Robustness and Invariance in the Generalization Error of Deep Neural Networks

Jure Sokolić

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

of

University College London (UCL).

Department of Electronic and Electrical Engineering

University College London (UCL)

December 19, 2017

I, Jure Sokolić, confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the work.

Abstract

In recent years Deep Neural Networks (DNNs) have achieved state-of-the-art results in many fields such as speech recognition, computer vision and others. Despite their success in practice, many theoretical fundamentals of DNNs are still not clear. One of them is the generalization error of DNNs, which is the topic of this thesis.

The thesis first reviews the theory and practice of DNNs focusing specifically on theoretical results that provide generalization error bounds. We argue that the current state-of-the-art theoretical results, which rely on the width and depth of deep neural networks, do not apply in many practical scenarios where the networks are very wide or very deep.

A novel approach to the theoretical analysis of the generalization error of DNNs is proposed next. The proposed approach relies on the classification margin of the DNN and on the complexity of the data. As this result does not rely on the width or the depth of the network it provides a rationale behind the practical success of learning with very wide and deep neural networks. These results are then extended to learning problems where symmetries are present in the data. The analysis shows that if a DNN is invariant to such symmetries its generalization error may be much smaller than the generalization error of a non-invariant DNN.

Finally, two novel regularization methods for DNNs motivated by the theoretical analysis are presented and their performance is evaluated on various datasets such as MNIST, CIFAR-10, ImageNet and LaRED. The thesis is concluded by a summary of contributions and discussion of possible extensions of the current work.

Impact

This thesis studies the fundamentals of learning with deep neural networks. It proposes to view deep neural networks as large margin classifiers and explains how invariance of deep neural networks to symmetries in the data leads to a lower generalization error. The insights from the analysis in this thesis were used to help develop two regularizers that improve the classification margin and increase the invariance of deep neural networks.

Our results will have impact on both theory and practice of deep neural networks. In particular, the theoretical results offer novel insights into success of deep neural networks and open various promising research directions. For example, our approach emphasizes the role of data complexity and classification margin, which is different from many other established approaches in the field of deep neural networks and will promote discussion and possibly other advances in the field. Moreover, our approach naturally explains the role of invariance in the generalization of deep neural networks and the insights which might be used by practitioners to develop better deep neural network architectures.

On the practical side, our proposed regularizers, which improve performance of deep neural networks when the training set is small, are important in scenarios when obtaining large training datasets is impossible or too expensive, as for example in medical imaging. Our proposed techniques will lead to wider adoption of deep neural networks even in such cases.

Acknowledgements

First, I would like to thank my supervisor Dr. Miguel Rodrigues for his mentorship. I appreciate his support and patience, especially in the most challenging and stressful periods of my PhD. I would also like to acknowledge his guidance and contributions to my research that resulted in this thesis and many other works. He also played a crucial role in establishing successful collaborations with researchers around the world.

The work in this thesis benefited greatly from collaboration with Prof. Guillermo Sapiro and Dr. Raja Giryes. I would like to thank them for many discussions and advice, and for the time they invested in my research.

I would also like to thank Prof. Guillermo Sapiro and Prof. Robert Calderbank for hosting me at Duke University. I would like to acknowledge University College London for financial support and The Vest Scholarship Program that supported my visit to Duke University. Finally, I would also like to thank my family and my girlfriend for their support throughout my PhD.

Contents

List of Figures

List of Tables

List of Symbols

General Mathematical Symbols

a	Scalar
$\lvert a \rvert$	Absolute value of a
$\mathbb{1}(a)$	Indicator function of a
$\mathbf{a} = [a_1, a_2, \ldots, a_N]^T$	Column vector with N elements
$(\mathbf{a})_i = a_i$	The i-th element of the vector $\mathbf{a}$
$(\mathbf{a})_{i:j} =$ $[a_i, a_{i+1}, \ldots, a_j]^T$	Sub-vector of $\mathbf{a}$
$\mathbf{A}$	Matrix
$(\mathbf{A})_{ij}$	The i-th row and j-th column element of $\mathbf{A}$
$\mathrm{diag}(\mathbf{A})$	The diagonal of $\mathbf{A}$
$\lVert \mathbf{a} \rVert_\infty$	The infinity norm of $\mathbf{a}$
$\lVert \mathbf{a} \rVert_2$	The Euclidean norm of $\mathbf{a}$
$\lVert \mathbf{A} \rVert_2$	The spectral norm of $\mathbf{A}$
$\lVert \mathbf{A} \rVert_F$	The Frobenius norm of $\mathbf{A}$
$\mathcal{A}$	Set
$\lvert \mathcal{A} \rvert$	Cardinality of $\mathcal{A}$
$\mathsf{conv}(\mathcal{A})$	Convex hull of $\mathcal{A}$
$\mathbb{N}$	The set of natural numbers
$\mathbb{R}$	The set of real numbers
$\mathbb{R}^N$	N-dimensional real vector space
$\mathbb{R}^{M \times N}$	$M \times N$-dimensional real matrix space

$\mathbf{I}$	The identity matrix
$\mathbf{I}_N$	The identity matrix in $\mathbb{R}^{N \times N}$
$\mathbf{0}$	The zero matrix
$\mathbf{0}_{M \times N}$	The zero matrix in $\in \mathbb{R}^{M \times N}$
$\mathbf{1}$	The column vector of ones
$\mathbf{1}_N$	The column vector of ones in $\mathbb{R}^N$
$\mathbf{e}_k$	The k-th basis vector of the standard basis in $\mathbb{R}^N$
$\mathcal{N}(\mathcal{X}; d, \rho)$	The covering number of $\mathcal{X}$ with d-metric balls of radius ρ
$\mathcal{O}$	Asymptotic notation
$\mathbb{E}_x$	Expected value with respect to the random variable x
sup	Supremum
inf	Infimum
max	Maximum
min	Minimum
arg max	Argument of the maximum
arg min	Argument of the minimum

Other Symbols

$\mathbf{x}$	Input signal
$\mathcal{X}$	Input space
y	Class label
$\mathcal{Y}$	Label space
$N_\mathcal{Y}$	Number of classes
$\mathcal{S} = \mathcal{X} \times \mathcal{Y}$	Sample space
$s = (\mathbf{x}, y)$	Training sample
m	Number of training samples
$S_m = \{s_i\}_{i=1}^m$	Training set
$\mathcal{T}$	Set of input transformations
t	Input transformation

g	Classifier
ℓ	Loss function
GE	Generalization error
γ	Classification margin
o	Training sample score
$\mathbf{J}$	Jacobian matrix
f	DNN
L	Number of DNN layers
ϕ_l	l-th DNN layer
$\mathbf{z}^l$	Output of the l-th DNN layer
θ_l	Parameters of the l-th DNN layer
$\mathbf{W}_l$	Weight matrix of the l-th DNN layer
$\mathbf{b}_l$	Bias vector of the l-th DNN layer
σ	DNN non-linearity
Θ	Set of all DNN parameters
$\mathcal{L}$	Objective function
R	Regularizer
$\mathbf{N}$	Batch normalization matrix
$\mathbf{n}$	Batch normalization mean vector

List of Abbreviations

CNN	Convolutional Neural Network
CCE	Categorical Cross Entropy
DL	Deep Learning
DNN	Deep Neural Network
ERM	Empirical Risk Minimizer
GE	Generalization Error
GPU	Graphical Processing Unit
IID	Independent and Identically Distributed
JM	Jacobian Matrix
LSTM	Long-Short Term Memory
RC	Rademacher Complexity
RNN	Recurrent Neural Network
SGD	Stochastic Gradient Descent
SVM	Support Vector Machine
VC	Vapnik-Cervonenkis

Chapter 1

Introduction

We live in the age of data. For example, smart-phones are very powerful data collection devices with various sensors such as camera, GPS and acceleration sensor. The data is also collected on the internet, where online companies keep records of users' transactions and behaviour. In medicine, electronic healthcare records are being used as a standardized tool for collection and storage of patient data. Factories use various sensors to monitor their manifacturing equipment, and autonomous vehicles collect data about their surroundings with multiple cameras and LIDAR sensors [1].

However, the data is not exciting because of the data itself, but rather because we may leverage it to obtain novel scientific discoveries, or introduce novel products or services. For example, online shops use recommendation engines that leverage users' shopping history to offer product recommendations to new users. Online advertising is also using users' data to serve them more relevant advertisements. Electronic health records might be used in large scale medical studies or they can be used to build data-powered tools that help doctors to better diagnose their patients. The manufacturing equipment data can be used to offer predictive maintenance – a service where the issues with the equipment are identified before they actually occur. Finally, autonomous cars, which require full understanding of their environment, leverage data to build better models of the environment [1].

One of the dominant disciplines dealing with the design of algorithms that leverage data is machine learning, which also intersects with statistics, learning theory, information theory, signal processing and computer science [2]. The goal of machine learning is to build computational models that leverage data to learn and make predictions. This goal is broad and includes everything from unsupervised learning, where one usually

looks for a simpler representation of the original data, to supervised learning, where the goal is to learn predictive models that, given as input a set of features, can reliably predict the quantity of interest.

The conventional approach to machine learning usually includes the so-called feature extraction [3]. For example, raw data, such as text, speech audio or an image, is not fed directly to a machine learning model, but features extracted from this data are. The goal of feature extraction is to reduce the dimensions of the data while preserving the information content. These features are usually hand designed and leverage domain expertise.

However, in the recent years, a paradigm shift has occurred in the community with a great success of Deep Learning (DL) and Deep Neural Networks (DNNs) [3]. In contrast to conventional methods, in DL feature extraction is replaced by a DNN module, which can be interpreted as a hierarchical feature extractor and it is trained simultaneously with a classifier or a regressor at the output of the network. This approach has delivered significant gains in performance of speech recognition systems, it offers state-of-the-art performance on many computer vision tasks and has also been successfully applied in computational chemistry, physics and many other fields [3].

A DNN is a machine learning model inspired by the neurons in the human brain [2]. A single layer of a DNN computes features of the input via a non-linear transformations. A neural networks is *deep* when several such layers are stacked one after another. The rationale here is that features computed at higher layers give a more abstract description of the underlying signal [4]. The parameters of each layer are estimated from the training data. For example, in case of image classification, training of a DNN works as follows: images are passed through a network and a prediction is produced at the network output. Then the error between the predicted and the correct labels is computed. The training algorithm adopts the parameters of all the layers in such a way that the average error of all the examples in the training set is minimized [5].

The DL approach is very successful in practice, but at the same time also challenges our current understanding of machine learning fundamentals. DNNs usually have a very large number of parameters and their objective function is non-convex [3, 6]. There are many unanswered questions, for example:

- Why can DNNs be trained successfully in practice given that their optimization

is a non-convex problem?

- Why do DNNs not over-fit when in practice the number of training samples is often smaller than the number of network parameters?
- Can we train DNNs with very small training sets?

This thesis aims to provide some answers to these questions.

1.1 Aim and Motivation

Our motivation to study DNNs comes from their great success in practice and from many open questions about their fundamentals [3, 6]. We are now going discuss the motivations and aims for this thesis more specifically.

One of the most important properties of a learning algorithm is its ability to generalize to new data. In particular, models are usually trained by minimization of the average error on the training set. However, what we really want is to have a small error (on average) on all the samples that we will see in the future. The difference between the two is called the Generalization Error (GE). The GE is large when the trained model relies too much on features of the data specific to particular training examples that are in fact not relevant for the task that we are trying to solve. We also say that the model suffers from *over-fitting* to the training data [7].

Clearly, the success of DNNs in practice indicates that they do generalize well. However, large training sets are often needed to achieve such good performance. And in many domains, such as medical research, obtaining large datasets may not be possible due to various ethical or cost constraints. Therefore, it would be beneficial to understand the mechanism behind successful generalization of DNNs, and improve their training methods so that training with smaller training sets will also lead to DNNs with good generalization.

Moreover, as it will be further discussed in this thesis, DNNs seem to contradict many established theoretical justifications for good generalization of learning algorithms. For example, one may establish a link between the ability of the machine learning model to fit noise and its generalization properties. In particular, a more constrained model that has a lesser ability of fitting noise is more likely to generalize well [6]. However, DNNs in practice are very powerful and are able to fit noise, yet they also

generalize well. Therefore, the fundamental question of why DNNs generalize well is also interesting from theoretical perspective, which we partially address in this thesis.

1.2 Main Contributions

The main contributions of the thesis can be summarized as follows: i) New GE bounds for DNN classifiers are derived. In contrast to other GE bounds in the literature that are dominated by either DNN width or depth, the proposed bounds are dominated by the complexity of the data and hold for DNNs of any size or depth; ii) A novel framework for bounding the GE of DNNs is presented that allows to quantify how the data symmetries and invariance of DNNs to such symmetries in the data may reduce the GE; iii) The theoretical insights are extended to practice by introducing novel network regularization methods for DNNs that lead to a better GE.

A detailed list of contributions is provided below.

- We propose novel GE bounds for DNN classifiers that are based on their classification margin, i.e. the distance between the training sample and a non-linear decision boundary induced by a DNN classifier in the sample space, and on the complexity of the data, which is measured via the covering number.

- The network's Jacobian Matrix (JM) is used to bound the classification margin. The consequence of this is that the derived bounds apply to any DNN for which a JM is defined. Informally, the method applies to all DNNs that can be trained by a gradient descent optimization method (as their gradients, and consequently the JM, is defined).

- In contrast to other works on the GE of DNNs, our analysis shows that the GE of a DNN can be bounded independently of its depth or width provided that the spectral norm of the JM in the neighbourhood of the training samples is bounded. We argue that this result gives a justification for a low GE of very deep or wide DNNs in practice. Moreover, recently proposed DNN re-parametrization techniques such as weight normalization and batch normalization that are used in the state-of-the-art DNNs may be analysed within our framework.

- The analysis also leads to a novel Jacobian regularizer. We provide a series of

experiments on the MNIST, CIFAR-10, LaRED and ImageNet datasets that validate our analysis and demonstrate the effectiveness of the Jacobian regularizer.

- We extend the GE bounds framework to include the notion of data symmetries and DNN invariance. In particular, we show that if data has certain symmetries, then a DNN classifier that is invariant to these symmetries has a much smaller GE than its non-invariant counterpart. The theoretical results are supported by experiments on the MNIST dataset.

- Based on the developed theory we also propose the invariance regularizer that enforces invariance of a DNN and consequently leads to a lower GE. The effectiveness of the invariance regularizer is demonstrated on the CIFAR-10 datasets.

1.3 Publications and Talks

The work presented in this thesis has been published in a journal and in various conferences proceedings. It has also been presented at various conferences, as listed below. We also include a list of other publications produced during the PhD program.

1.3.1 Publications and Talks Related to this Thesis

Publications

1. Jure Sokolić, Raja Giryes, Guillermo Sapiro, and Miguel R D Rodrigues. Robust large margin deep neural networks. *IEEE Transactions on Signal Processing*, 65(16):4265–4280, 2017.

2. Jure Sokolić, Raja Giryes, Guillermo Sapiro, and Miguel R D Rodrigues. Generalization error of invariant classifiers. *Conference on Artificial Intelligence and Statistics (AISTATS)*, 1094–1103, 2017.

3. Jure Sokolić, Raja Giryes, Guillermo Sapiro, and Miguel R D Rodrigues. Generalization error of deep neural networks: Role of classification margin and data structure. *International Conference on Sampling Theory and Applications (SampTA) (to appear)*, 2017.

Talks

- *Margin preservation of deep neural networks*. Theory of Deep Learning workshop, International Conference on Machine Learning (ICML), June 2016, New York, USA.

- *Generalization error of deep neural networks: classification margin and invariance*. Deep Neural Networks: Theory and Applications mini-symposia, Applied Inverse Problems (AIP) conference, May 2017, Hangzhou, China.

1.3.2 Other Publications

4. Jure Sokolić, Majid Zamani, Andreas Demosthenous, and Miguel R D Rodrigues. A feature design framework for hardware efficient neural spike sorting. *International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, 1516–1519, 2015.

5. Jure Sokolić, Francesco Renna, Robert Calderbank, and Miguel R D Rodrigues. Mismatch in the classification of linear subspaces: Upper bound to the probability of error. *IEEE International Symposium on Information Theory (ISIT)*, 2201–2205, 2015.

6. Jure Sokolić, Francesco Renna, Robert Calderbank, and Miguel R D Rodrigues. Mismatch in the classification of linear subspaces: Sufficient conditions for reliable classification. *IEEE Transactions on Signal Processing*, 64(12):3035–3050, 2016.

7. Jure Sokolić, Qiang Qiu, Miguel R D Rodrigues, and Guillermo Sapiro. Learning to Identify while Failing to Discriminate. *International Workshop on Cross-domain Human Identification at International Conference on Computer Vision (ICCV) (to appear)*, 2017.

1.4 Thesis Organization

We now describe the thesis structure.

In Chapter 2 we provide an overview of the statistical learning theory and techniques used to bound the GE, and an overview of DNNs and their properties. A particular focus of Chapter 2 are the existing GE bounds for DNNs, which do not apply to modern very deep or wide DNNs. The first contribution that establishes new GE bounds based on the classification margin of DNNs and the covering number of the sample space are presented in Chapter 3. The GE bounds presented in Chapter 3 are then extended in Chapter 4 to a scenario where data possesses certain symmetries and a DNN is invariant to these symmetries. Chapter 5 presents two regularizers: the Jacobian regularizer and the invariance regularizer, that are motivated by the theory developed in Chapters 3 and 4. The two regularizers are evaluated on various image classification datasets. Chapter 6 includes summary of the contributions, conclusions and discussion of possible future work. Proofs related to Chapter 3 are provided in Appendix A and proofs related to Chapter 4 are provided in Appendix B.

