)}
\]
3. We have for the subgaussian norm of centered $X$ that

$$\|X - \mathbb{E}[X]\|_2^2 \leq \frac{C}{4(\alpha + \beta_1)}$$

with $C$ from Proposition S.14 3.

**Proof.** For 1. and 2. see [68], in particular their Theorem 2.1. For 3. use 2. and Proposition S.14 3. \hfill \Box

### S.3. Technical auxiliary results

Here some technical results are collected that are used in the main text, but which would either reduce the readability considerably or which are peripheral to the main arguments.

**Lemma S.27. Bound on perturbations of outputs**

Let $f$ be a neural network function, let $x$ be from its domain, $1 \leq \hat{L} \leq L$, $1 \leq i \leq m^{[0]}$ and define $\hat{x} = x + \eta e_i$ for $|\eta| \leq \epsilon$, $\epsilon > 0$, then we have

$$\|y^{[\hat{L}]}(\hat{x}) - y^{[\hat{L}]}(x)\|_2 \leq \left( \prod_{i=1}^{\hat{L}} \kappa_{[1]}^{[i]} \right) \left( \prod_{i=2}^{L} \|W^{[i]}\|_F \right) \left( \sqrt{\sum_{i_1=1}^{m^{[1]}} (a^{[1]}_{i_1})^2} \right) \epsilon$$

**Proof.** We get

$$\|y^{[\hat{L}]}(\hat{x}) - y^{[\hat{L}]}(x)\|_2 = \sqrt{\sum_{i_1=1}^{m^{[1]}} |a^{[\hat{L}]}_{i_1} \langle a^{[\hat{L}]}_{i_1}, y^{[\hat{L}-1]}(\hat{x}) \rangle - a^{[\hat{L}]}_{i_1} \langle a^{[\hat{L}]}_{i_1}, y^{[\hat{L}-1]}(x) \rangle|^2}$$

$$\leq \sum_{i_1=1}^{m^{[1]}} \kappa_{[1]}^{[i_1]} \left( \sum_{i_2=1}^{m^{[2]}} \|a^{[\hat{L}]}_{i_2}\|^2 \|y^{[\hat{L}-1]}(\hat{x}) - y^{[\hat{L}-1]}(x)\|_2 \right)$$

$$\leq \sum_{i_1=1}^{m^{[1]}} \kappa_{[1]}^{[i_1]} \left( \prod_{i_2=1}^{L} \|W^{[i]}\|_F \right) \|y^{[1]}(\hat{x}) - y^{[1]}(x)\|_2$$

$$\leq \sum_{i_1=1}^{m^{[1]}} \kappa_{[1]}^{[i_1]} \left( \prod_{i_2=1}^{L} \|W^{[i]}\|_F \right) \left( \sqrt{\sum_{i=1}^{m^{[1]}} (a^{[1]}_{i_1})^2} \right) \epsilon$$

where we used Lipschitz continuity of the activation functions for $\mathbb{1}$, Cauchy-Schwarz for $\mathbb{2}$ and we repeated these steps (using also the definition of $\|\cdot\|_F$). Finally, the Lipschitz
continuity of the first layer activation functions and the definition of the perturbation lead to

\[
\|y^{[1]}(\tilde{x}) - y^{[1]}(x)\|_2 = \sqrt{\sum_{i_1=1}^{m^{[1]}} \left| g_{i_1}^{[1]} \left[ \langle a_{i_1}^{[1]} , \tilde{x} \rangle \right] - g_{i_1}^{[1]} \left[ \langle a_{i_1}^{[1]} , x \rangle \right] \right|^2} \\
\leq \kappa_1^{[1]} \sqrt{\sum_{i_1=1}^{m^{[1]}} \left| \langle a_{i_1}^{[1]} , \eta e_i \rangle \right|^2} \\
\leq \kappa_1^{[1]} \left( \sqrt{\sum_{i_1=1}^{m^{[1]}} \left( a_{i_1}^{[1]} \right)^2} \right) \epsilon
\]

which was used for \( \delta \).

The following corollary can be useful because in some situations one can assume that \( \|a^{[l]}\|_2 = 1 \) by simply rescaling the weights and activation functions.

**Corollary S.28.** If in the situation of the preceding lemma all weights in layers \( 2, \ldots, \tilde{L} \) have \( \| \cdot \|_2 \)-norm less or equal 1, then we have

\[
\|y^{[L]}(\tilde{x}) - y^{[L]}(x)\|_2 \leq \left( \prod_{l=1}^{\tilde{L}} \kappa_1^{[l]} \right) \left( \prod_{l=2}^{\tilde{L}} \sqrt{m^{[l]}} \right) \left( \sqrt{\sum_{i_1=1}^{m^{[1]}} \left( a_{i_1}^{[1]} \right)^2} \right) \epsilon
\]

**Proof.** The result follows immediately from Lemma S.27 since

\[
\prod_{l=2}^{L} \|W^{[l]}\|_F = \prod_{l=2}^{L} \sqrt{\sum_{i_1=1}^{m^{[l]}} \|a_{i_1}^{[l]}\|^2} \leq \prod_{l=2}^{L} \sqrt{m^{[l]}}
\]

\( \square \)

**Lemma S.29.** *Bound on magnitude of partial derivatives*

\[
\left\| \frac{\partial y^{[L]}}{\partial x_i} (x) \right\|_2 \leq \left( \prod_{l=1}^{L} \kappa_1^{[l]} \right) \left( \prod_{l=2}^{L} \|W^{[l]}\|_F \right) \sqrt{\sum_{i_1=1}^{m^{[1]}} \left( a_{i_1}^{[1]} \right)^2}
\]
Proof. We have
\[
\left\| \frac{\partial y^{[L]}}{\partial x_i} (x) \right\|_2 = \sqrt{\sum_{i_{L-1}=1}^{m^{[L]}} \left| a_{i_{L}}^{[L]} \right| \left( z_{i_{L}}^{[L]} (x) \right) \left( \frac{\partial y^{[L-1]}}{\partial x_i} (x) \right)}
\]
\[
\lesssim \sum_{i_{L-1}=1}^{m^{[L]}} \left\| a_{i_{L}}^{[L]} \right\| \left\| \frac{y^{[L-1]}}{\partial x_i} (x) \right\|_2
\]
\[
\lesssim \sum_{i_{L-1}=1}^{m^{[L]}} \left\| a_{i_{L}}^{[L]} \right\|_2 \left\| \frac{y^{[L-1]}}{\partial x_i} (x) \right\|_2
\]
\[
\lesssim \prod_{i_{L-1}=1}^{m^{[L]}} \left\| a_{i_{L}}^{[L]} \right\| \prod_{l=1}^{L} \left\| W^{[l]} \right\|_F \left\| \frac{y^{[L-1]}}{\partial x_i} (x) \right\|_2
\]
where the boundedness of the activation functions was used for (1), Cauchy-Schwarz for (2), the definition of \( \| \cdot \|_F \) for (3). The steps were then repeated down to layer 2 and finally the boundedness of the derivatives of the activation functions in layer 1 was used.

Corollary S.30. If in the situation of Lemma S.29 \( \| a_{i_l}^{[l]} \|_2 \leq 1 \) for all \( i_l = 1, \ldots, m^{[l]} \) and \( l = 2, \ldots, L \), then
\[
\left\| \frac{\partial y^{[L]}}{\partial x_i} (x) \right\|_2 \leq \left( \prod_{l=1}^{L} \kappa_{[l]} \right) \left( \prod_{l=2}^{L} \sqrt{m^{[l]}} \right) \left( \prod_{i_{L-1}=1}^{m^{[L]}} \left( \frac{\partial y^{[L-1]}}{\partial x_i} (x) \right) \right)_i
\]

Proof. Immediate from Lemma S.29, cf. also the proof of Corollary S.28.

Note that Lemma S.27 and Corollary S.28 also follow from Lemma S.29 and the Mean Value Theorem.