Chapter 2

Statistical Learning and Deep Neural Networks

In this chapter we first formally define the Generalization Error (GE) in the framework of statistical learning and also review various tools for bounding the GE of machine learning algorithms. We then present the Deep Neural Network (DNN) model used in this thesis and review the relevant literature. We conclude the chapter by a careful examination of the literature that studies the GE in the context of DNNs and expose some open problems.

2.1 Statistical Learning and Generalization Error

Consider a classification problem where we want to learn a classifier that will associate a signal, e.g., an image of a digit, with a label, e.g., the value of the digit in the image. The classifier is trained on a training set that consists of signal and label pairs. Note that the statistical learning theory considers a much wider set of problems, including problems and unsupervised or semi-supervised learning [2, 15]. However, since we focus on DNNs applied to classification problems, we will only discuss the classification problem in detail.

Formally, we observe a vector $\mathbf{x} \in \mathcal{X} \subseteq \mathbb{R}^N$ that has a corresponding class label $y \in \mathcal{Y}$. The set $\mathcal{X}$ is called the input space and it is a subset of $\mathbb{R}^N$. The set $\mathcal{Y} = \{1, 2, \ldots, N_\mathcal{Y}\}$ is called the label space and $N_\mathcal{Y}$ denotes the number of classes. For example, $\mathbf{x}$ may represent an image and y may represent the label of the object in the image that we want to recognize. The product of the input space and the label space is called the sample space and it is denoted by $\mathcal{S} = \mathcal{X} \times \mathcal{Y}$. An element of $\mathcal{S}$

is denoted by $s = (\mathbf{x}, y)$. We assume that samples from $\mathcal{S}$ are drawn according to a probability distribution P, which is defined on $\mathcal{S}$. A training set of m samples, which are Independent and Identically Distributed (IID) and drawn from P, is denoted by

$$S_m = \{s_i\}_{i=1}^m = \{(\mathbf{x}_i, y_i)\}_{i=1}^m . \tag{2.1}$$

The goal of learning is to leverage the training set S_m to find a classifier $g(\mathbf{x})$ that provides a label estimate $\hat{y}$ given the input $\mathbf{x}$:

$$\hat{y} = g(\mathbf{x}) . \tag{2.2}$$

The success of the classifier is measured by a loss function

$$\ell(g(\mathbf{x}), y) \in \mathbb{R} , \tag{2.3}$$

which measures the discrepancy between the true label y and the estimated label $\hat{y} = g(\mathbf{x})$ provided by the classifier. In general, the larger the loss $\ell(g(\mathbf{x}), y)$, the larger the discrepancy between the classifier output and the true label. The most appropriate loss for classification problems is the 0-1 loss:

$$\ell_{0\text{-}1}(g(\mathbf{x}), y) = \mathbb{1}(g(\mathbf{x}) \neq y) , \tag{2.4}$$

where $\mathbb{1}$ is the indicator function, i.e., $\mathbb{1}(g(\mathbf{x}) \neq y)$ outputs 1 if the prediction is correct, and 0 if the prediction is incorrect. We also assume that in this case the output of the classifier is a discrete label in $\mathcal{Y}$, i.e., $g(\mathbf{x}) \in \{1, 2, \ldots, N_{\mathcal{Y}}\}$. In practice, we may use losses that are differentiable and are more suitable for optimization via gradient descent algorithms, such as the hinge loss, which in the case of a binary classification problem is defined as

$$\ell_{\mathrm{H}}(g(\mathbf{x}), y) = \max(0, 1 - y g(\mathbf{x})) , \tag{2.5}$$

where $y \in \{-1, 1\}$ and $g(\mathbf{x}) \in \mathbb{R}$, or the Categorical Cross Entropy (CCE) loss, which

in the case of binary classification problem is defined as

$$\ell_{\mathrm{CCE}}(g(\mathbf{x}), y) = y \log(g(\mathbf{x})) + (1-y)\log(1-g(\mathbf{x}))\,, \tag{2.6}$$

where $g(\mathbf{x}) \in [0,1]$ denotes the probability of $y = 1$ and $y \in \{0,1\}$.

The empirical loss of the classifier g associated with the training set S_m is defined as

$$\ell_{\mathrm{emp}}(g) = 1/m \sum_{s_i=(\mathbf{x}_i,y_i)\in S_m} \ell(g(\mathbf{x}_i), y_i)\,. \tag{2.7}$$

We will use the empirical loss to train the classifier. However, we are actually interested in the expected loss of the classifier g, which is defined as

$$\ell_{\mathrm{exp}}(g) = \mathbb{E}_{s=(\mathbf{x},y)\sim P}\left[\ell(g(\mathbf{x}), y)\right], \tag{2.8}$$

where $\mathbb{E}_{s=(\mathbf{x},y)\sim P}$ denotes expected value with respect to the samples $s = (\mathbf{x}, y)$ drawn from the distribution P. Note that the distribution P is not known and we only access the training set S_m sampled from P. By the Empirical Risk Minimizer (ERM) principle [16] we use a classifier $\hat{g}$ that minimizes the empirical loss:

$$\hat{g} = \arg\min_{g} \ell_{\mathrm{emp}}(g)\,. \tag{2.9}$$

Since we can not estimate the expected loss directly, we may only hope that the empirical loss provides a good estimate for the expected loss. The measure that determines the discrepancy between the empirical loss and the expected loss of a classifier g is called the GE:

$$GE(g) = |\ell_{\mathrm{exp}}(g) - \ell_{\mathrm{emp}}(g)|\,. \tag{2.10}$$

In general, there are no guarantees that the GE will be small. If there are no restrictions on the class of functions that represent our classifier we may find a classifier that fits the training set perfectly – the empirical loss is small, but at the same time does not perform well on the new data – the expected loss is large. This phenomena is also

called over-fitting to training data [7]. Since the GE decreases with the size of the training set, we may also pose the problem in terms of sample complexity. Specifically, the sample complexity corresponds to the size of a training set for which the GE is smaller than a given threshold. Therefore, the notion of GE is analogue to the notion of sample complexity. We review various tools used to bound the GE next.

2.1.1 Generalization Error Bounds

There are two main approaches to bounding the GE: (i) approaches based on a constrained function class of classifiers; and (ii) approaches based on the properties of a learning algorithm:

(i) We may limit the set of possible classifiers to a set $\mathcal{G}$. We can think of $\mathcal{G}$ as a collection of all possible classifiers that we consider when solving the ERM problem in (2.9). To be more precise, we pose the optimization problem as

$$\hat{g} = \underset{g \in \mathcal{G}}{\arg\min}\, \ell_{\text{emp}}(g)\,, \tag{2.11}$$

where compared to (2.9) we have added a constrain $g \in \mathcal{G}$. The GE bounds are then established by measuring the complexity of the hypothesis class $\mathcal{G}$, which may be achieved using the Vapnik-Cervonenkis (VC)-dimension or the Rademacher Complexity (RC). Both the VC-dimension and the RC will be defined and further discussed in Sections 2.1.1.1 and 2.1.1.2.

(ii) On the other hand, approaches based on algorithmic stability or algorithmic robustness do not necessarily measure the complexity of the hypothesis class $\mathcal{G}$, but rather bound the GE based on the behaviour of the learning algorithm. In particular, we can view a learned classifier $\hat{g}$ as an output of a learning algorithm $A(S_m)$, which is a function of the training set S_m:

$$\hat{g} = A(S_m)\,. \tag{2.12}$$

We will briefly review the notions of the VC-dimension, RC, algorithmic stability and algorithmic robustness next. We will then see in Section 2.2.3 how different approaches to bounding the GE lead to different conclusions about the sample com-

plexity of DNNs and we will also discuss how well the theoretical results are aligned with practice.

2.1.1.1 Rademacher Complexity

We start by the definition of the RC [17].

Definition 2.1 (Rademacher Complexity)**.** *Let $\mathcal{F}$ be a class of functions $f : \mathcal{X} \to \mathbb{R}$ that map from the space $\mathcal{X}$ to real numbers, suppose $P_{\mathcal{X}}$ is a probability distribution on $\mathcal{X}$ and $\mathbf{x}_1, \ldots, \mathbf{x}_m$ are IID samples drawn from $P_{\mathcal{X}}$. The empirical RC of the class $\mathcal{F}$ is*

$$\overline{RC}_m(\mathcal{F}; \mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m) = \mathbb{E}\left[\sup_{f \in \mathcal{F}} \sum_{i=1}^{m} \sigma_i f(\mathbf{x}_i)\right], \tag{2.13}$$

where σ_i, $i = 1, \ldots, m$, are IID and drawn from the uniform distribution over $\{-1, 1\}$, $\sup_{f \in \mathcal{F}}$ denotes the supremum over functions $f \in \mathcal{F}$, and the expectation $\mathbb{E}$ is taken over the random variables σ_i, $i = 1, \ldots, m$. The RC is then defined as

$$RC_m(\mathcal{F}) = \mathbb{E}\left[\overline{RC}_m(\mathcal{F}; \mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m)\right], \tag{2.14}$$

where the expectation is taken over samples $\mathbf{x}_1, \ldots, \mathbf{x}_m$.

Intuitively, if a function f represents a binary classifier for inputs $\mathbf{x}_i$, $i = 1, \ldots, m$ and σ_i, $i = 1, \ldots, m$, represent random labels, the RC measures how well the best function in $\mathcal{F}$ can fit these random labels. Clearly, as the labels σ_i, $i = 1, \ldots, m$ are random, we can also interpret this as an ability of the function class $\mathcal{F}$ to fit noise. The following theorem that bounds the GE based on the RC was proposed in [17]:

Theorem 2.1. *Consider a loss function ℓ, such that $\ell(g(\mathbf{x}), y) \in [0, 1]$ and a loss class $\mathcal{L} = \{(\mathbf{x}, y) \mapsto \ell(g(\mathbf{x}), y) : g \in \mathcal{G}\}$, which is a composition of the loss function with the classifiers in $\mathcal{G}$. Then for every $g \in \mathcal{G}$ with probability at least $1 - \delta$*

$$\ell_{exp}(g) - \ell_{emp}(g) \leq RC_m(\mathcal{L}) + \sqrt{\frac{8 \log(2/\delta)}{m}}. \tag{2.15}$$

Clearly, as we increase the number of training samples m, the GE will approach zero if the RC approaches zero. Therefore, if we interpret the RC of a function class as the ability of the functions in this class to fit noise, we require that as the number

of training samples grows the ability to fit random labels vanishes. A related quantity where the variables σ_i, $i = 1, \ldots, m$, have a Gaussian distribution rather than a Rademacher distribution is called Gaussian complexity and has very similar properties to the RC [17].

2.1.1.2 Vapnik-Cervonenkis Dimension

The VC-dimension [16, 18, 19] measures the capacity of a class of functions in a similar way to the RC. Historically, the VC-dimension was used to estabish a principled theory of generalization for Support Vector Machine (SVM) classifiers [20, 21].

The VC-dimension is defined as follows.

Definition 2.2 (VC-dimension)**.** *Take a class of functions $\mathcal{F}$ and assume $f \in \mathcal{F} : \mathbb{R}^N \mapsto \{0, 1\}$. Take m points $\mathbf{x}_1, \ldots, \mathbf{x}_m$, where $\mathbf{x}_i \in \mathbb{R}^N$, $i = 1, \ldots, m$, take $[f(\mathbf{x}_1), \ldots, f(\mathbf{x}_m)]$ to be a vector in $\{0, 1\}^m$, and take $|\{[f(\mathbf{x}_1), \ldots, f(\mathbf{x}_m)] : f \in \mathcal{F}\}|$ to be the cardinality of the set of vectors $[f(\mathbf{x}_1), \ldots, f(\mathbf{x}_m)]$ induced by the class $\mathcal{F}$. Then the shattering coefficient of the class $\mathcal{F}$ is defined as*

$$sh(\mathcal{F}, m) = \max_{\mathbf{x}_i \in \mathbb{R}^N, i=1\ldots,m} |\{[f(\mathbf{x}_1), \ldots, f(\mathbf{x}_m)] : f \in \mathcal{F}\}| . \tag{2.16}$$

The VC-dimension of the class $\mathcal{F}$ is then defined as

$$VC(\mathcal{F}) = \sup\{m : sh(\mathcal{F}, m) = 2^m\} . \tag{2.17}$$

Intuitively, the VC-dimension of a function class $\mathcal{G}$ is the maximum number of samples for which some $g \in \mathcal{G}$ can perfectly memorize the labels. We can see that intuition behind the VC-dimension is similar to the intuition behind the RC.

The GE bounds of the following format can be obtained via the VC-dimension [19]:

Theorem 2.2. *Take a binary classification problem, take $\mathcal{G}$ to be a class of binary classifiers and consider ℓ to be the 0-1 loss. Then, there is an absolute constant C such that for any integer m, where m corresponds to the size of the training set, with probability at least $1 - \delta$, for every $g \in \mathcal{G}$:*

$$\ell_{exp}(g) - \ell_{emp}(g) \leq C\sqrt{\frac{VC(\mathcal{G})}{m}} . \tag{2.18}$$

Clearly, the VC-dimension of $\mathcal{G}$ has to be bounded in order for the classifier in $\mathcal{G}$ to have GE guarantees. In particular, as the number of training samples m grows, at some point the classifiers in $\mathcal{G}$ must not be able to memorize all 2^m possible combinations of binary labels associated with the training samples.

2.1.1.3 Algorithmic Stability

The approaches based on the RC complexity or the VC-dimension bound the GE by constraining the hypothesis class. They do not take into account the role of a particular learning algorithm that is used to obtain a classifier g. The notion of algorithmic stability, proposed by the authors in [22], relates the algorithm A, which is used to obtain the classifier g as in (2.12), to the GE of a classifier g obtained by this algorithm. It is defined as follows.

Definition 2.3 (Algorithmic stability)**.** *Take a training set $S_m = \{s_1, \ldots, s_m\}$ and a modified training set*

$$S_m^i = \{s_1, \ldots, s_{i-1}, s_i', s_{i+1}, \ldots, s_m\}, \tag{2.19}$$

where s_i' is drawn from P and is independent of S_m. An algorithm A has uniformly stability β with respect to the loss function ℓ if for any training set S_m drawn IID from P and for any index $i \in \{1, 2, \ldots, m\}$

$$\|\ell(g_{A(S_m)}(\mathbf{x}), y) - \ell(g_{A(S_m^i)}(\mathbf{x}), y)\|_\infty \leq \beta \quad \forall s = (\mathbf{x}, y) \in \mathcal{S} = \mathcal{X} \times \mathcal{Y}, \tag{2.20}$$

where

$$g_{A(S_m)} = A(S_m) \text{ and } g_{A(S_m^i)} = A(S_m^i). \tag{2.21}$$

The GE bounds of the following form can be established for stable learning algorithms [22]:

Theorem 2.3. *Let A be a learning algorithm with uniform stability β with respect to a loss function ℓ, which is bounded as $0 \leq \ell(g(\mathbf{x}), y) \leq M$. Then for any m, where m is the size of the training set, with probability at least $1-\delta$*

$$\ell_{exp}(g) - \ell_{emp}(g) \leq \beta + (\beta m + M)\sqrt{\frac{2\log(1/\delta)}{m}}, \tag{2.22}$$

where $g = A(S_m)$.

Intuitively, the uniform stability requires that a learning algorithm A behaves similarly for training sets that only differ in a single training example. In other words, the learning algorithm should not rely too much on a particular training example. Moreover, the term βm in (2.22) implies that stability β should scale as $1/m$. Therefore, as the training set grows, the dependence of the learning algorithm A on a particular training example should diminish.

2.1.1.4 Algorithmic Robustness

Similar to the algorithmic stability framework, the algorithmic robustness framework [23] bounds the GE based on the properties of the algorithm A that is used to obtain the classifier as in (2.12). In contrast to the algorithmic stability, where the algorithm output must be similar for similar training sets, the algorithm is robust if the classifier that it outputs has a similar performance for a testing sample and a training sample that are "close".

First, we provide a definition of a robust learning algorithm [23].

Definition 2.4 (Algorithmic robustness)**.** *Let $S_m = \{s_i\}_{i=1}^m = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$ be a training set and $\mathcal{S} = \mathcal{X} \times \mathcal{Y}$ the sample space with samples $s = (\mathbf{x}, y) \in \mathcal{S}$. A learning algorithm A is $(K, \epsilon(S_m))$-robust, where $K \in \mathbb{N}$, $\epsilon(S_m) \in \mathbb{R}$ and $\epsilon(S_m) > 0$, if the sample space $\mathcal{S}$ can be partitioned into K disjoint sets denoted by $\mathcal{K}_k$, $k = 1, \ldots, K$,*

$$\mathcal{K}_k \subseteq \mathcal{S}, \quad k = 1, \ldots, K, \tag{2.23}$$

$$\mathcal{S} = \cup_{k=1}^K \mathcal{K}_k, \tag{2.24}$$

$$\mathcal{K}_k \cap \mathcal{K}_{k'} = \emptyset, \quad \forall k \neq k', \tag{2.25}$$

such that for all $s_i \in S_m$ and all $s \in \mathcal{S}$

$$s_i = (\mathbf{x}_i, y_i) \in \mathcal{K}_k \text{ and } s = (\mathbf{x}, y) \in \mathcal{K}_k \implies$$
$$|\ell(g(\mathbf{x}_i), y_i) - \ell(g(\mathbf{x}), y)| \leq \epsilon(S_m)\,, \tag{2.26}$$

where $g = A(S_m)$.

Note that s_i is an element of the training set S_m and s is an arbitrary element of the sample space $\mathcal{S}$. Therefore, a robust learning algorithm chooses a classifier g for which the losses of any s and s_i in the same partition $\mathcal{K}_k$ are close. The main theorem that establishes a relationship between the robustness and the GE is provided next [23].

Theorem 2.4. *If a learning algorithm A is $(K, \epsilon(S_m))$-robust and $\ell(g(\mathbf{x}), y) \leq M$ for all $s = (\mathbf{x}, y) \in \mathcal{S}$, then for any $\delta > 0$, with probability at least $1 - \delta$,*

$$GE(g) \leq \epsilon(S_m) + M\sqrt{\frac{2K\log(2) + 2\log(1/\delta)}{m}}\,, \tag{2.27}$$

where $g = A(S_m)$.

The first term in the GE bound in (2.27) depends on the training set S_m but does not decrease with the number of training samples m, in general. The second term approaches zero as the number of training samples m grows when other quantities are fixed. $M = 1$ in the case of 0-1 loss, and K corresponds to the number of partitions of the samples space $\mathcal{S}$.

There are several relevant extension of Theorem 2.4 in [23]. In particular, the notion of pseudo-robustness is defined, which requires (2.26) to hold only for a subset of training samples, with the GE bounds based on it. Moreover, the authors show that pseudo-robustness of a learning algorithm is necessary for the resulting classifiers to generalize.

Covering Number

The bounds based on the algorithmic robustness rely on partitioning of the sample space $\mathcal{S}$ into K disjoint partitions. This partitioning can be constructed by covering the samples space by (pseudo-)metric balls of certain radius. The smallest number of balls needed to cover the space is called the covering number [24]:

Definition 2.5 (Covering Number)**.** *Take $\mathcal{M}$ to be a metric space with metric d and let $\mathcal{S} \subset \mathcal{M}$. $\mathcal{S}' \subset \mathcal{M}$ is an ρ-cover of $\mathcal{S}$, if $\forall s \in \mathcal{S}, \exists s' \in \mathcal{S}'$ such that $d(s, s') \leq \rho$. The ρ-covering number of $\mathcal{S}$ is*

$$\mathcal{N}(\mathcal{S}; d, \rho) = \min\{|\mathcal{S}'| : \mathcal{S}' \text{ is an } \rho\text{-cover of } \mathcal{S}\}, \tag{2.28}$$

where $|\mathcal{S}'|$ is the cardinality of $\mathcal{S}'$.

We will leverage the notion of covering number throughout this thesis.

2.2 Deep Neural Networks

The term *neural network* is inspired by the early approaches to mathematical modelling of the information processing in the brain [2, 25, 26]. It encompasses a wide range of mathematical models which are applied in various applications of machine learning and information processing such as speech recognition, computer vision and others.

The term *deep neural network* denotes neural network architectures that have many layers and are, therefore, *deep*. Specifically, a single layer of a neural network is a relatively simple module that computes a non-linear input-output mapping. By stacking multiple such layers we obtain a deep neural network, which leads to a much more powerful model [3, 4, 5].

DNNs are usually trained using the gradient descent optimization and the gradients are computed efficiently by the backpropagation algorithm [3, 25]. For example, given a training objective, the gradient of the objective with respect to the parameters of a particular layer are computed. The parameters are then iteratively updated by taking steps in the negative direction of the gradient. As the network is deep, the computation of the gradient of the objective with respect to the parameters of a particular layer corresponds to "backpropagation" of the objective from the top of the network to the appropriate layer. This general approach of using a DNN and the backpropagation algorithm for training has lead to the state-of-the-art results in image recognition, speech recognition and many other applications [3, 27, 28, 29].

Next, we will review various DNN architectures, with an emphasis on the feed-forward DNNs. We will also review various theoretical works related to DNNs. We will not cover the topics related to the GE of DNNs as this is the topic of Section 2.2.3.

A general overview of DNNs is provided in [30].

2.2.1 Architectures and Learning

We now define the feed-forward DNN, the Convolutional Neural Network (CNN) and review various related architectures. We also discuss training and regularization of DNNs.

2.2.1.1 Feed-Forward Deep Neural Network

In this work we consider a feed-forward DNN architecture, where a DNN, which is denoted by f, maps from an input space $\mathcal{X}$ to a feature space $\mathcal{Z}$:

$$f : \mathcal{X} \to \mathcal{Z}\,. \tag{2.29}$$

We will assume that the input space is a subset of the real N dimensional space $\mathbb{R}^N$: $\mathcal{X} \subseteq \mathbb{R}^N$. In the case of a classification problem with $N_\mathcal{Y}$ classes we will assume that the feature space is the real $N_\mathcal{Y}$ dimensional space: $\mathcal{Z} = \mathbb{R}^{N_\mathcal{Y}}$. In particular, a L layer DNN is obtained as

$$f(\mathbf{x}) = \phi_L(\phi_{L-1}(\cdots\phi_1(\mathbf{x},\theta_1),\cdots\theta_{L-1}),\theta_L)\,, \tag{2.30}$$

where $\phi_l(\cdot,\theta_l)$ represents the output of the l-th layer of a DNN with parameters θ_l:

$$\mathbf{z}^l = \phi_l(\mathbf{z}^{l-1},\theta_l)\,, \tag{2.31}$$

and $\mathbf{z}^l \in \mathbb{R}^{M_l}$ is the output of the l-th layer and $\mathbf{z}^{l-1} \in \mathbb{R}^{M_{l-1}}$ is the input of the l-th layer, $l = 1,\ldots,L$. The input of the first layer is denoted by $\mathbf{z}^0$ and corresponds to $\mathbf{x}$, i.e., $\mathbf{z}^0 = \mathbf{x}$. The output of the last layer is denoted by $\mathbf{z} = f(\mathbf{x})$. Such a DNN is visualized in Figure 2.1.

A typical network layer is a combination of an affine transformation and an element-wise non-linearity:

$$\mathbf{z}^l = [\hat{\mathbf{z}}^l]_\sigma = [\mathbf{W}_l\mathbf{z}^{l-1} + \mathbf{b}_l]_\sigma\,, \tag{2.32}$$

where $[\hat{\mathbf{z}}^l]_\sigma$ represents the element-wise non-linearity σ applied to each element of $\hat{\mathbf{z}}^l \in$

$\mathbb{R}^{M_l}$, and $\hat{\mathbf{z}}^l$ represents the affine transformation of the input $\mathbf{z}^{l-1}$: $\hat{\mathbf{z}}^l = \mathbf{W}_l \mathbf{z}^{l-1} + \mathbf{b}_l$. The layer parameters $\theta_l = \{\mathbf{W}_l, \mathbf{b}_l\}$ are the weight matrix $\mathbf{W}_l \in \mathbb{R}^{M_l \times M_{l-1}}$ and the bias vector $\mathbf{b}_l \in \mathbb{R}^{M_l}$. M_l corresponds to the number of rows of $\mathbf{W}_l$ and is often called the layer width. The typical non-linearities σ are:

- the ReLU [31]: $\mathrm{ReLU}(x) = \max(x, 0)$,
- the sigmoid: $\mathrm{sigm}(x) = \frac{1}{1+e^{-x}}$, and
- the hyperbolic tangent: $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$.

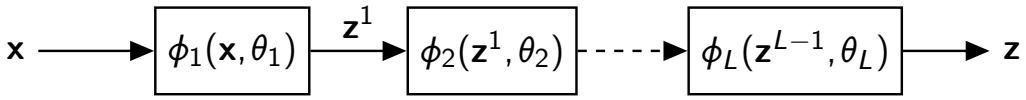


Figure 2.1: DNN transforms the input vector $\mathbf{x}$ to the feature vector $\mathbf{z}$ by a series of (non-linear) transforms.

We may obtain a DNN classifier from a DNN as

$$g(\mathbf{x}) = \arg\max_i (f(\mathbf{x}))_i \,, \tag{2.33}$$

where $(f(\mathbf{x}))_i$ corresponds to the i-th entry of the vector $f(\mathbf{x})$. Note that a loss function ℓ is often applied to the DNN output, i.e. $\ell(f(\mathbf{x}), y)$. This is due to the fact that the DNN classifier in (2.33) may not be optimized effectively by a gradient descent algorithm due to the $\arg\max$.

We will denote by $\mathcal{W}$ the set of all weight matrices of a DNN f:

$$\mathcal{W} = \{\mathbf{W}_1, \mathbf{W}_2, \ldots, \mathbf{W}_L\} \,. \tag{2.34}$$

This set will be relevant for the study of various DNN properties.

2.2.1.2 Convolutional Neural Network

A specialized version of a feed-forward DNN is the CNN, which is particularly important in practical applications, especially in computer vision. A CNN uses convolutional layers and pooling layers, which we describe next.

Convolutional Layer

Convolutional layer has the form of a typical DNN layer in (2.32) where that the weight matrices are block cyclic. In addition, each layer $\mathbf{z}^l$ of a CNN contains many channels or feature maps, i.e.,

$$\mathbf{z}^l = [\mathbf{z}_1^l, \mathbf{z}_2^l, \ldots, \mathbf{z}_{C_l}^l]^T, \tag{2.35}$$

where $\mathbf{z}_c^l \in \mathbb{R}^{F_l}$, $c = 1, \ldots, C_l$ are the feature maps of dimension F_l, C_l is the number of feature maps and $M_l = C_l \times F_l$ is the dimension of the layer. Note that $\mathbf{z}^l$ is obtained by applying an element-wise non-linearity to a vector of feature maps

$$\hat{\mathbf{z}}^l = [\hat{\mathbf{z}}_1^l, \hat{\mathbf{z}}_2^l, \ldots, \hat{\mathbf{z}}_{C_l}^l]^T \tag{2.36}$$

as in (2.32).

Take $\left(\hat{\mathbf{z}}_{c'}^l\right)_i$ to be the i-th element of the feature map $\hat{\mathbf{z}}_{c'}^l$, take $(\mathbf{z}_c^{l-1})_{i:i+k_l}$ to be the sub-vector of the feature map $\mathbf{z}_c^{l-1}$ between the indices i and $i+k_l$, and take $\mathbf{k}_{c'c}^l \in \mathbb{R}^{k_l}$ to be a filter at layer l, where $c' = 1, \ldots, C_l$, $c = 1, \ldots, C_{l-1}$ and $i = 1, \ldots, F_{l-1} - k_l$. Then the affine transformation $\hat{\mathbf{z}}^l = \mathbf{W}_l \mathbf{z}^{l-1} + \mathbf{b}_l$ in (2.32) can be represented as follows:

$$\left(\hat{\mathbf{z}}_{c'}^l\right)_i = \sum_{c=1}^{C_{l-1}} (\mathbf{k}_{c'c}^l)^T (\mathbf{z}_c^{l-1})_{i:i+k_l}, \quad i = 1, \ldots, F_l - k_l, \quad c' = 1, \ldots, C_l. \tag{2.37}$$

Note that filters $\mathbf{k}_l$ are applied locally and convolutionaly, i.e., if index i is interpreted as a time index then (2.37) represents a linear combination of convolutions between filters $\mathbf{k}_{cc'}$ and feature maps $\mathbf{z}_c^{l-1}$, $c = 1, \ldots, C_{l-1}$.

Pooling Layer

Pooling layers act as (non-linear) down-sampling operators and may also provide invariance to various input transformations such as translation [32, 33]. In particular, a pooling layer reduces the dimension of a feature map as

$$\mathbf{z}_c^l = \mathbf{P}^l(\mathbf{z}_c^{l-1})\mathbf{z}_c^{l-1}, c = 1, \ldots, C_{l-1}, \tag{2.38}$$

where $\mathbf{P}^l(\mathbf{z}_c^{l-1}) \in \mathbb{R}^{F_l \times F_{l-1}}$ is the pooling matrix. Note that the pooling layer preserves the number of channels, i.e., $C_l = C_{l-1}$.

The usual choices of pooling are down-sampling, max-pooling and average pooling. We denote by $\mathbf{p}_i^l(\mathbf{z}_c^{l-1})$ the i-th row of $\mathbf{P}^l(\mathbf{z}_c^{l-1})$ and assume that there are F_l pooling regions $\mathcal{P}_i$, $i = 1,\dots,F_l$, where each pooling region $\mathcal{P}_i$ is a subset of indices $1,2,\dots,F_{l-1}$. Various pooling operators can be represented as follows:

- **Down-sampling:** $\mathbf{p}_i^l(\mathbf{z}_c^{l-1}) = \mathbf{e}_{\mathcal{P}_i(1)}$, $i = 1,\dots,F_l$, where $\mathcal{P}_i(1)$ is the first element of the pooling region $\mathcal{P}_i$;
- **Max-pooling:** $\mathbf{p}_i^l(\mathbf{z}^{l-1}) = \mathbf{e}_{j^\star}^T$, $i = 1,\dots,F_l$, where $j^\star = \arg\max_{j'\in\mathcal{P}_i} |(\mathbf{z}^{l-1})_{j'}|$, and $\mathbf{e}_i$ is the i-th basis vector of the standard basis in $\mathbb{R}^{F_{l-1}}$;
- **Average pooling:** $\mathbf{p}_i^l(\mathbf{z}^{l-1}) = \frac{1}{|\mathcal{P}_i|}\sum_{j\in\mathcal{P}_i}\mathbf{e}_j^T$, $i = 1,\dots,F_l$, where $\mathbf{e}_i$ is the i-th basis vector of the standard basis in $\mathbb{R}^{F_{l-1}}$.

Note that F_l is usually much smaller than F_{l-1}. We will say that pooling regions are non-overlapping if $\mathcal{P}_i \cap \mathcal{P}_j = \emptyset \; \forall i \neq j \in \{1,\dots,F_l\}$.

The CNN structure is motivated by the research on cat's visual cortex [34], where the notion of *local receptive fields* was introduced. The local receptive field of a filter means that the filter is applied locally as in (2.37). The modern CNN architecture was proposed in [35]. Other authors have also proposed related architectures, such as the Neocognitron [36] and HMAX [37].

2.2.1.3 Other Deep Neural Network Architectures

In addition to feed-forward DNNs and CNNs there exist many other network architectures. As there are many variations of DNN architectures we are only going discuss the ones most relevant to this thesis.

Residual Network

In a Residual network [29, 38] the output of the l-th layer is a sum of a non-linear transformation applied to the output of the $l-1$-th layer and the output of the $l-1$-th layer:

$$\mathbf{z}^l = \phi_l(\mathbf{z}^{l-1}, \theta_l) + \mathbf{z}^{l-1}, \tag{2.39}$$

where $\phi_l(\mathbf{z}^{l-1}, \theta_l)$ may be a series of typical DNN layer (2.32). It has been shown in [29, 38] that the Residual networks with convolutional layers achieve better performance than standard CNNs on various computer vision tasks.

Extensions of CNN

Inspired by the translation invariance of CNNs, various extensions to CNN architectures have been proposed. For example, the authors in [39] view the convolutional layer of a conventional CNN as a specific case of convolution over a translation group. They show that the convolutional layer can be extended to implement a convolution over an arbitrary discrete group such as rotations and reflections. By employing an appropriate form of pooling one can then achieve invariance to corresponding group transformations. Similar ideas applied to rotation invariance have been explored in [40, 41].

Various authors have also explored layers that allow invariance to affine transformations [42, 43, 44]. For example, an affine transformation layer at a CNN input is introduced in [42, 43]. Such a layer allows a CNN to be invariant to affine transformations of the image plane.

Scattering Transform

The Scattering transform is a CNN like transform, which is based on the wavelet transform and and does not involve learning. Although the Scattering transform does not involve learning of the filters, it gives important theoretical and practical insights. For example, the authors in [45, 46] have shown that the Scattering transform achieves translation invariance and stability to deformations. These insights can be used to better understand properties of CNNs. The Scattering transform may also be extended to more general shift-invariant frames [47] and specialized to the discrete domain [48]. A rotationaly invariant scatering transform has also been proposed in [49].

Recurrent Neural Network and Long-Short Term Memory Network

Many problems in machine learning also involve time domain or sequential data. In such cases an architecture such as Recurrent Neural Network (RNN) [50] or Long-Short Term Memory (LSTM) network [51] might be used. In particular, layers are now indexed by a time index i, $i = 1, \ldots, i_{\mathsf{max}}$. A non-linear layer of a RNN has the following form:

$$\mathbf{z}_i^l = \left[\mathbf{W}_l[\mathbf{z}_i^{l-1}, \mathbf{z}_{i-1}^{l-1}]^T + \mathbf{b}_l\right]_\sigma , \tag{2.40}$$

where $\mathbf{z}_i^{l-1} \in \mathbb{R}^{M_{l-1}}$, $\mathbf{z}_i^l \in \mathbb{R}^{M_l}$, $t = 1, \ldots, i_{\mathsf{max}}$ and $\mathbf{W}_l \in \mathbb{R}^{M_l \times 2 \cdot M_{l-1}}$, $\mathbf{b}_l \in \mathbb{R}^{M_l}$. Therefore, output of the l-th layer is a function of the output of the $l-1$-th layer at the time

step i and the output of the $l-1$-th layer at the time step $i-1$. Note that due to recurrence in (2.40) the output of a RNN at time i is a function of all input vectors up to time i: $\mathbf{x}_1, \ldots, \mathbf{x}_i$.