Lemma S.31. Bound on perturbation of partial derivatives
In the situation of Lemma S.27 we have
\[
\left\| \frac{\partial}{\partial x_i} y^{[L]} (\bar{x}) - \frac{\partial}{\partial x_i} y^{[L]} (\bar{x}) \right\|_2 \leq C_i^{[S.31]} \epsilon
\]
with
\[
C_i^{[S.31]} = \left( \sum_{i_{L-1}=1}^{m^{[L]}} \left( a_{i_{L}}^{[L]} \right) \right) \sum_{l=2}^{L} \sqrt{m^{[l-1]}} \left( \max_{i_{l-1}=1, \ldots, m^{[l]}} \left( a_{i_{l-1}}^{[l]} \right) \right) \left( \prod_{l=1}^{L} \left\| W^{[l]} \right\|_F \right) \left( \prod_{l=2}^{L} \left\| W^{[l-1]} \right\|_F \right) \kappa_{[l]} \kappa_{[l-1]} \left( \prod_{l=1}^{L} \kappa_{[l]} \right)
\]
\[
+ \left( \prod_{l=2}^{L} \left\| W^{[l]} \right\|_F \right) \left( \prod_{l=2}^{L} \kappa_{[l]} \right) \left( \sum_{i_{L-1}=1}^{m^{[L]}} \left( a_{i_{L}}^{[L]} \right) \right)
\]

106
Proof. For \( \bar{L} = 1 \) we find immediately

\[
\left\| \frac{\partial}{\partial x_i} y^{[1]}(\bar{x}) - \frac{\partial}{\partial x_i} y^{[1]}(x) \right\|_2 = \left\| \sum_{i=1}^{m[1]} \left( \left( g_i^{[1]} \right)' \left[ a_i^{[1]} \right] \right) \left( a_i^{[1]} \right) \right\|_2
\]

\[
\leq \sum_{i=1}^{m[1]} \left( \kappa_2^{[1]} \eta \right) \left( a_i^{[1]} \right)_i^4
\]

\[
\leq \kappa_2^{[1]} \left( \sum_{i=1}^{m[1]} \left( a_i^{[1]} \right)_i^4 \right) \varepsilon
\]

using Lipschitz continuity of the activation functions, the form of the perturbation, i.e.

\[
\langle a_i^{[1]} , \bar{x} \rangle - \langle a_i^{[1]} , x \rangle = \langle a_i^{[1]} , x + \eta c_i - x \rangle = \eta \left( a_i^{[1]} \right)_i,
\]

and that \( |\eta| \leq \varepsilon \). For \( \bar{L} \geq 2 \) we start with

\[
\left\| \frac{\partial}{\partial x_i} y^{[\bar{L}]}(\bar{x}) - \frac{\partial}{\partial x_i} y^{[\bar{L}]}(x) \right\|_2
\]

\[
(\#) \left\| G^{[\bar{L}]}(\bar{x}) W^{[\bar{L}]} G^{[\bar{L}-1]}(\bar{x}) \ldots G^{[1]}(\bar{x}) W^{[1]}(x) G^{[\bar{L}-1]}(x) \ldots G^{[1]}(x) W^{[1]} \right\|_2
\]

\[
(\#) \left\| \left( G^{[\bar{L}]}(\bar{x}) - G^{[\bar{L}]}(x) \right) W^{[\bar{L}]} G^{[\bar{L}-1]}(\bar{x}) \ldots G^{[1]}(\bar{x}) W^{[1]} \right\|_2
\]

\[
+ \left\| G^{[\bar{L}]}(x) \left( W^{[\bar{L}]} G^{[\bar{L}-1]}(\bar{x}) \ldots G^{[1]}(\bar{x}) W^{[1]} - W^{[\bar{L}]} G^{[\bar{L}-1]}(x) \ldots G^{[1]}(x) W^{[1]} \right) \right\|_2
\]

where we used Proposition 1.3 for (\#), the elementary identity \( \bar{x} \bar{y} - x y = (\bar{x} - x) \bar{y} + x (\bar{y} - y) \) for (\#), the submultiplicativity of the operator norm for (\#) and Proposition 1.3 for (\#).

Now, since by definition all \( G^{[l]} \) are diagonal matrices we get

\[
\left\| G^{[\bar{L}]}(\bar{x}) - G^{[\bar{L}]}(x) \right\|_2 \leq \max_{i_{\bar{L}} = 1, \ldots, m[\bar{L}]} \left| \left( g_i^{[\bar{L}]} \right)' \left[ a_i^{[\bar{L}]} \right] \right| - \left( \left( g_i^{[\bar{L}]} \right)' \left[ a_i^{[\bar{L}]} \right] \right)
\]

\[
= \kappa_2^{[\bar{L}]} \left( \max_{i_{\bar{L}} = 1, \ldots, m[\bar{L}]} \left\| a_i^{[\bar{L}]} \right\|_2 \right) \left\| y^{[\bar{L}-1]}(\bar{x}) - y^{[\bar{L}-1]}(x) \right\|_2.
\]