2.2.1.4 Learning and Optimization of Deep Neural Networks

A DNN is trained by minimizing the objective function $\mathcal{L}(\Theta, S_m)$ with respect to the DNN's parameters $\Theta = \{\theta_1, \theta_2, \ldots, \theta_L\}$, where θ_l represents the parameters of layer l, $l = 1, \ldots, L$, and evaluated on the training set S_m. The objective $\mathcal{L}(\Theta, S_m)$ function is usually obtained as a sum of the empirical loss $\ell_{\text{emp}}(\Theta, S_m)$ and a regularizier function, which is denoted by $R(\Theta, S_m)$:

$$\mathcal{L}(\Theta, S_m) = \ell_{\text{emp}}(\Theta, S_m) + \lambda R(\Theta, S_m)\,. \tag{2.41}$$

In general both, the empirical loss $\ell_{\text{emp}}(\Theta, S_m)$ and the regularizer $R(\Theta, S_m)$ are functions of the DNN's parameters Θ and the training set S_m. λ balances the contributions of the loss and the regularizer in the objective function. We cover regularizers in more detail in the next section.

As mentioned at the beginning of this section, DNNs are usually trained by a gradient descent algorithm or its stochastic version. In particular, the gradient of the objective function $\mathcal{L}(\Theta, S_m)$ with respect to the parameters θ_l in the l-th layer is denoted by

$$\nabla_{\theta_l} \mathcal{L}(\Theta, S_m)\,. \tag{2.42}$$

The learning problem is then solved by applying a gradient descent step for $i_{\max}$ iterations:

$$\theta_l^{(i)} = \theta_l^{(i-1)} - \delta \nabla_{\theta_l} \mathcal{L}(\Theta, S_m)\,, \quad i = 1, \ldots, i_{\max}\,, \tag{2.43}$$

where δ is the learning rate and the superscript (i) denotes the gradient descent iteration.

In practice, especially when the number of training samples is large, we often use the Stochastic Gradient Descent (SGD) where at each step the objective is evaluated on a small sample from the training set [5]. In particular, take $\bar{S}_{m'}$ to be a subset of the

training set S_m with m' samples, which are picked at random from S_m. $\bar{S}_{m'}$ is called a batch, m' is called the batch size and $m' \ll m$, usually. The objective function at each optimization step i is then a function of the batch $\bar{S}_{m'}$:

$$\mathcal{L}'(\Theta, \bar{S}_{m'}) = \ell_{\text{emp}}(\Theta, \bar{S}_{m'}) + \lambda R(\Theta, \bar{S}_{m'}) . \tag{2.44}$$

Iterating the SGD step approximately m/m' times, so that the batches cover the entire training set, is usually referred to as running one epoch of training. The number of iterations used to train a DNN is then expressed in the number of epochs.

Momentum, Adagrad and Adam

There exist many improvements to the vanilla gradient descent or SGD that are used in practice. For example, gradient descent with momentum employs an exponentially decaying average of previous gradient steps to compute an update step in order to prevent oscillation of the parameters during optimization [52]. Other methods may also adopt the learning rate δ. Adagrad method adapts the learning step for every parameter by dividing it by the norm of the gradient [53]. Similarly to momentum and Adagrad method, Adam uses exponentially decaying average of previous gradients to compute an update step and also normalizes the learning rate of each parameter by an exponentially decaying variance of its gradients [54].

Global and Local Optima

Optimization of DNNs is a non-convex problem and the rationale behind the consistent success in optimization of these networks is still unclear. The authors in [55] model the objective function used to train DNNs with a spin-glass model and show that for large DNNs the local optima of the objective function are close to the global optima. This leads to an explanation that even local optima achieve sufficiently good performance in practice. A different line of work [56] leverages tensor factorization framework to study the optimization of DNNs and it shows that if the DNNs width is large enough it is possible to find the global minima from any initialization with a local descent algorithm.

Convergence Speed and Re-parametrization Methods

Another aspect of DNN optimization is its convergence speed. Optimization dynamics of a deep linear network are studied in [57], where it is shown that the learning speed

of deep networks may be independent of their depth if the weight matrices are roughly orthonormal. Orthonormal weight matrices ensure that gradients do not vanish as it might happen if the weight matrices have singular values close to 0.

A very successful approach for improving convergence speed of DNN optimization is re-parametrization of DNNs. Compared to the gradient step in (2.43), a re-parametrized gradient step is of the form

$$\theta_l^{(i)} = \theta_l^{(i-1)} - \delta \mathbf{A}_l^{(i-1)} \nabla_{\theta_l} \mathcal{L}(\Theta, S_m)\,, \tag{2.45}$$

where $\mathbf{A}_l^{(i-1)}$ is a re-parametrization matrix associated with the parameters θ_l of the l-th layer and evaluated at iteration index $i-1$. In particular, the idea is to pick an $\mathbf{A}_l^{(i)}$ that is better adopted to the geometry of the optimization landscape and which will lead to a faster convergence [58]. Due to a high dimensionality of the parameter space, $\mathbf{A}_l$ is often constrained to be a diagonal matrix. For example, the authors in [59] propose to set $\mathbf{A}_l^{(i)}$ to the diagonal of the inverse of the Fisher Information matrix. Other methods, such as in Path-SGD [60] or weight normalization [61], set the diagonal elements of $\mathbf{A}_l^{(i)}$ to the inverse of the norms of the corresponding weight matrix rows. In particular, weight normalization leads to weight matrices with normalized rows, i.e.,

$$\mathbf{W}_l = \mathrm{diag}(\hat{\mathbf{W}}_l^T \hat{\mathbf{W}}_l)^{-1} \hat{\mathbf{W}}_l\,, \tag{2.46}$$

where $\mathrm{diag}(\cdot)$ denotes the diagonal part of the matrix and $\hat{\mathbf{W}}_l \in \mathbb{R}^{M_l \times M_{l-1}}$.

The widely adopted batch normalization, where the output of each layer is normalized by the standard deviation of its outputs and centered to have a zero mean, may also be interpreted as a re-parametrization technique [62]. A batch normalized version of a non-linear layer $\mathbf{z}^{l+1} = [\mathbf{W}_l \mathbf{z}^l + \mathbf{b}_l]_\sigma$ in (2.32), where $\mathbf{W}_l$ is the weight matrix of the l-th layer and $\mathbf{b}_l$ is the bias vector of the l-th layer, has the following form:

$$\mathbf{z}^{l+1} = \left[\mathbf{N}\left(\{\mathbf{z}_i^l\}_{i=1}^m, \mathbf{W}_l, \mathbf{b}_l\right) \mathbf{W}_l \mathbf{z}^l + \mathbf{b}_l + \mathbf{n}\left(\{\mathbf{z}_i^l\}_{i=1}^m, \mathbf{W}_l, \mathbf{b}_l\right)\right]_\sigma\,, \tag{2.47}$$

where

$$\mathbf{N}\left(\{\mathbf{z}_i\}_{i=1}^m, \mathbf{W}_l, \mathbf{b}_l\right) = \operatorname{diag}\left(\sum_{i=1}^{m}(\mathbf{W}_l\mathbf{z}_i + \mathbf{b}_l)(\mathbf{b}_l^T + \mathbf{z}_i^T\mathbf{W}_l^T)\right)^{-\frac{1}{2}} \tag{2.48}$$

is the normalization matrix and

$$\mathbf{n}\left(\{\mathbf{z}_i^l\}_{i=1}^m, \mathbf{W}_l, \mathbf{b}_l\right) = -1/m\sum_{i=1}^{m}(\mathbf{W}_l\mathbf{z}_i + \mathbf{b}_l) \tag{2.49}$$

is the mean vector. A wide range of general re-parameterication methods is also studied in depth in [58].

2.2.1.5 Regularization of Deep Neural Networks

The role of the regularizer $R(\Theta, S_m)$ in (2.41) is to constrain the parameter space of a DNN with the aim of finding a solution that has a smaller GE than the solution of a non-regularized problem. There are many different approaches to regularization of DNNs. In some cases regularization is not achieved explicitly by a regularizer as in (2.41), but it is implicitly incorporated in the structure of the DNN or in the training procedure. We review various regularization methods next.

Weight Decay

A very well known method for DNN regularization includes weight decay [63], which penalizes the norm of the weight matrices. For example, ℓ_2-norm weight decay has the form

$$R(\Theta, S_m) = R(\Theta) = \sum_{\mathbf{W}\in\mathcal{W}} \|\mathbf{W}\|_F^2, \tag{2.50}$$

where $\|\cdot\|_F$ is the Frobenius norm and $\mathcal{W}$ is the set of all weight matrices of a DNN. The weight decay has been studied extensively from theoretical perspective and will be further discussed in Section 2.2.3.1.

Dropout and Effects of Re-parametrization

Another very popular regularization method is dropout [64]. The dropout method randomly sets some of the weight matrix rows to zero during training. This can be interpreted as randomly sampling smaller DNNs from a given larger DNN, and leads to a regularization effect. Although proposed as a heuristic, there are works that inter-

pret dropout as an approximation to Bayesian inference of a deep Gaussian process [65]. Note that dropout was critical for the state-of-the-art image classification results in 2012 [27]; however, the Residual networks, which achieve state-of-the-art results in 2017, do not use dropout [29] since it is not as effective when novel re-parametrization methods such as batch normalization are used [62]. Moreover, the effects of DNN re-normalization leading to a smaller GE has been observed in the case of path-SGD [60], weight normalization [61] and batch normalization [62].

Large Classification Margin

Various works attempt to promote a large classification margin in order to reduce the GE. The authors in [66, 67] focus on increasing the classification margin at the DNN output by leveraging the hinge loss. For example, the authors in [67] use the following regularization function:

$$R(\Theta, S_m) = \sum_{i=1}^{m} \left(1 - ((f(\mathbf{x}_i))_{y_i} - \max_{k \neq y_i}(f(\mathbf{x}_i))_k)\right)^2, \tag{2.51}$$

to promote a larger classification margin at the network output. Theoretical analysis of this approach is provided in [67] and will be further discussed in Section 2.2.3.1.

The authors in [68] constrain the spectral norms of the weight matrices to promote a large classification margin at the DNN input and evaluate the method on various datasets. In particular, they train DNNs with the following constraints on the weight matrices $\mathbf{W}_l$, $l = 1, \ldots, L$:

$$\tag{2.52}$$

$$\mathbf{W}_l \mathbf{W}_l^T \leq \mathbf{I}, \quad l = 1, \ldots, L, \tag{2.53}$$

which ensures that all singular values of $\mathbf{W}_l$ are upper bounded by 1. Their results are limited to DNNs with ReLU non-linearities, do not apply to CNNs. Moreover, the authors do not provide any theoretical analysis of the GE.

Stability and Robustness

Many works regularize DNNs by trying to improve their stability or robustness. For example, the authors in [69] regularize auto-encoders by constraining the Frobenius norm of the encoder's Jacobian matrix to improve the robustness of the obtained features. A

regularization method that enforces a DNN map to be a local isometry has been proposed in [70] and it has been shown that such a DNN is robust and generalizes better. In particular, they first construct a nearest neighbour graph of the training samples, which connects training samples with the same labels and an Euclidean distance smaller than $\alpha > 0$:

$$\mathcal{NB}_\alpha = \{(i,j) : y_i = y_j, \|\mathbf{x}_i - \mathbf{x}_j\|_2 \leq \alpha; s_i = (\mathbf{x}_i, y_i), s_j = (\mathbf{x}_j, y_j) \in S_m\}. \quad (2.54)$$

Then, the regularizer is defined as

$$R(\Theta, S_m) = \sum_{(i,j)\in\mathcal{NB}_\alpha} |\|f(\mathbf{x}_i) - f(\mathbf{x}_j)\|_2 - \|\mathbf{x}_i - \mathbf{x}_j\|_2| . \quad (2.55)$$

The stability training proposed in [71] aims at reducing the sensitivity of DNNs to various image distortion and artefacts, which appear as a consequence of image compression, cropping or rescaling. The stability regularization penalizes the ℓ_2-norm of the difference of a DNN outputs for the same image with different crops, scales or compression methods. The authors show empirically that such training improves the generalization of DNNs to such artefacts.

Data Augmentation

DNNs are very often trained with augmented training data, where in additional to the original training samples, the training set also contains transformed training samples [84]. For example, in image recognition training images are usually augmented by their translated, scaled and flipped versions. These transformations do not change the semantic meaning of the image, i.e., its class label. Data augmentation is used in all state-of-the-art DNNs [27, 29, 38, 110].

In addition, state-of-the-art results are usually reported by averaging DNN's predictions over augmented testing samples [27, 29, 38, 110]. In particular, the transformations used to create the augmented training set are also used to create transformed versions of each testing sample. For each of the transformed samples the network independently predicts its class label and the predictions of all the transformed samples are then averaged. This improves robustness of classification as the averaging reduces sensitivity of the network to a particular transformation of a testing sample.

2.2.2 Properties

We now review various properties of DNNs such as their approximation power, invariance and information preservation.

2.2.2.1 Approximation Power of Deep Neural Networks

An important topic in theoretical analysis of DNNs is their approximation power. For example, the works in [72, 73] showed that neural networks with a single hidden layer – shallow networks – can approximate any Borel measurable function.

However, many authors argue that although shallow networks may approximate any function, they are inefficient in doing so [4, 74], i.e., the number of parameters needed to approximate some functions is prohibitive. The authors in [74] model the expressive power of DNNs by the number of locally linear regions in the input space that a particular DNN can realize. They show that a number of input regions where the behaviour of a DNN is locally linear is of order

$$(M/N)^{(L-1)N} \times M^N, \tag{2.56}$$

where M represents the row dimension of the DNN weight matrices, N represents input dimension and L represents the number of layers. It is clear that the number of these regions is exponential in the network depth.

The authors in [75, 76] employ tensor factorization methods and conclude that functions implemented by DNNs are exponentially more expressive than functions implemented by shallow networks, meaning that number of parameters in a shallow network must grows exponentialy with the depth of a DNN in order to approximate the same class of functions. The work in [77] shows that for a given number of parameters and a given depth, there always exists a DNN that can be approximated by a shallower network only if the number of parameters in the shallow network is exponential in the number of layers of the deep network.

2.2.2.2 Invariance of Deep Neural Networks

Invariance properties of DNNs tell us how sensitive the DNNs are to certain "nuisances" present in the data. These "nuisances" are domain and task specific [78, 79]. For example, many image classification tasks are invariant to translation, i.e., object

has the same label no matter where in the image plane it appears. On the other hand, object localization, where the goal is to find a location of the object in the image plane, is not translation invariant.

Take a set of transformations

$$\mathcal{T} = \{t_0, t_1, \ldots, t_{T-1}\}, \tag{2.57}$$

where T represent the number of transformations and transformation $t \in \mathcal{T}$ is a map:

$$t : \mathcal{X} \to \mathcal{X} \tag{2.58}$$

that acts on $\mathbf{x} \in \mathcal{X}$ as $\mathbf{x}' = t(\mathbf{x})$. We say that a (classification) task is invariant to transformations in $\mathcal{T}$ if the target, e.g., class label is the same for all $t(\mathbf{x})$, $t \in \mathcal{T}$.[1] And we will say that a DNN is invariant if

$$f(t(\mathbf{x})) = f(t'(\mathbf{x})), \forall t', t \in \mathcal{T}. \tag{2.59}$$

Clearly, this does not hold for a general DNN. Note also that the equality in (2.59) may be relaxed to an approximate equality and such a DNN is then approximately invariant.

In many theoretical works it is often assumed that the set of transformations $\mathcal{T}$ is a group [79, 80, 81, 82] as this simplifies the analysis. In particular, an invariant representation of $\mathbf{x}$ can be obtained by either averaging over a group [79]:

$$\mathbf{x}_{\text{inv}} = \frac{1}{T} \sum_{i=0}^{T-1} t_i(\mathbf{x}), \tag{2.60}$$

or by maximization over a group [79]:

$$\mathbf{x}_{\text{inv}} = \max_{t \in \mathcal{T}} t(\mathbf{x}), \tag{2.61}$$

where the maximum is applied element-wise. Note that partial invariance is achieved by averaging or maximization over a subset of $\mathcal{T}$.

These ideas are a basis for translation invariance of CNNs. In particular, the con-

[1] We consider a discrete set of transformation for simplicity. The ideas also apply to continuous sets.

volutional layer is covariant with translations [39], where the layer $\phi(\mathbf{x}, \theta_l)$ is covariant with image plane translations, which are denoted by $\mathcal{T}$, when:

$$\phi(t(\mathbf{x}), \theta_l) = t(\phi(\mathbf{x}, \theta_l)), \quad t \in \mathcal{T}. \tag{2.62}$$

The convolutional layer is then followed by a pooling layer that either employs averaging over $\mathcal{T}$ or its subset, as in (2.60), or maximization over $\mathcal{T}$ or its subset, as in (2.61). The authors in [39] generalize the convolutional layers to be covariant with arbitrary discrete groups, which leads to architectures that can be invariant to various other transformations such as rotations and reflections. Similar ideas are applied in [40] to achieve rotation invariance.

As an alternative to encoding the invariances in the network architecture we can train a DNN or a CNN to become invariant. This is particularly helpful when we do not know how to explicitly construct a network with desired invariance properties [83]. Such an "approximate invariance" is achieved by training DNNs with data augmentation, which involves training the network with the transformed samples of the training examples [84]. Scattering transform is also translation invariant, and can be extended to rotation and scaling invariance [45, 46, 47, 48, 85].