107
where we used additionally the Lipschitz continuity of the activation functions in layer \( \hat{L} \) and Cauchy-Schwarz. Similarly

\[
\|G^{[\hat{L}]}(x)\|_{2 \to 2} = \max_{i_L = 1, \ldots, m^{[\hat{L}]}} \| (g^{[\hat{L}]}_i)' \left[ \langle a^{[\hat{L}]}_i, y^{[\hat{L}-1]}(x) \rangle \right] \| \leq \kappa^{[\hat{L}]}_1. \tag{S.2}
\]

Furthermore, we have from Lemma S.27 that

\[
\| y^{[\hat{L}-1]}(\tilde{x}) - y^{[\hat{L}-1]}(x) \|_2 \leq \left( \prod_{l=1}^{L-1} \kappa^{[l]}_1 \right) \left( \prod_{l=2}^{L-1} \| W^{[l]} \|_F \right) \sqrt{\sum_{i_1=1}^{m^{[1]}} \left( a^{[1]}_{i_1} \right)^2} \epsilon \tag{S.3}
\]

and from Lemma S.29

\[
\left\| \frac{\partial}{\partial x_i} y^{[\hat{L}-1]}(\tilde{x}) \right\|_2 \leq \left( \prod_{l=1}^{L-1} \kappa^{[l]}_1 \right) \left( \prod_{l=2}^{L-1} \| W^{[l]} \|_F \right) \sqrt{\sum_{i_1=1}^{m^{[1]}} \left( a^{[1]}_{i_1} \right)^2} \tag{S.4}
\]

We can now continue with

\[
\left\| \frac{\partial}{\partial x_i} y^{[L]}(\tilde{x}) - \frac{\partial}{\partial x_i} y^{[L]}(x) \right\|_2 \\
\leq \| W^{[L]} \|_{2 \to 2} \left( \| G^{[\hat{L}]}(\tilde{x}) - G^{[\hat{L}]}(x) \|_{2 \to 2} \right. \\
+ \left. \left\| G^{[L]}(x) \|_{2 \to 2} \left[ y^{[L-1]}(\tilde{x}) - y^{[L-1]}(x) \right] \| \right\|_2 \\
\lesssim \| W^{[L]} \|_{2 \to 2} \left( \kappa^{[L]}_2 \left( \max_{i_L = 1, \ldots, m^{[L]}} \| a^{[L]}_i \|_2 \right) \left\| y^{[L-1]}(\tilde{x}) - y^{[L-1]}(x) \right\|_2 \right. \\
+ \left. \left\| \frac{\partial}{\partial x_i} y^{[L-1]}(\tilde{x}) - \frac{\partial}{\partial x_i} y^{[L-1]}(x) \right\|_2 \right. \\
\lesssim \| W^{[L]} \|_{2 \to 2} \left( \kappa^{[L]}_2 \left( \max_{i_L = 1, \ldots, m^{[L]}} \| a^{[L]}_i \|_2 \right) \left( \prod_{l=1}^{L-1} \kappa^{[l]}_1 \right) \left( \prod_{l=2}^{L-1} \| W^{[l]} \|_F \right) \right. \\
\left. \left( \sum_{i_1=1}^{m^{[1]}} \left( a^{[1]}_{i_1} \right)^2 \right) \epsilon \right)
\]

where we used our intermediate results (S.1) and (S.2) for \( 1 \) and the results (S.3) and (S.4) for \( 2 \). Note that the first term depends only on the \( f \) (via various quantities like
norms of weights or size of derivates). The previous steps can be repeated in a recursive manner for \( L - 1, \tilde{L} - 2, \ldots, 2 \), resulting in