2.2.2.3 Information Preservation of Deep Neural Networks

Another relevant question related to DNNs is what information is preserved at each layer. This question can be studied empirically, where insights are obtained by inverting a DNN representation at various DNN layers. For example, in case of CNNs an approximate inverse network called deconvolutional neural network was constructed by [86] and it is used to visualize particular layer representataions by re-projecting them onto the image domain. The authors show that deeper layers are more discriminative as well as more invariant. The authors in [87] propose to invert a DNN by solving the following optimization problem:

$$\mathbf{x}' = \arg\min_{\mathbf{x}} \ell(\phi_l(\mathbf{x}_0), \phi_l(\mathbf{x})) + \lambda R(\mathbf{x}), \tag{2.63}$$

where $\phi_l(\cdot)$ represents the output of the l-th layer of a DNN and $\phi_l(\mathbf{x}_0)$ represents the output of the l-th layer given a target image $\mathbf{x}_0$, which we want to reconstruct, as the

input. The loss ℓ compares the two representations and is usually set to the squared Euclidean norm. The term $R(\mathbf{x})$ is a regularizer, such as total variation, that promotes optimization within the space of natural images, and λ determines the trade-off between the loss function and the regularizer. It is observed in [87] that an image can be almost perfectly reconstructed from a shallower layers of a DNN and that the deeper layers preserve more abstract features of the image. Similar ideas are used in [88] to achieve "style transfer" between two images.

From theoretical perspective, the authors in [89] establish lower Lipschitz bounds of pooling layers and non-linear layers with ReLU non-linearities, where the Lipschitz bounds of an operator $f(\mathbf{x})$ are defined as constants $A, B \geq 0$ such that

$$\forall \mathbf{x}, \mathbf{x}', \quad A\|\mathbf{x}-\mathbf{x}'\|_2 \leq \|f(\mathbf{x})-f(\mathbf{x}')\|_2 \leq B\|\mathbf{x}-\mathbf{x}'\|_2\,, \tag{2.64}$$

and A is called the lower Lipschitz bound and B is called the upper Lipschitz bound. Positive lower Lipschitz bounds of all operators implies that a DNN is invertible. However, these bounds are hard to estimate in practice and authors rely on empirical experiments to show that in practice DNNs with random or trained weight are invertible. A related line of work studies DNNs with random weight where it is shown that if weight matrices are sufficiently large such DNNs perform distance preserving embedding of low-dimensional data manifolds [90], where the distance preservation is established by obtaining bounds on the upper and lower Lipschitz bounds A and B, which indicates that the network representation is invertible.

Information preservation is also studied in the context of the Scattering transform. For example, the authors in [91] establish energy decay bounds of layers that form the Scattering transform. In particular, they establish bounds on how many layers are needed to preserve a particular percentage of the input signal's energy.

2.2.3 Generalization Error Bounds for Deep Neural Networks

Of particular interest in this thesis are the works that study the GE of DNNs. DNNs differ from many other established methods such as SVMs because they usually have a much larger number of parameters, i.e., the number of parameters of DNNs in practice is often greater than the number of training samples [6]. We will review the existing work on the GE bounds of DNNs next.

2.2.3.1 Existing Generalization Error Bounds and Their Limitations

The GE of DNNs has been studied using the VC-dimension [7, 16, 92], the Rademacher or Gaussian complexities [17] and algorithmic robustness [23]. We review various approaches and report the GE bounds.

VC-dimension

The VC-dimension of DNNs with hard threshold non-linearities is bounded in the following theorem.

Theorem 2.5 ([7])**.** *Take $\mathcal{F}$ to be a class of DNNs with p parameters and hard threshold non-linearities: $\sigma(x) = \mathsf{sign}(x)$. The VC-dimension of $\mathcal{F}$ is*

$$VC(\mathcal{F}) = \mathcal{O}(p\log(p))\,, \tag{2.65}$$

where $\mathcal{O}$ denotes asymptotic behaviour in the sense $h(x) = \mathcal{O}(h'(x))$ if $\lim_{x\to\infty} \frac{h(x)}{h'(x)} = c$ and $c > 0$.

Theorem 2.2 then predicts that the GE of a DNN f in $\mathcal{F}$ is upper bounded by

$$GE(f) \leq \mathcal{O}\left(\sqrt{\frac{p\log(p)}{m}}\right). \tag{2.66}$$

Note that in the case of modern DNNs, where the number of parameters p if often greater than the number of training samples m [6], such bounds are clearly very loose and do not give a sufficient explanation for why the DNNs generalize well.

Rademacher Complexity

The GE of a DNN can also be bounded independently of the number of parameters, provided that the norms of the weight matrices are constrained appropriately. This can be acheived by the weight decay (2.50). For example, the work [93] studies the GE of DNNs with ReLU non-linearities and with constraints on the norms of the weight matrices.

Theorem 2.6. *Take a class of feed-forward DNNs $\mathcal{F}$, where $f \in \mathcal{F}$ has ReLU non-linearities and L layers. Assume that the weight matrices of $f \in \mathcal{F}$ satisfy:*

$$\prod_{i=1}^{L} \|\mathbf{W}_l\|_F \leq W\,, \tag{2.67}$$

and the bias vectors $\mathbf{b}_l$, $l = 1, \ldots, L$, *are set to zero. Then the RC of the class* $\mathcal{F}$ *is upper bounded by*

$$RC_m(\mathcal{F}) \leq \mathcal{O}\left(\frac{2^L W}{\sqrt{m}}\right). \tag{2.68}$$

The GE of a DNN f in $\mathcal{F}$ defined in Theorem 2.6 can be upper bounded by (via Theorem 2.1)

$$GE(f) \leq \mathcal{O}\left(\frac{2^L W}{\sqrt{m}}\right). \tag{2.69}$$

In the provided bound, the norm constraint on the weight matrices indicates that the number of parameters is not crucial for successful generalization. However, the bound also contains the 2^L term, which causes the bound to scale exponentially with the network depth. The GE bounds exponential in the number of DNN layers are also obtained in [67, 94]. Again, in practice DNNs with hundreds or even thousand layers generalize well [38], and these bounds do not explain their generalization ability.

Algorithmic Robustness

In the case of algorithmic robustness framework the authors in [23] provide the following theorem about robustness of DNNs:

Theorem 2.7 ([23])**.** *Take* f *to be a feed-forward DNN with* L *layers and ReLU nonlinearities, and take the biases* $\mathbf{b}_l$, $l = 1, \ldots, L$, *to be zero. Assume that the last layer of* f *is one dimensional, i.e.,* $f(\mathbf{x}) \in \mathbb{R}^N$, *and take the loss* $\ell(f(\mathbf{x}), y) = |y - f(\mathbf{x})|$. *Finally, assume that there exists* α *such that*

$$\max_{l,i} \sum_{j=1}^{M_l} |(\mathbf{W}_l)_{ij}| \leq \alpha, \tag{2.70}$$

where $\mathbf{W}_l$ *is the weight matrix at layer* l, $l = 1, \ldots, L$, *and* $\sum_{j=1}^{M_l} |(\mathbf{W}_l)_{ij}|$ *is the* ℓ_1*-norm of the* i*-th row of the matrix* $\mathbf{W}_l$. *Then the DNN* f *is* $(\mathcal{N}(\mathcal{S}; \|\cdot\|_\infty, \gamma/2), \alpha^L \gamma)$*-robust for all* $\gamma > 0$, *where* $\mathcal{N}(\mathcal{S}; \|\cdot\|_\infty, \gamma/2)$ *is the* $\gamma/2$*-covering number of the sample space* $\mathcal{S}$.

Following Theorem 2.4, the GE of such a robust DNN is bounded by

$$GE(f) \leq \alpha^L \gamma + \mathcal{O}\left(\sqrt{\frac{\mathcal{N}(\mathcal{S}; \|\cdot\|_\infty, \gamma/2)}{m}}\right). \tag{2.71}$$

The second term in (2.71) depends only on the covering number $\mathcal{N}(\mathcal{S}; \|\cdot\|_\infty, \gamma/2)$ of the sample space $\mathcal{S}$. However, the first term depends exponentially on the DNN depth via α^L. This term vanishes with depth if $\alpha < 1$ and grows exponentially with depth if $\alpha > 1$. Note that many successful re-parametrization approaches such as weight normalization [61] and batch normalization [62] lead to ℓ_2-normalized rows of the weight matrices. This implies that $\alpha \geq 1$ in practice, which is the main disadvantage of this bound as it indicates that the GE is always greater than zero independently of the number of samples for any $\gamma > 0$. Note that a smaller γ leads to a smaller first term in (2.71), however, this also leads to a larger covering number $\mathcal{N}(\mathcal{S}; \|\cdot\|_\infty, \gamma/2)$.

Limitations of the VC-dimension and the RC

Other authors have also observed that the existing GE bounds are not aligned well with practice. In particular, the authors in [6] provide the following result.

Theorem 2.8. *There exists a two-layer neural network with ReLU non-linearities and $2m+N$ parameters that can represent any function on a training set of size m, where N is the dimension of the network input.*

Such a relationship between the number of parameters and the signal dimension is often observed in practice [6]. The authors have also shown experimentally that DNNs in practice have ability to fit random labels. As per discussion in Section 2.1.1, the VC-dimension and the RC are measures of the ability of a function class to fit random labels. Based on this observations, the authors in [6] conclude that the VC-dimension and the RC are not appropriate for obtaining meaningful GE bounds for DNNs.

2.2.3.2 Data Symmetries and Generalization Error

As discussed in Section 2.2.2.2, DNNs are often designed to be invariant to certain symmetries present in the data. There are not many works that provide GE bounds for such scenarios. The authors in [81] discuss the intuition that DNN invariance reduces the sample complexity and the authors in [95] claim that pooling adopted to the

input signal geometry leads to better generalization, however, they do not offer any mathematical justification.

The GE of classifiers with invariance properties has been studied via the VC-dimension in [96], where the invariances are incorporated into the learning problem via additional constraints on the function class. It is shown that the subset of an hypothesis class that is invariant to certain transformations is smaller than the general hypothesis class. Therefore, it has a smaller VC-dimension. Yet, the authors do not provide any characterization of how much smaller the VC-dimension of an invariant method might be. This approach also does not take into account the symmetries of the data. Group symmetry in data was also explored in the problem of covariance estimation, where it is shown that leveraging group symmetry leads to gains in sample complexity of the covariance matrix estimation [97, 98].

2.3 Summary

In this chapter we have introduced a classification problem and reviewed various frameworks that provide GE bounds. We have then presented DNNs and overviewed their properties. In particular, we have explored existing GE bounds applied to DNNs and noticed that the existing GE bounds do not apply to modern DNNs that are very deep or wide, as shown empirically and theoretically in [6].

In the rest of the thesis we will focus on novel GE bounds for DNNs. In Chapter 3 we will leverage the notion of classification margin to show that GE of a DNN does not scale with its depth or width. In Chapter 4 we extend our proposed GE bounds to naturally incorporate data symmetries and DNN invariance. We directly relate the ratio of the GE bounds of an invariant and a non-invariant DNN to the size of the set of symmetries that a DNN is invariant to. Finally, we propose two regularizers that are derived directly from the theoretical results: the Jacobian regularizer and the invariance regularizer, that are presented in Chapter 5.

Chapter 3

Robustness and Generalization of Deep Neural Networks

In this chapter we present Generalization Error (GE) bounds for Deep Neural Networks (DNNs) based on the classification margin and the data covering number. We start by discussing a general large margin classifier and then show how DNNs can be understood as large margin classifiers. We verify the theoretical findings by experiments on real data.

3.1 Setup

We are interested in the GE of DNN classifiers. In particular, we will consider a DNN f that maps an input space $\mathcal{X}$ to an output space $\mathcal{Z}$:

$$f : \mathcal{X} \to \mathcal{Z}\,, \tag{3.1}$$

where the input space is a subset of the real N dimensional space $\mathbb{R}^N$: $\mathcal{X} \subseteq \mathbb{R}^N$ and the output space $\mathcal{Z}$ is the real $N_\mathcal{Y}$ dimensional space: $\mathcal{Z} = \mathbb{R}^{N_\mathcal{Y}}$, where $N_\mathcal{Y}$ represents the number of classes. A L layer DNN is obtained as

$$f(\mathbf{x}) = \phi_L(\phi_{L-1}(\cdots\phi_1(\mathbf{x},\theta_1),\cdots\theta_{L-1}),\theta_L)\,, \tag{3.2}$$

where

$$\mathbf{z}^l = \phi_l(\mathbf{z}^{l-1},\theta_l) \tag{3.3}$$

represents the l-th layer with parameters θ_l, output $\mathbf{z}^l \in \mathbb{R}^{M_l}$ and input $\mathbf{z}^{l-1} \in \mathbb{R}^{M_{l-1}}$, $l = 1, \ldots, L$. The input corresponds to $\mathbf{z}^0 = \mathbf{x}$ and the output of the last layer is denoted by $\mathbf{z} = f(\mathbf{x})$. A DNN classifier is defined as

$$g(\mathbf{x}) = \arg\max_i (f(\mathbf{x}))_i \,, \tag{3.4}$$

where $(f(\mathbf{x}))_i$ corresponds to the i-th entry of the vector $f(\mathbf{x})$.

We will assume that the DNN is trained on a training set

$$S_m = \{s_i\}_{i=1}^m = \{(\mathbf{x}_i, y_i)\}_{i=1}^m \,, \tag{3.5}$$

where $\mathbf{x}_i \in \mathcal{X}$, $i = 1, \ldots, m$, $y_i \in \{1, 2, \ldots, N_\mathcal{Y}\}$, $i = 1, \ldots, m$, and m is the number of training samples. The GE is defined as

$$GE(g) = |\ell_{\text{exp}}(g) - \ell_{\text{emp}}(g)| \,, \tag{3.6}$$

where $\ell_{\text{exp}}(g)$ is the expected loss defined in (2.8), and $\ell_{\text{emp}}(g)$ is the empirical loss defined in (2.7). The classification performance will be measured by the 0-1 loss: $\ell_{0\text{-}1}(g(\mathbf{x}), y) = \mathbb{1}(g(\mathbf{x}) \neq y)$, where $\mathbb{1}(\cdot)$ is the indicator function.[1]

3.1.1 Sample Space Covering Number and Data Complexity

Recall Theorem 2.4 that establishes GE bounds for robust learning algorithms. It is based on the division of the sample space $\mathcal{S} = \mathcal{X} \times \mathcal{Y}$, which is a product of the input space $\mathcal{X}$ and the label space $\mathcal{Y}$, into K partitions. A natural way to achieve a partitioning of the sample space is by covering it with metric balls of certain radius, as discussed in Section 2.1.1.4.[2] The number of partitions K is then equal to the covering number of the sample space.

In particular, the ρ-covering number of $\mathcal{S}$ corresponds to the smallest number of (pseudo-)metric balls of radius ρ needed to cover $\mathcal{S}$, and it is denoted by $\mathcal{N}(\mathcal{S}; d, \rho)$, where d denotes the (pseudo-)metric (see Definition 2.5). The space $\mathcal{S}$ is the Cartesian product of a continuous input space $\mathcal{X}$ and a discrete label space $\mathcal{Y}$, and we can write

[1] Note that the 0-1 loss is not continuous and it is used only to measure the classification performance, and not to train DNNs.

[2] In general, the partitions do not have to be of the same size. However, for the purpose of the analysis the partitions of the same size are the most convenient.

[23]

$$\mathcal{N}(\mathcal{S}; d, \rho) \leq N_{\mathcal{Y}} \cdot \mathcal{N}(\mathcal{X}; d, \rho)\,, \tag{3.7}$$

where $N_{\mathcal{Y}}$ corresponds to the number of classes. The choice of metric d determines how efficiently one may cover $\mathcal{X}$. A common choice is the Euclidean metric

$$d(\mathbf{x}, \mathbf{x}') = \|\mathbf{x} - \mathbf{x}'\|_2,, \quad \mathbf{x}, \mathbf{x}' \in \mathbb{R}^N\,, \tag{3.8}$$

which will be the default metric in this chapter.

The covering number of many structured low-dimensional data models can be bounded in terms of their "intrinsic" properties. For example, assuming that $\mathcal{X} = \mathbb{R}^N$:

- a Gaussian mixture model with L Gaussians and covariance matrices of rank at most k leads to a covering number $\mathcal{N}(\mathcal{X}; d, \rho) = L(1 + 2/\rho)^k$ [99];
- k-sparse signals in a dictionary with L elements have a covering number $\mathcal{N}(\mathcal{X}; d, \rho) = \binom{L}{k}(1 + 2/\rho)^k$ [90];
- a C_M regular k-dimensional manifold, where C_M is a constant that captures its "intrinsic" properties, has a covering number $\mathcal{N}(\mathcal{X}; d, \rho) = \left(\frac{C_M}{\rho}\right)^k$ [100].

Moreover, we will say that the covering number is a measure of the data complexity. As shown above, the covering number grows with the rank of the covariances matrices in the case of Gaussian mixture models, or with the signal sparsity in the case of sparse models, or with the manifold dimension in the case of manifolds. The notion of data complexity will play an important role in the interpretation of our results.

3.2 Generalization Error Bounds for Large Margin Classifiers

We now discuss the GE bounds for large margin classifiers. We will leverage the results in later sections where we will provide the GE bounds for DNN classifiers.

3.2.1 Large Margin Classifier

A particular example of a robust learning algorithm is a large margin classifier. As it will be shown below, its GE is a function of the sample space covering number and the

classification margin. Formally, we define the classification margin as follows:

Definition 3.1 (Classification margin)**.** *The classification margin of a classifier $g(\mathbf{x})$ and a training sample $s_i = (\mathbf{x}_i, y_i)$ measured by a metric d is defined as*

$$\gamma^d(s_i) = \sup\{a \in \mathbb{R} : d(\mathbf{x}_i, \mathbf{x}) \leq a \implies g(\mathbf{x}) = y_i \, \forall \mathbf{x}\}. \tag{3.9}$$

The classification margin of a training sample s_i is the radius of the largest metric ball (induced by d) in $\mathcal{X}$ centered at $\mathbf{x}_i$ that is fully contained in the decision region associated with class label y_i induced by the classifier $g(\mathbf{x})$.[3] The robustness of a large margin classifier $g(\mathbf{x})$ is given by the following Theorem.

Theorem 3.1 (Adapted from Example 9 in [23])**.** *If there exists γ such that*

$$\gamma^d(s_i) > \gamma > 0 \quad \forall s_i \in S_m, \tag{3.10}$$

then the classifier $g(\mathbf{x})$ is $(N_{\mathcal{Y}} \cdot \mathcal{N}(\mathcal{X}; d, \gamma/2), 0)$-robust.

3.2.2 Generalization Error Bounds

The GE bound for a large margin classifier now follows directly from Theorems 2.4 and 3.1.

Corollary 3.1. *Assume that classifier $g(\mathbf{x})$ has a classification margin γ, i.e.,*

$$\gamma^d(s_i) > \gamma > 0 \quad \forall s_i \in S_m, \tag{3.11}$$

and take $\ell(g(\mathbf{x}_i), y_i)$ to be the 0-1 loss. Then for any $\delta > 0$, with probability at least $1 - \delta$,

$$GE(g) \leq \sqrt{\frac{2\log(2) \cdot N_{\mathcal{Y}} \cdot \mathcal{N}(\mathcal{X}; d, \gamma/2)}{m}} + \sqrt{\frac{2\log(1/\delta)}{m}}. \tag{3.12}$$

Proof: The proof follows directly from Theorems 2.4 and 3.1. ■

[3] Note that the metric ball may not have spherical shape. For example, it may have square or elliptical shape. The shape of the metric ball is determined by the choice of the metric.

The GE bound in (3.12) approaches zero at the rate $\sqrt{m}$, where m is the number of training samples. The GE bound also increases sub-linearly with the number of classes $N_{\mathcal{Y}}$. Finally, the GE bound depends on the complexity of the input space $\mathcal{X}$ and the classification margin γ via the covering number $\mathcal{N}(\mathcal{X}; d, \gamma/2)$. Next, we specialize Corollary 3.1 to the case where $\mathcal{X}$ is a C_M-regular k-dimensional manifold.

Corollary 3.2. *Assume that $\mathcal{X}$ is a (subset of) a C_M-regular k-dimensional manifold, where $\mathcal{N}(\mathcal{X}; d, \rho) \leq \left(\frac{C_M}{\rho}\right)^k$. Assume also that a classifier $g(\mathbf{x})$ has a classification margin γ, i.e.,*

$$\gamma^d(s_i) > \gamma > 0 \quad \forall s_i \in S_m\,, \tag{3.13}$$

and take $\ell(g(\mathbf{x}_i), y_i)$ to be the 0-1 loss. Then for any $\delta > 0$, with probability at least $1-\delta$,

$$GE(g) \leq \sqrt{\frac{\log(2) \cdot N_{\mathcal{Y}} \cdot 2^{k+1} \cdot (C_M)^k}{\gamma^k m}} + \sqrt{\frac{2\log(1/\delta)}{m}}\,. \tag{3.14}$$

Proof: The proof follows directly from Theorems 2.4 and 3.1. ■

The GE bound in (3.14) now clearly exposes the role of the classifier, which is captured via the achieved classification margin γ, and the role of the data model, which is captured by the constant C_M and the manifold dimension k that act as a proxy to data complexity. In particular, for a given classification margin γ, the GE bound increases with the manifold constant C_M and the manifold dimension k. In other words, given a classification margin γ, the sample complexity of learning increases with the data complexity. Similarly, for a given constant C_M and the manifold dimension k a larger classification margin γ leads to a smaller GE (bound).

3.3 Generalization Error Bounds for Large Margin Deep Neural Network Classifiers

We have discussed in Chapter 2 that there is a need for GE bounds for DNNs that do not depend on the DNN depth or width. In the previous section we have explored the GE of

a large margin classifier. In the remainder of this chapter we will focus on showing how a DNN can be interpreted as a large margin classifier and will also present arguments that show that its GE does not scale with the network depth or width.

3.3.1 The Geometrical Properties of Deep Neural Networks

Recall, that a DNN f is a map

$$f : \mathcal{X} \to \mathcal{Z}, \tag{3.15}$$

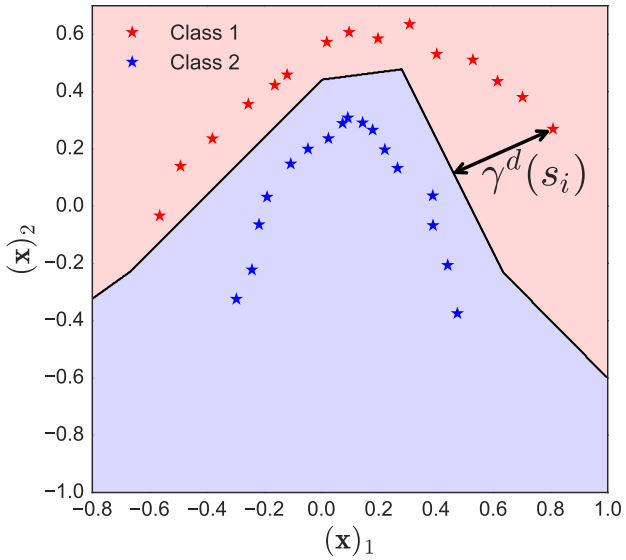
where we will refer to $\mathcal{X}$ as the input space and to $\mathcal{Z}$ as the output space. The classification margin introduced in Definition 3.1 is a function of the decision boundary in the input space. This is visualized in Figure 3.1 (a). However, DNN training aims at separation of the training samples in the output space (Figure 3.1 (b)), which does not necessarily imply a large classification margin and, hence, good generalization. In this section we introduce a general approach that allows us to bound the distance between two vectors at the network output by the distance between the vectors at the network input. We then use this results to establish bounds of the classification margin and the GE bounds that are independent of the network depth or width.

We start by defining the Jacobian Matrix (JM) of the DNN f[4]:

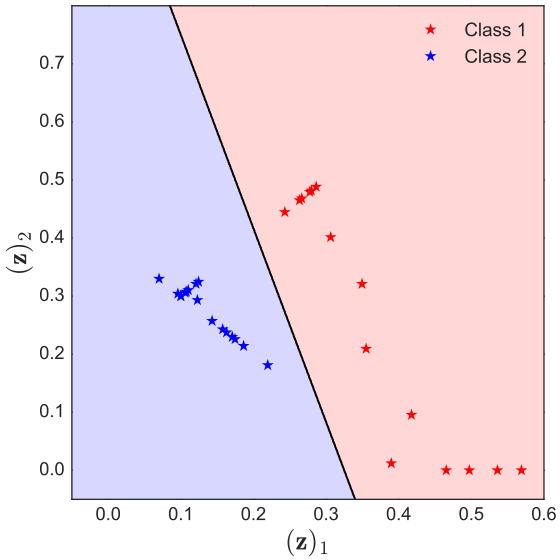
$$\mathbf{J}(\mathbf{x}) = \frac{df(\mathbf{x})}{d\mathbf{x}} = \prod_{l=0}^{L} \frac{d\phi_l(\mathbf{z}^{l-1})}{d\mathbf{z}^{l-1}} = \prod_{l=1}^{L} \frac{d\phi_l(\mathbf{z}^{l-1})}{d\mathbf{z}^{l-1}} \cdot \frac{d\phi_1(\mathbf{x})}{d\mathbf{x}}, \tag{3.16}$$

where we recall that $\mathbf{z}^0 = \mathbf{x}$ and corresponds to the DNN input, and $\mathbf{J}(\mathbf{x}) \in \mathbb{R}^{M_L \times N}$, M_l is the dimension of the output space $\mathcal{Z}$ and N is the dimension of the input space $\mathcal{X}$. Note that by the properties of the chain rule, the JM is computed as the product of the JMs of the individual network layers, evaluated at the appropriate values of the layer inputs $\mathbf{x}, \mathbf{z}^1, \ldots, \mathbf{z}^{L-1}$. We use the JM to establish a relation between a pair of vectors in the input space and the output space.

[4]The $\frac{df(\mathbf{x})}{d\mathbf{x}}$, where $f(\mathbf{x})$ is a vector function and $\mathbf{x}$ is a vector, will denote the JM of $f(\mathbf{x})$ throughout this thesis.



(a) Input space.



(b) Output space.

Figure 3.1: Decision boundaries in the input space and in the output space. Plot (a) shows samples of class 1 and 2 and the decision regions produced by a two-layer network projected into the input space. Plot (b) shows the samples transformed by the network and the corresponding decision boundary at the network output.

Theorem 3.2 (Jacobian property)**.** *For any* $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$ *and a DNN* f*, we have*

$$f(\mathbf{x}') - f(\mathbf{x}) = \int_0^1 \mathbf{J}(\mathbf{x} + u(\mathbf{x}' - \mathbf{x}))\, du\, (\mathbf{x}' - \mathbf{x}) \tag{3.17}$$

$$= \mathbf{J}_{\mathbf{x},\mathbf{x}'}(\mathbf{x}' - \mathbf{x}), \tag{3.18}$$

where

$$\mathbf{J}_{\mathbf{x},\mathbf{x}'} = \int_0^1 \mathbf{J}(\mathbf{x} + u(\mathbf{x}' - \mathbf{x}))\, du \tag{3.19}$$

is the average Jacobian on the line segment between $\mathbf{x}$ *and* $\mathbf{x}'$.

Proof: The proof appears in Appendix A.1. ■

As a direct consequence of Theorem 3.2, we can bound the distance $\|f(\mathbf{x}') - f(\mathbf{x})\|_2$, where f is the DNN and $\mathbf{x}, \mathbf{x}'$ are two input vectors, by the distance $\|\mathbf{x}' - \mathbf{x}\|_2$ and the DNN's JM:

Corollary 3.3 (Distance expansion bound)**.** *For any* $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$ *and a DNN* f*, we have*

$$\begin{aligned} \|f(\mathbf{x}') - f(\mathbf{x})\|_2 &= \|\mathbf{J}_{\mathbf{x},\mathbf{x}'}(\mathbf{x}' - \mathbf{x})\|_2 \\ &\leq \sup_{\mathbf{x}'' \in conv(\mathcal{X})} \|\mathbf{J}(\mathbf{x}'')\|_2 \|\mathbf{x}' - \mathbf{x}\|_2 , \end{aligned} \tag{3.20}$$

where $conv(\mathcal{X})$ *is the convex hull of* $\mathcal{X}$.

Proof: The proof appears in Appendix A.2. ■

We have established that $\mathbf{J}_{\mathbf{x},\mathbf{x}'}$ is the effective linear operator that relates the distance between $\mathbf{x}$ and $\mathbf{x}'$ to the distance between $f(\mathbf{x})$ and $f(\mathbf{x}')$. This implies that the maximum distance between any $f(\mathbf{x})$ and $f(\mathbf{x}')$ for any $\mathbf{x}$ and $\mathbf{x}'$ is bounded by the maximum spectral norm of the network's JM, as suggested by (3.20).

3.3.2 Network Layers and their Jacobian Matrices

Since the JM of f corresponds to the product of JMs of all the layers of f, as shown in (3.16), we now look into various layers used in modern DNNs and evaluate their JMs.

3.3.2.1 Linear and Softmax Layers

Linear and softmax layers are usually the last layers in a DNN and in case of classification have dimension $\mathbb{R}^{N_\mathcal{Y}}$, where $N_\mathcal{Y}$ corresponds to the number of classes.[5] A linear layer has the following form:

$$\mathbf{z} = \hat{\mathbf{z}}, \quad \hat{\mathbf{z}} = \mathbf{W}_L \mathbf{z}^{L-1} + \mathbf{b}_L , \tag{3.21}$$

[5] Assuming that there are $N_\mathcal{Y}$ one-vs.-all classifiers.

where $\mathbf{W}_L \in \mathbb{R}^{N_\mathcal{Y} \times M_{L-1}}$ is the weight matrix associated with the last layer and $\mathbf{b} \in \mathbb{R}^{N_\mathcal{Y}}$ is the bias vector associated with the last layer.

A more common choice for the last layer is the softmax layer:

$$\mathbf{z} = \zeta(\hat{\mathbf{z}}) = e^{\hat{\mathbf{z}}} / \left(\mathbf{1}^T e^{\hat{\mathbf{z}}}\right), \quad \hat{\mathbf{z}} = \mathbf{W}_L \mathbf{z}^{L-1} + \mathbf{b}_L\,, \tag{3.22}$$

where $\zeta(\cdot)$ is the softmax function and $\mathbf{W}_L$ and $\mathbf{b}_L$ are the same as in (3.21). Note that the exponential is applied element-wise. The elements of $\mathbf{z}$ are in range $(0,1)$ and are often interpreted as "probabilites" associated with the corresponding class labels.

For the remainder of this work we will take the softmax layer as the last layer of DNN, but note that all results still apply if the linear layer is used.

Jacobian Matrix

The JM of the linear layer defined in (3.21) is equal to the weight matrix

$$\frac{d\mathbf{z}}{d\mathbf{z}^{L-1}} = \mathbf{W}_L\,. \tag{3.23}$$

Similarly, in the case of softmax layer defined in (3.22) the JM is

$$\begin{aligned}\frac{d\mathbf{z}}{d\mathbf{z}^{L-1}} &= \frac{d\mathbf{z}}{d\hat{\mathbf{z}}} \cdot \frac{d\hat{\mathbf{z}}}{d\mathbf{z}^{L-1}} \\ &= \left(-\zeta(\hat{\mathbf{z}})\zeta(\hat{\mathbf{z}})^T + \mathrm{diag}(\zeta(\hat{\mathbf{z}})\right) \cdot \mathbf{W}_L\,. \end{aligned} \tag{3.24}$$

Note that $\left(-\zeta(\hat{\mathbf{z}})\zeta(\hat{\mathbf{z}})^T + \mathrm{diag}(\zeta(\hat{\mathbf{z}})\right)$ corresponds to the JM of the softmax function $\zeta(\hat{\mathbf{z}})$, due to the fact that

$$\begin{aligned}\frac{d(\zeta(\hat{\mathbf{z}}))_i}{d(\hat{\mathbf{z}})_i} &= \frac{d}{d(\hat{\mathbf{z}})_i}\frac{e^{(\hat{\mathbf{z}})_i}}{\left(\mathbf{1}^T e^{\hat{\mathbf{z}}}\right)} = \frac{1}{\left(\mathbf{1}^T e^{\hat{\mathbf{z}}}\right)^2}\left(e^{(\hat{\mathbf{z}})_i}\left(\mathbf{1}^T e^{\hat{\mathbf{z}}}\right) - e^{(\hat{\mathbf{z}})_i} e^{(\hat{\mathbf{z}})_i}\right) \\ &= \frac{e^{(\hat{\mathbf{z}})_i}}{\left(\mathbf{1}^T e^{\hat{\mathbf{z}}}\right)}\left(1 - \frac{e^{(\hat{\mathbf{z}})_i}}{\left(\mathbf{1}^T e^{\hat{\mathbf{z}}}\right)}\right) = -(\zeta(\hat{\mathbf{z}}))_i(\zeta(\hat{\mathbf{z}}))_i + (\zeta(\hat{\mathbf{z}}))_i\,. \end{aligned} \tag{3.25}$$

and

$$\begin{aligned}\frac{d(\zeta(\hat{\mathbf{z}}))_i}{d(\hat{\mathbf{z}})_j} &= \frac{d}{d(\hat{\mathbf{z}})_j}\frac{e^{(\hat{\mathbf{z}})_i}}{\left(\mathbf{1}^T e^{\hat{\mathbf{z}}}\right)} = \frac{1}{\left(\mathbf{1}^T e^{\hat{\mathbf{z}}}\right)^2}\left(0 - e^{(\hat{\mathbf{z}})_i} e^{(\hat{\mathbf{z}})_i}\right) \\ &= -\frac{e^{(\hat{\mathbf{z}})_i}}{\left(\mathbf{1}^T e^{\hat{\mathbf{z}}}\right)}\frac{e^{(\hat{\mathbf{z}})_j}}{\left(\mathbf{1}^T e^{\hat{\mathbf{z}}}\right)} = -(\zeta(\hat{\mathbf{z}}))_i(\zeta(\hat{\mathbf{z}}))_j\,. \end{aligned} \tag{3.26}$$

Table 3.1: Point-wise non-linearities.

Name	Function: $\sigma(x)$	Derivative: $\frac{d}{dx}\sigma(x)$	Bound: $\sup_x \left\lvert \frac{d}{dx}\sigma(x) \right\rvert$
ReLU	$\max(x, 0)$	$\{1 \text{ if } x > 0; 0 \text{ if } x \leq 0\}$	≤ 1
Sigmoid	$\frac{1}{1+e^{-x}}$	$\frac{e^{-x}}{(1+e^{-x})^2}$	$\leq \frac{1}{4}$
Hyperbolic tangent	$\tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$	$1 - (\tanh(x))^2$	≤ 1

3.3.2.2 Pooling layers

A pooling layer reduces the dimension of the intermediate representation and is defined as

$$\mathbf{z}^l = \mathbf{P}^l(\mathbf{z}^{l-1})\mathbf{z}^{l-1}, \tag{3.27}$$

where $\mathbf{P}^l(\mathbf{z}^{l-1})$ is the pooling matrix. The usual choices of pooling are down-sampling, max-pooling and average pooling and are described in detail in Section 2.2.1.2.