\[
\left| \frac{\partial}{\partial x_l} y^{(L)}(\tilde{x}) - \frac{\partial}{\partial x_l} y^{(L)}(x) \right|_2 \\
\leq |W^{[L]}|_{2 \rightarrow 2} \kappa_2^{[L]} \left( \max_{i=1, ..., m^{[L]}} |a^{[L]}_{i}|_2 \right) \left( \prod_{l=1}^{L-1} \kappa_1^{[l]} \right) \left( \frac{\prod_{l=2}^{L-1} \kappa_2^{[l]}}{\prod_{l=2}^{L-1} \kappa_1^{[l]}} \right)^{2} \left( \sum_{i_1=1}^{m^{[1]}} \left( a^{[1]}_{i_1} \right)_2^2 \right) \epsilon \\
+ \|W^{[L]}\|_{2 \rightarrow 2} \kappa_1^{[L]} \left( \max_{i=1, ..., m^{[L]}} \kappa_2^{[L]} \right) \left( \prod_{l=1}^{L-1} \kappa_1^{[l]} \right) \left( \prod_{l=2}^{L-1} \kappa_2^{[l]} \kappa_1^{[l-1]} \right) \left( \sum_{i_1=1}^{m^{[1]}} \left( a^{[1]}_{i_1} \right)_2^2 \right) \epsilon \\
+ \|W^{[L]}\|_{2 \rightarrow 2} \kappa_1^{[L]} \left( \prod_{l=1}^{L-1} \kappa_1^{[l]} \right) \left( \prod_{l=2}^{L-1} \kappa_2^{[l]} \kappa_1^{[l-1]} \right) \left( \sum_{i_1=1}^{m^{[1]}} \left( a^{[1]}_{i_1} \right)_2^2 \right) \epsilon \\
\leq \sum_{l=2}^{L} \left( \prod_{l=1}^{L} \kappa_1^{[l]} \right) \left( \prod_{l=1}^{L} \kappa_2^{[l]} \right) \left( \prod_{l=1}^{L} \kappa_1^{[l]} \right) \left( \prod_{l=2}^{L-1} \kappa_2^{[l]} \kappa_1^{[l-1]} \right) \left( \sum_{i_1=1}^{m^{[1]}} \left( a^{[1]}_{i_1} \right)_2^2 \right) \epsilon \\
+ \sum_{l=2}^{L} \left( \prod_{l=1}^{L} \kappa_1^{[l]} \right) \left( \prod_{l=1}^{L} \kappa_2^{[l]} \right) \frac{\partial}{\partial x_l} y^{(1)}(\tilde{x}) - \frac{\partial}{\partial x_l} y^{(1)}(x) \right|_2 \\
\leq \sum_{l=2}^{L} \left( \prod_{l=1}^{L} \kappa_1^{[l]} \right) \left( \prod_{l=1}^{L} \kappa_2^{[l]} \right) \left( \prod_{l=1}^{L} \kappa_1^{[l]} \right) \left( \prod_{l=2}^{L-1} \kappa_2^{[l]} \kappa_1^{[l-1]} \right) \left( \sum_{i_1=1}^{m^{[1]}} \left( a^{[1]}_{i_1} \right)_2^2 \right) \epsilon \\
+ \left( \prod_{l=2}^{L} \kappa_1^{[l]} \right) \left( \prod_{l=1}^{L} \kappa_2^{[l]} \right) \left( \sum_{i_1=1}^{m^{[1]}} \left( a^{[1]}_{i_1} \right)_2^2 \right) \epsilon \\
\text{where we used the bound for layer 1 in the last step.}
\]

Similarly to Lemma S.27 if \( \|a^{[l]}_{i_l}\|_2 \leq 2 \) for all layers the result can be considerably simplified.

**Corollary S.32.** Consider the situation of Lemma S.31.

1. If all weights have Euclidean norm less or equal 1, i.e. \( \|a^{[l]}_{i_l}\|_2 \leq 1 \) for all \( i_l = 1, \ldots, m^{[l]} \) and \( l = 1, \ldots, L \), then we have

\[
\left| \frac{\partial}{\partial x_l} y^{(L)}(\tilde{x}) - \frac{\partial}{\partial x_l} y^{(L)}(x) \right|_2 \leq C_i^{[S.32]} \epsilon
\]

with

\[
C_i^{[S.32]} = \left( \prod_{i_1=1}^{m^{[1]}} \left( a^{[1]}_{i_1} \right)_2 \right) \left( \prod_{l=2}^{L} \sqrt{m^{[l]}} \right) \sum_{l=1}^{L} \sqrt{m^{[l-1]} \kappa_2^{[l]} \kappa_1^{[l-1]} \prod_{i_1=1}^{m^{[1]}} \left( \kappa_1^{[1]} \right)^4}
\]