Jacobian Matrix

The pooling operator defined in (3.27) is a linear or a piece-wise linear operator. The corresponding JM is, therefore, also a linear or a piece-wise linear and is equal to:

$$\frac{d\mathbf{z}}{d\mathbf{z}^{L-1}} = \mathbf{P}^l(\mathbf{z}^{l-1}). \tag{3.28}$$

3.3.2.3 Non-linear layers

A non-linear layer is defined as

$$\mathbf{z}^l = [\hat{\mathbf{z}}^l]_\sigma = [\mathbf{W}_l\mathbf{z}^{l-1} + \mathbf{b}_l]_\sigma, \tag{3.29}$$

where $[\hat{\mathbf{z}}^l]_\sigma$ represents the element-wise non-linearity applied to each element of $\hat{\mathbf{z}}^l \in \mathbb{R}^{M_l}$, and $\hat{\mathbf{z}}^l$ represents the linear transformation of the layer input: $\hat{\mathbf{z}}^l = \mathbf{W}_l\mathbf{z}^{l-1} + \mathbf{b}_l$. $\mathbf{W}_l \in \mathbb{R}^{M_l \times M_{l-1}}$ is the weight matrix and $\mathbf{b}_l \in \mathbb{R}^{M_l}$ is the bias vector. The typical non-linearities are the ReLU, the sigmoid and the hyperbolic tangent. They are listed in Table 3.1. The choice of non-linearity σ is usually the same for all the layers in the network.

Jacobian Matrix

The JM of the non-linear layer (3.29) can be derived in the same way as the JM of the softmax layer. We first define the JM of the point-wise non-linearity, which is a diagonal matrix[6]

$$\left(\frac{d\mathbf{z}^l}{d\hat{\mathbf{z}}^l}\right)_{ii} = \frac{d\sigma\left((\hat{\mathbf{z}}^l)_i\right)}{d(\hat{\mathbf{z}}^l)_i}, \quad i = 1, \ldots, M_l\,. \tag{3.30}$$

The derivatives associated with various non-linearities are provided in Table 3.1. The JM of the non-linear layer can be expressed as

$$\frac{d\mathbf{z}^l}{d\mathbf{z}^{l-1}} = \frac{d\mathbf{z}^l}{d\hat{\mathbf{z}}^l} \cdot \mathbf{W}_l\,. \tag{3.31}$$

3.3.2.4 Properties of the Jacobain Matrices

The most important properties of the JMs of the layers presented above are collected in the following Lemma.

Lemma 3.1 (Jacobian matrix spectral norm bounds)**.** *The following statements hold:*

1. *The spectral norm of the JMs of the linear layer in* (3.21)*, the softmax layer in* (3.22) *and non-linear layer in* (3.29) *with the ReLU, Sigmoid or Hyperbolic tangent non-linearities is upper bounded by*

$$\left\|\frac{d\mathbf{z}^l}{d\mathbf{z}^{l-1}}\right\|_2 \leq \|\mathbf{W}_l\|_2 \leq \|\mathbf{W}_l\|_F\,. \tag{3.32}$$

2. *Assume that the pooling regions of the down-sampling, max-pooling and average pooling operators are non-overlapping (as per definition in Section 2.2.1.2). Then the spectral norm of their JMs can be upper bounded by*

$$\left\|\frac{d\mathbf{z}^l}{d\mathbf{z}^{l-1}}\right\|_2 \leq 1\,. \tag{3.33}$$

Proof: The proof appears in Appendix A.3. ■

[6]Note that in case of ReLU the derivative of $max(x, 0)$ is not defined for $x = 0$, and we need to use subderivatives (or subgradients) to define the JM. We avoid this technical complication and simply take the derivative of $max(x, 0)$ to be 0 when $x = 0$. Note that this does not change the results in any way because the subset of $\mathcal{X}$ for which the derivatives are not defined has zero measure.

Lemma 3.1 shows that the spectral norms of all layers can be bounded in terms of their weight matrices. As a consequence, the spectral norm of the network's JM is bounded by the product of the spectral norms of the weight matrices. We leverage these facts in the next section where we provide new GE bounds.

3.3.3 Generalization Error Bounds

In this section we provide the classification margin bounds for a DNN classifier that allow us to bound its GE. We follow the common practice and assume that the networks are trained by a loss that promotes separation of different classes at the network output, e.g. Categorical Cross Entropy (CCE) loss or the hinge loss. In other words, the training aims at maximizing the score of each training sample, where the score is defined as follows:

Definition 3.2 (Score)**.** *The score of a training sample $s_i = (\mathbf{x}_i, y_i)$ is given by*

$$o(s_i) = \min_{j \neq y_i} \sqrt{2}(\boldsymbol{\delta}_{y_i} - \boldsymbol{\delta}_j)^T f(\mathbf{x}_i)\,, \tag{3.34}$$

where $\boldsymbol{\delta}_i \in \mathbb{R}^{N_\mathcal{Y}}$ is the Kronecker delta vector with $(\boldsymbol{\delta}_i)_i = 1$ and other elements equal to zero.

Recall the definition of the DNN classifier $g(\mathbf{x})$ in (3.4) and note that the decision boundary between class i and class j in the output space $\mathcal{Z}$ is given by the hyperplane $\{\mathbf{z} : (\mathbf{z})_i = (\mathbf{z})_j\}$, where $(\mathbf{z})_i$ is the i-th element of $\mathbf{z} = f(\mathbf{x})$. Therefore, the score of a training sample $(\mathbf{x}, y)$ corresponds to the distance between the $f(\mathbf{x})$ and the decision boundary between classes y and $i \neq y$ that is closest to $f(\mathbf{x})$ (assuming that the score is positive). However, a large score $o(s_i)$ does not necessarily imply a large classification margin $\gamma^d(s_i)$, which is important for robustness and generalization. Theorem 3.3 provides classification margin bounds expressed as a function of the score and the properties of the network.

Theorem 3.3 (Classification margin bounds)**.** *Assume that a DNN classifier $g(\mathbf{x})$, as defined in* (3.4)*, classifies a training sample $s_i = (\mathbf{x}_i, y_i)$ with the score $o(s_i) > 0$. Then the classification margin for this sample can be bounded as*

$$\gamma^d(s_i) \geq \frac{o(s_i)}{\sup_{\mathbf{x}:\|\mathbf{x}-\mathbf{x}_i\|_2 \leq \gamma^d(s_i)} \|\mathbf{J}(\mathbf{x})\|_2} \triangleq \gamma_1^d(s_i) \tag{3.35}$$

$$\geq \frac{o(s_i)}{\sup_{\mathbf{x} \in conv(\mathcal{X})} \|\mathbf{J}(\mathbf{x})\|_2} \triangleq \gamma_2^d(s_i) \tag{3.36}$$

$$\geq \frac{o(s_i)}{\prod_{\mathbf{W}_l \in \mathcal{W}} \|\mathbf{W}_l\|_2} \triangleq \gamma_3^d(s_i) \tag{3.37}$$

$$\geq \frac{o(s_i)}{\prod_{\mathbf{W}_l \in \mathcal{W}} \|\mathbf{W}_l\|_F} \triangleq \gamma_4^d(s_i)\,, \tag{3.38}$$

where $\mathcal{W} = \{\mathbf{W}_1, \ldots, \mathbf{W}_L\}$ is the set of all weight matrices of f.

Proof: The proof appears in Appendix A.4. ■

Note that the classification margin bound in (3.35) is a function of the classification margin $\gamma^d(s_i)$. Although this means that the bound in (3.35) can not be evaluated without knowing the actual classification margin $\gamma^d(s_i)$, the bound still provides important insight that will be discussed in Section 3.3.3.1. Given the classification margin bounds (3.35)-(3.38), we can specialize Corollary 3.2 to DNN classifiers.

Corollary 3.4. *Assume that $\mathcal{X}$ is a (subset of) C_M regular k-dimensional manifold, where $\mathcal{N}(\mathcal{X}; d, \rho) \leq \left(\frac{C_M}{\rho}\right)^k$. Assume also that a DNN classifier $g(\mathbf{x})$ has a lower bound to the classification margin γ_b, i.e.,*

$$\gamma_b^d(s_i) > \gamma_b > 0 \quad \forall s_i \in S_m\,, \tag{3.39}$$

for some $b \in \{1,2,3,4\}$, and take $\ell(g(\mathbf{x}_i), y_i)$ to be the 0-1 loss. Then for any $\delta > 0$, with probability at least $1-\delta$,

$$GE(g) \leq \sqrt{\frac{\log(2) \cdot N_{\mathcal{Y}} \cdot 2^{k+1} \cdot (C_M)^k}{\gamma_b^k m}} + \sqrt{\frac{2\log(1/\delta)}{m}}\,. \tag{3.40}$$

Proof: The proof follows from Theorems 2.4, 3.1 and 3.3. ■

Interpretation of Corollary 3.4 follows the interpretation of Corollary 3.2, which

hold for general large margin classifiers. Next, we study in more detail how we can guarantee a large classification margin when training DNNs.

3.3.3.1 Classification Margin and Regularization

We now leverage the classification margin bounds in Theorem 3.3 to construct sets of weight matrices that ensure a bounded classification margin and, therefore, a bounded GE.

Recall that $\mathcal{W} = \{\mathbf{W}_1, \ldots, \mathbf{W}_L\}$ is the set of weight matrices of a DNN. The set of all possible weight matrix sets is denoted by

$$\overline{\mathcal{W}} = \{\mathcal{W} : \mathbf{W}_l \in \mathbb{R}^{M_l \times M_{l-1}} \quad \forall \mathbf{W}_l \in \mathcal{W}\}. \tag{3.41}$$

Next, we consider weight matrix sets that constrain the weight matrices in $\mathcal{W}$ in such a way that the classification margin (or its lower bound) is greater than some $\gamma > 0$:

$$\overline{\mathcal{W}}_b = \{\mathcal{W} : \gamma_b^d(s_i) > \gamma \quad \forall s_i = (\mathbf{x}_i, y_i) \in S_m\}, \quad b \in \{1,2,3,4\}, \tag{3.42}$$

where $\gamma_b^d(s_i)$, $b = 1,2,3,4$, correspond to the classification margin bounds (3.35)-(3.38). Note also that $\gamma_b^d(s_i)$, $b = 1,2,3,4$, are functions of the DNN weight matrices in $\mathcal{W}$. Finally, according to Corollary 3.4, $\mathcal{W} \in \overline{\mathcal{W}}_b$ for some $b \in \{1,2,3,4\}$ ensures that the GE is bounded by $C\frac{1}{\sqrt{m}}\gamma^{-k/2}$, where $C = \sqrt{\log(2) \cdot N_{\mathcal{Y}} 2^{k+1} (C_M)^k}$ and we have neglected the term $\sqrt{\frac{2\log(1/\delta)}{m}}$.

We now express the sets $\overline{\mathcal{W}}_b$, $b = 1,2,3,4$, explicitly:

$$\overline{\mathcal{W}}_1 = \Big\{\mathcal{W} : \gamma \cdot \sup_{\mathbf{x}:\|\mathbf{x}-\mathbf{x}_i\|_2 \le \gamma^d(s_i)} \|\mathbf{J}(\mathbf{x})\|_2 < o(s_i) \quad \forall s_i = (\mathbf{x}_i, y_i) \in S_m\Big\}, \tag{3.43}$$

$$\overline{\mathcal{W}}_2 = \Big\{\mathcal{W} : \gamma \cdot \sup_{\mathbf{x} \in \mathrm{conv}(\mathcal{X})} \|\mathbf{J}(\mathbf{x})\|_2 < o(s_i) \qquad \forall s_i = (\mathbf{x}_i, y_i) \in S_m\Big\}, \tag{3.44}$$

$$\overline{\mathcal{W}}_3 = \Big\{\mathcal{W} : \gamma \cdot \prod_{\mathbf{W}_l \in \mathcal{W}} \|\mathbf{W}_l\|_2 < o(s_i) \qquad \forall s_i = (\mathbf{x}_i, y_i) \in S_m\Big\}, \tag{3.45}$$

$$\overline{\mathcal{W}}_4 = \Big\{\mathcal{W} : \gamma \cdot \prod_{\mathbf{W}_l \in \mathcal{W}} \|\mathbf{W}_l\|_F < o(s_i) \qquad \forall s_i = (\mathbf{x}_i, y_i) \in S_m\Big\}. \tag{3.46}$$

Note that while we want to maximize the score $o(s_i)$, we also need to constrain the network's Jacobian matrix $\mathbf{J}(\mathbf{x})$ (following $\overline{\mathcal{W}}_1$ and $\overline{\mathcal{W}}_2$) or the weight matrices in $\mathcal{W}$ (following $\overline{\mathcal{W}}_3$ and $\overline{\mathcal{W}}_4$) in order to have the GE bounded by $C\frac{1}{\sqrt{m}}\gamma^{-k/2}$, as discussed above. This stands in line with the common rationale of training DNNs in which we do not only aim at maximizing the score of the training samples to ensure a correct classification of the training set, but also have a regularization that constrains the network parameters. This combination eventually leads to a lower GE. The constraint sets in (3.43)-(3.46) impose different regularization techniques:

- The term $\gamma \cdot \sup_{\mathbf{x}:\|\mathbf{x}-\mathbf{x}_i\|_2 \leq \gamma^d(s_i)} \|\mathbf{J}(\mathbf{x})\|_2 < o(s_i)$ in (3.43) considers only the supremum of the spectral norm of the JM evaluated at the points within $\mathcal{N}_i = \{\mathbf{x} : \|\mathbf{x}-\mathbf{x}_i\|_2 \leq \gamma^d(s_i)\}$, where $\gamma^d(s_i)$ is the classification margin of the training sample $s_i = (\mathbf{x}_i, y_i)$ (see Definition 3.1). We can not compute the margin $\gamma^d(s_i)$, but can still obtain a rationale for regularization: as long as the spectral norm of the JM in the neighbourhood of the training sample $\mathbf{x}_i$ given by $\mathcal{N}_i$ is bounded, we will have good GE guarantees.

- The constraint on the JM $\gamma \cdot \sup_{\mathbf{x}\in\text{conv}(\mathcal{X})} \|\mathbf{J}(\mathbf{x})\|_2 < o(s_i)$ in (3.44) is more restrictive as it requires bounded spectral norm for all samples $\mathbf{x}$ in the convex hull of the input space $\mathcal{X}$.

- The constraints in (3.45) and (3.46) have a similar form, i.e.,

$$\gamma \cdot \prod_{\mathbf{W}_l \in \mathcal{W}} \|\mathbf{W}_l\|_2 < o(s_i)$$

and

$$\gamma \cdot \prod_{\mathbf{W}_l \in \mathcal{W}} \|\mathbf{W}_l\|_F < o(s_i)\,,$$

respectively. Note that the weight decay (see (2.50)), which aims at bounding the Frobenius norms of the weight matrices is linked to the constraint in (3.46). However, note also that the classification margin bound based on the spectral norm in (3.45) is tighter than one based on the Frobenius norm in (3.46). For example, take $\mathbf{W}_l \in \mathcal{W}$ to have orthonormal rows and be of dimension $M \times M$.

Then the constraint in (3.45), which is based on the spectral norm, is of the form $\gamma < o(s_i)$ and the constraint in (3.46), which is based on the Frobenius norm, is of the form $\gamma \cdot M^{L/2} < o(s_i)$. In the former case we have a constraint on the score, which is independent of the network width or depth. In the latter case the constraint on the output score is exponential in network depth and polynomial in network width. The difference is that the Frobenius norm does not take into account the correlation (angles) between the rows of the weight matrix $\mathbf{W}_l$, while the spectral norm does. Therefore, the bound based on the Frobenius norm corresponds to the worst case when all the rows of $\mathbf{W}_l$ are aligned. In that case $\|\mathbf{W}_l\|_F = \|\mathbf{W}_l\|_2 = \sqrt{M}$. On the other hand, if the rows of $\mathbf{W}_l$ are orthonormal $\|\mathbf{W}_l\|_F = \sqrt{M}$, but $\|\mathbf{W}_l\|_2 = 1$.

3.3.3.2 Illustration

The gist of our approach is now illustrated by a simple example. The input space $\mathcal{X}$ is a two dimensional real space, i.e., $\mathcal{X} = \mathbb{R}^2$ and $\mathbf{x} = [x_1, x_2]^T \in \mathcal{X}$. We consider a two class classification problem where both class 1 (red) and class 2 (blue) are formed from points sampled from a circle as shown in Figures 3.2(a) and 3.2(b). Both red and blue points can be represented by the circle equation

$$\mathbf{x}^T\mathbf{x} = x_1^2 + x_2^2 = q^2\,, \tag{3.47}$$

where q represents the radius of a circle. In the case of the red class the radius is denoted by q_r and is equal to $q_r = 0.5$ and in the case of the blue class the radius is denoted by q_b and is equal to $q_b = 1.5$.