109
2. If in addition the weights in layers \( l = 2, \ldots, \tilde{L} \) are \( \epsilon_{q_0} \)-nearly orthogonal, then we have
\[
\left\| \frac{\partial}{\partial x_i} y^{[L]}(\tilde{x}) - \frac{\partial}{\partial x_i} y^{[L]}(\bar{x}) \right\|_2 \leq C_i^{[S,32]} \epsilon
\]
with
\[
C_i^{[S,32]} = \left( \sum_{i=1}^{m^{[1]}} \left( a_{i_1}^{[1]} \right)^2 \right) \prod_{l=2}^{\tilde{L}} \left( \prod_{i=1}^{\tilde{L}-1} (1 + \epsilon_{q_0}) \right) \left( \prod_{l=2}^{\tilde{L}} \sqrt{m^{[l]}} \right) \left( \prod_{l=2}^{\tilde{L}} \epsilon_{1,1}^{[l]} \right) \left( \prod_{l=2}^{\tilde{L}} \epsilon_{1}^{[l]} \right)
\]
\[
+ \left( \prod_{l=2}^{\tilde{L}} (1 + \epsilon_{q_0}) \right) \left( \prod_{l=2}^{\tilde{L}} \epsilon_{1}^{[l]} \right) \left( \prod_{l=1}^{\tilde{L}} \left( \sum_{i=1}^{m^{[1]}} \left( a_{i_1}^{[1]} \right)^4 \right) \right)
\]

**Proof.** 1. Follows immediately from Lemma S.31 since \( \| a_i^{[l]} \|_2 \leq 1 \) and \( \| W^{[l]} \|_F \leq \sqrt{m^{[l]}} \) for all \( l = 2, \ldots, \tilde{L} \).

2. Same situation as in the previous case, just use in addition Proposition S.11 (i).

\( \square \)

**Remark S.33.** 1. The bound derived in the preceding lemma has the form
\[
\left\| \frac{\partial}{\partial x_i} y^{[L]}(\tilde{x}) - \frac{\partial}{\partial x_i} y^{[L]}(\bar{x}) \right\|_2 \leq C^{[S,31]} \epsilon,
\]
where \( C^{[S,31]} \epsilon \) depends only on \( f \). Holding \( f \) constant we find that
\[
\left\| \frac{\partial}{\partial x_i} y^{[L]}(\tilde{x}) - \frac{\partial}{\partial x_i} y^{[L]}(\bar{x}) \right\|_2 \to 0
\]
for \( \epsilon \to 0 \).

2. Unfortunately \( C^{[S,31]} \epsilon \) contains terms of the form \( \prod_{l=L_0}^{L_1} \sqrt{m^{[l]}} \) for certain \( 1 \leq L_0 < L_1 \leq \tilde{L} \). This is to be expected because intuitively small changes in the input might be amplified while being propagated through the network and hence finding good hard bounds on the magnitude of the derivates is very difficult, unless one has additional knowledge about the behaviour of the network. Intuitively, a rough measure of how strongly input changes are amplified should be provided by the maximum absolute values of the derivates of the activation functions. Indeed, note that the term \( \prod_{l=2}^{\tilde{L}} \sqrt{m^{[l]}} \) occurs in a product together with \( \prod_{l=2}^{\tilde{L}} \epsilon_{1}^{[l]} \). If the latter factor is small (e.g. if all activation functions are rather flat or some activation functions are steep but some layers have only rather flat activation functions) then the bound might still be rather small.

3. Problems with the magnitude of derivates occur also in backpropagation (though here the derivates are usually with respect to the weights) for deep or recurrent networks. This phenomenon is usually called vanishing gradient problem or exploding gradient problem and indeed posed a considerable practical challenge for training neural networks, see [53], [16] and [54], [76] for more discussions and a modern perspective.
4. In order to bound \( \| \frac{\partial^2}{\partial x_i \partial y^{[l]}(\hat{x})} \|_2 \) for \( l = \hat{L} - 1, \ldots, 1 \) we used \( \| \cdot \|_2 \leq \sqrt{m[l]} \| \cdot \|_\infty \) for \( l = 1, \ldots, \hat{L} \). Going from \( \| \cdot \|_\infty \) to \( \| \cdot \|_2 \) in this manner can lead to very loose bounds (e.g. if all but one of components are zero), however, in general this bound is tight (this happens if all components are equal) and hence without additional knowledge about the behaviour of the network we cannot sharpen this step.

S.4. Guide to notation

Since this work is based on [82] and most results therein are contained here as special cases we provide a short dictionary between our notation (right hand side) and the notation used in [82] (left hand side).

**Network architecture**

**Widths** \( d = m^{[0]}, m = m^{[1]}, m_1 = m^{[2]} \)

**Weights** \( a_1, \ldots, a_m = a^{[1]}_1, \ldots, a^{[m]}_m, b_1, \ldots, b_{m_1} = a^{[2]}_1, \ldots, a^{[m_2]}_m \)

**Weight matrices** \( A = W^{[1]T}, B = W^{[2]T} \)

**Activation functions** \( g_{i_1} = g^{[1]}_{i_1}, h_{i_2} = g^{[2]}_{i_2} \)

**Activation function shortcuts** \( g(y) = \begin{pmatrix} g_1(y_1) \\ \vdots \\ g_m(y_m) \end{pmatrix}, h(z) = \begin{pmatrix} h_1(z_1) \\ \vdots \\ h_{m_1}(z_{m_1}) \end{pmatrix} = g^{[2]}(z^{[2]}) \)

**Elements related to derivatives**

**Maxima of activation functions** \( \kappa_n = \kappa^{[1]}_n, \eta_n = \kappa^{[2]}_n \)

**Activation function matrices** \( G_x = G^{[1]}(x) \)

**Components of Hessian**

\[
\begin{align*}
 v_{x,l} &= AG_x b_l = v^{[2]}_l = W^{[1]T} G^{[1]}(x) a^{[2]}_l \\
 s_{x,lt} &= h^{[n]}_l(b^T_l g(A^T x)) = s^{[2]}_l(x) = (g^{[2]})^{[n]}(z^{[2]}(x)) \\
 S_x &= \text{diag}(s_{x,lt}, \ell = 1, \ldots, m_1) = S^{[2]}(x) = \text{diag}(s^{[2]}_l(x), i_2 = 1, \ldots, m^{[2]}) \\
 t_{x,ii}(x) &= [Bh'(B^T g(A^T x))]_i g''(a^{[1]}_i x) = s^{[1]}_i(x) \\
 T_x &= \text{diag}(t_{x,ii}, i = 1, \ldots, m) = S^{[1]}(x) = \text{diag}(s^{[1]}_i(x), i_1 = 1, \ldots, m^{[1]})
\end{align*}
\]

**Concentration of Hessians**
Vectors and spaces

\[ v_l = v_{0,l} = A G_{0,b_l} = v_i^{[2]} = v_i^{[2]}(0) \]

\[ L = \text{span}_{\mathbb{R}^{d*4}} \{ a_i^{[2]}, v_i^{[2]} \mid i = 1, \ldots, m, l = 1, \ldots, m_1 \} \]

\[ = M = \text{span}_{\mathbb{R}^{m[l] \times m[l]}} \{ v_i^{[l]} \otimes v_i^{[l]} \mid i_l = 1, \ldots, m[l], l = 1, 2 \} \]

\[ L_{\text{vec}} = \text{span}_{\mathbb{R}^2} \{ \text{vec}(a_i^{[2]}), \text{vec}(v_i^{[2]}) \mid i = 1, \ldots, m, l = 1, \ldots, m_1 \} \]

\[ = M = \text{span}_{\mathbb{R}^{m[l] \times 2}} \{ \text{vec}(v_i^{[l]} \otimes v_i^{[l]}) \mid i_l = 1, \ldots, m[l], l = 1, 2 \} \]

Components of algorithm

\[ M = \left( \text{vec}(H[f](x_1)) \cdots \text{vec}(H[f](x_{m_x})) \right) = Y \in \mathbb{R}^{d_x \times m_x} (= \mathbb{R}^{m[l] \times N}) \]

\[ \tilde{M} = \left( \text{vec}(P_L H[f](x_1)) \cdots \text{vec}(P_L H[f](x_{m_x})) \right) \]

\[ = \tilde{Y} = \left( \text{vec}(P_M H[f](x_1)) \cdots \text{vec}(P_M H[f](x_{m_x})) \right) \in \mathbb{R}^{d_x \times m_x} (= \mathbb{R}^{m[l] \times N}) \]

Temporary quantities \( \Delta_x = \Pi^{[1,1]}(x) - \Pi^{[1,1]}(0) \)
Bibliography


118