To solve the classification problem we decide to map the points into one dimensional real output space, i.e., $\mathcal{Z} = \mathbb{R}$ by using the following function

$$z = f(\mathbf{x}) = w_1 \left(\mathbf{x}^T\mathbf{x}\right)^{p/2} + w_2\,, \tag{3.48}$$

where $w_1, w_2, p \in \mathbb{R}$, $p > 0$, are parameters of the non-linear transform $f(\mathbf{x})$. Note that $f(\mathbf{x})$ in (3.48) maps all points that lie on a circle in the input space $\mathcal{X}$ into a single point in the output space $\mathcal{Z}$ as demonstrated in Figure 3.2(c).

Next, we design a learning rule. In particular, we will denote the red class point

in the output space by z_r and the blue class point in the output space by z_b. We set the decision boundary in the output space $\mathcal{Z}$ to $z = 0$ so that $z < 0$ is classified as red and $z > 0$ is classified as blue. We also require the red and blue points to have a distance 1 from the decision boundary, i.e.,

$$z_r = -1, \quad z_b = 1\,, \tag{3.49}$$

which leads to the setup in Figure 3.2(c). By solving (3.48) for the conditions in (3.49) we obtain the parameter values

$$w_1 = \frac{2}{|r_r^p - r_b^p|}\,, \tag{3.50}$$

$$w_2 = -w_1 \left(\min(r_r, r_b)\right)^p - 1\,, \tag{3.51}$$

and $p > 0$ is arbitrary. We discuss how to set p next.

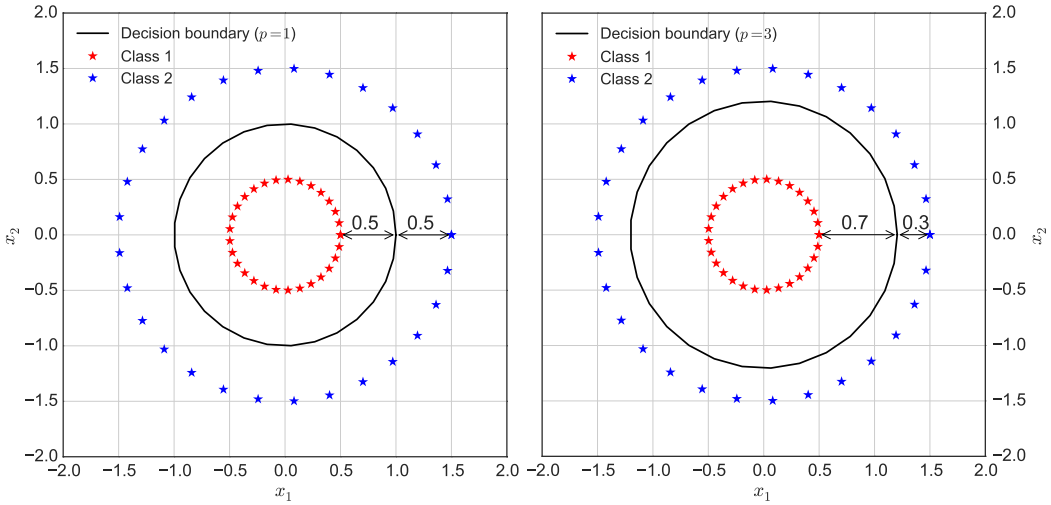
Recalling the discussion of this section, we want to design a classifier with the largest classification margin. Note that due to the form of $f(\mathbf{x})$ in (3.48), the decision boundary in the input space $\mathcal{X}$ is always a circle as shown in Figures 3.2(a) and 3.2(b). We will denote the radius of the decision boundary circle in the input space by q_m. It is now easy to confirm that the classification margin is equal to

$$\gamma = \min\left((q_b - q_m), (q_m - q_r)\right)\,. \tag{3.52}$$

Since the geometry of the points in the output space $\mathcal{Z}$ (see Figure 3.2(c)) is the same for all $p > 0$ we can not choose p based on the geometry of the points in the output space $\mathcal{Z}$. Therefore, we will use the rationale developed in this chapter and choose p for which the norm of the JM is the smallest and which also corresponds to the solution with the largest margin.

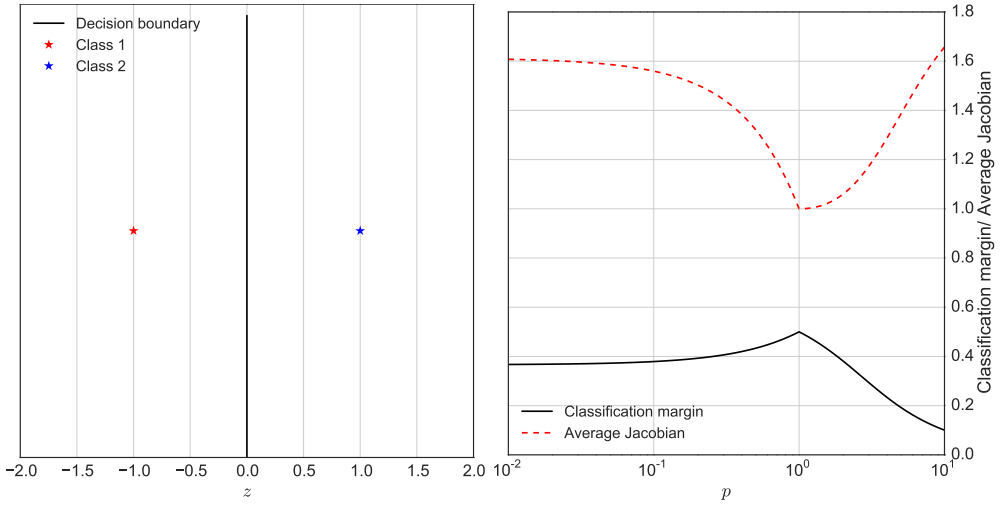
In particular, take a point on the red circle and denote it by $\mathbf{x}_r$ and take the closest point on the decision boundary and denote it by $\mathbf{x}_m$. Note that $f(\mathbf{x}_r) = -1$ and $f(\mathbf{x}_m) = 0$ due to our learning rule. By Theorem 3.2 we can write

$$f(\mathbf{x}_m) - f(\mathbf{x}_r) = 1 = \mathbf{J}_{\mathbf{x}_m,\mathbf{x}_r}(\mathbf{x}_m - \mathbf{x}_r)\,, \tag{3.53}$$



(a) Input space $\mathcal{X}$ and the decision boundary at $p = 1$.

(b) Input space $\mathcal{X}$ and the decision boundary at $p = 3$.



(c) Output space $\mathcal{Z}$ and the decision boundary.

(d) Classification margin (black) and the maximum norm of the average Jacobian $\max(\|\mathbf{J}_{\mathbf{x}_m,\mathbf{x}_r}\|_2, \|\mathbf{J}_{\mathbf{x}_b,\mathbf{x}_m}\|_2)$ (dashed red) for different values of parameter p.

Figure 3.2: Illustration of how the norm of the Jacobian matrix affects the classification margin.

where $\mathbf{J}_{\mathbf{x}_m,\mathbf{x}_r}$ is the average JM of $f(\mathbf{x})$ in (3.48) evaluated between $\mathbf{x}_m$ and $\mathbf{x}_r$ and the JM of $f(\mathbf{x})$ in (3.48) is

$$\mathbf{J}(\mathbf{x}) = w_1 p(\mathbf{x}^T\mathbf{x})^{p/2-1}\mathbf{x}. \tag{3.54}$$

Similarly, we take a point on the blue circle and denote it by $\mathbf{x}_b$ and take the closest line on the decision boundary and denote it by $\mathbf{x}_m$. Note that $f(\mathbf{x}_b) = 1$ and $f(\mathbf{x}_m) = 0$ due

to our learning rule. Again, by Theorem 3.2 we can write

$$f(\mathbf{x}_b) - f(\mathbf{x}_m) = 1 = \mathbf{J}_{\mathbf{x}_b,\mathbf{x}_m}(\mathbf{x}_b - \mathbf{x}_m)\,, \tag{3.55}$$

where $\mathbf{J}_{\mathbf{x}_b,\mathbf{x}_m}$ is the average JM of $f(\mathbf{x})$ in (3.48) evaluated between $\mathbf{x}_b$ and $\mathbf{x}_m$. We now note that (3.53) and (3.55) lead to inequalities

$$1/\|\mathbf{J}_{\mathbf{x}_m,\mathbf{x}_r}\|_2 \leq \|\mathbf{x}_m - \mathbf{x}_r\|_2 \tag{3.56}$$

and

$$1/\|\mathbf{J}_{\mathbf{x}_b,\mathbf{x}_m}\|_2 \leq \|\mathbf{x}_b - \mathbf{x}_m\|_2\,, \tag{3.57}$$

respectively. By recalling that $\|\mathbf{x}_m - \mathbf{x}_r\|_2 = q_m - q_r$ and $\|\mathbf{x}_b - \mathbf{x}_m\|_2 = q_m - q_r$ and by substituting (3.56) and (3.57) into (3.52) we obtain

$$\gamma \geq \min\left(1/\|\mathbf{J}_{\mathbf{x}_m,\mathbf{x}_r}\|_2, 1/\|\mathbf{J}_{\mathbf{x}_b,\mathbf{x}_m}\|_2\right) = 1/\max(\|\mathbf{J}_{\mathbf{x}_m,\mathbf{x}_r}\|_2, \|\mathbf{J}_{\mathbf{x}_b,\mathbf{x}_m}\|_2)\,. \tag{3.58}$$

Therefore, to maximize the classification margin γ we have to choose p such that the norms $\|\mathbf{J}_{\mathbf{x}_m,\mathbf{x}_r}\|_2$ and $\|\mathbf{J}_{\mathbf{x}_b,\mathbf{x}_m}\|_2$ will be as small as possible. The value of $\max(\|\mathbf{J}_{\mathbf{x}_m,\mathbf{x}_r}\|_2, \|\mathbf{J}_{\mathbf{x}_b,\mathbf{x}_m}\|_2,)$ for different values of p is plotted in Figure 3.2(d). We can see that for $p = 1$ we obtain the smallest value of $\max(\|\mathbf{J}_{\mathbf{x}_m,\mathbf{x}_r}\|_2, \|\mathbf{J}_{\mathbf{x}_b,\mathbf{x}_m}\|_2,)$. The classification margin is also plotted in the same plot in Figure 3.2(d). We can see that the classification margin is maximized at $p = 1$, as expected. The decision boundary at $p = 1$ is plotted at Figure 3.2(a) and we can see that it corresponds to the best possible classification margin. A suboptimal example of a decision boundary with $p = 3$ is plotted in Figure 3.2(b).

This simple example represents well the situation of DNNs where there are many parameters choices that achieve a correct classification of the training set. However, as we have shown here, choosing a solution with the smallest norm of the JM leads to a solution with the largest classification margin, which increases robustness and improves generalization.

3.3.3.3 Comparison to Other GE Bounds

We now compare our GE bounds to other bounds in the literature. We have also discussed these existing bounds in Section 2.2.3.

Rademacher Complexity and VC-dimension

First, we compare our GE bounds to the GE bounds based on the Rademacher Complexity (RC) in [93], which hold for DNNs with ReLU non-linearities. The work in [93] shows that if the product of the Frobenius norms of the weight matrices is bounded as

$$\prod_{\mathbf{W}\in\mathcal{W}} \|\mathbf{W}\|_F < W\,, \tag{3.59}$$

and the energy of training samples is bounded, then the GE of a DNN classifier g is bounded by:

$$GE(g) \leq \mathcal{O}\left(\frac{1}{\sqrt{m}} 2^{L-1} W\right). \tag{3.60}$$

Although the bounds (3.40) and (3.60) are not directly comparable, since the bounds based on the robustness framework rely on an underlying assumption on the data (covering number), there is still a remarkable difference between them. The behaviour in (3.60) suggests that the GE grows exponentially with the network depth even if the product of the Frobenius norms of all the weight matrices is fixed, which is due to the term 2^L. The bound in Corollary 3.4 and the constraint sets in (3.43)-(3.46), on the other hand, imply that the GE does not increase with the number of layers provided that the spectral/Frobenius norms of the weight matrices are bounded. Moreover, if we take the DNN to have weight matrices with orthonormal rows then the GE behaves as $\frac{1}{\sqrt{m}}(C_M)^{k/2}$ (assuming $o(s_i) \geq 1, i = 1, \ldots, m$), and, therefore, relies only on the complexity of the underlying data manifold and not on the network depth. This provides a possible answer to the open question of [93] that depth independent capacity control is possible in DNNs with ReLU non-linearities.

Second, we compare our results to the bounds based on the Vapnik-Cervonenkis (VC)-dimension. In particular, results in [7] suggest that the GE of a DNN classifier g

with p parameters is bounded by

$$GE(g) \leq \mathcal{O}\left(\sqrt{\frac{p\log(p)}{m}}\right). \tag{3.61}$$

Clearly, increasing the dimension of the weight matrices or adding layers will increase the number of parameters and consequently increase the GE. In contrast, the bound in Corollary 3.4 and the constraint sets in (3.43)-(3.46) imply that the GE of DNNs is not sensitive to the number of parameters.[7]

Algorithmic Robustness

Finally, we compare to the GE bound based on the algorithmic robustness framework [23] in (2.71):

$$GE(g) \leq \alpha^L \omega + \mathcal{O}\left(\sqrt{\frac{\mathcal{N}(\mathcal{S}; \|\cdot\|_\infty, \omega/2)}{m}}\right), \tag{3.62}$$

where $\omega > 0$. Note that the bound (3.62) is not directly comparable to the bound in Corollary 3.4 as in (3.62) the ℓ_1-norm loss is assumed and in Corollary 3.4 the 0-1 loss is assumed. Nevertheless, the bound in (3.62) includes the term $\omega\alpha^L$, where α is the constraint on the ℓ_1-norm of the rows of the weight matrices, and is exponential in the number of layers L. Due to our choice of the 0-1 loss, and the use of the classification margin, our bounds do not have such a term and the GE bound will approach zero as the number of training samples grows.

3.3.3.4 Comparison to Other Large Margin Approaches

Other works have considered the notion of a large classification margin applied to DNNs. In particular, the works in [66, 67] focus on increasing the classification margin at the DNN output. In this chapter we have shown that the classification margin at the DNN input, and not at the DNN output, determines the GE. As a results, our approach provides meaningful GE bounds.

The authors in [68] constrain the spectral norms of the weight matrices to promote a large classification margin at the DNN input. However, they do not provide any GE bounds. Moreover, their results are limited to DNNs with ReLU non-linearities and do

[7] Note that the (3.43)-(3.46) do not depend on the number of parameters, i.e., the size of the weight matrices, as long as the spectral or Frobenius norm of the weight matrices is bounded.

not apply to Convolutional Neural Networks (CNNs). Their results can be viewed as a special case of (3.37) applied to feed-forward DNN with ReLU non-linearities.

3.3.3.5 Analysis of the Weight Normalization and the Batch Normalization

As discussed in Section 2.2.1.5, many existing GE bounds for DNNs are a function of the norm of the weight matrices as in the case of RC [93] or algorithmic robustness [23]. On the other hand, the recent very successful re-parametrization methods such as the weight normalization [61] and the batch normalization [62], which improve optimization of DNNs as well as reduce the GE of the DNNs, lead to weight matrices with normalized rows. This implies that the norm of the weight matrix rows does not have a crucial effect on the GE. Note that our bounds in (3.43)-(3.45) do not depend on the norm of the weight matrix rows and do apply successfully to DNNs that have weight matrices with normalized rows.

An important value of our bounds is that they provide a possible explanation for the success of these state-of-the-art DNN re-parametrization techniques. In particular, weight normalized DNNs have weight matrices with normalized rows, i.e.,

$$\mathbf{W}_l = \operatorname{diag}(\hat{\mathbf{W}}_l^T \hat{\mathbf{W}}_l)^{-1} \hat{\mathbf{W}}_l, \tag{3.63}$$

where $\operatorname{diag}(\cdot)$ denotes the diagonal part of the matrix and $\hat{\mathbf{W}}_l \in \mathbb{R}^{M_l \times M_{l-1}}$. The Frobenious norm of the row-normalized weight matrix $\mathbf{W}_l$ is equal to $\|\mathbf{W}_l\|_F = \sqrt{M_l}$. Therefore, the GE bounds based on the Frobenius norm can not explain the good generalization of such networks as adding layers to a DNN or adding rows to the weight matrices will lead to a larger GE bound. This remark also applies to (3.46). However, (3.43)-(3.45) do not have this issue. A supporting experiment is presented in Section 3.5. We also note that the batch normalization [62] also leads to row-normalized weight matrices in DNNs with ReLUs:[8]

[8]To simplify the derivation we omit the bias vectors and, therefore, also the centering applied by the batch normalization. This does not affect the generality of the result.

Theorem 3.4 (Batch normalization property)**.** *Assume that the non-linear layers of a DNN with ReLUs are batch normalized as:*

$$\mathbf{z}^{l+1} = \left[\mathbf{N}\left(\{\mathbf{z}_i^l\}_{i=1}^m, \mathbf{W}_l\right)\hat{\mathbf{z}^l}\right]_\sigma, \quad \hat{\mathbf{z}}^l = \mathbf{W}_l\mathbf{z}^l, \tag{3.64}$$

where σ denotes the ReLU non-linearity and

$$\mathbf{N}(\{\mathbf{z}_i\}_{i=1}^m, \mathbf{W}) = diag\left(\sum_{i=1}^m \mathbf{W}\mathbf{z}_i\mathbf{z}_i^T\mathbf{W}^T\right)^{-\frac{1}{2}} \tag{3.65}$$

is the normalization matrix. Then all the weight matrices are row normalized. The exception is the weight matrix of the last layer, which is of the form $\mathbf{N}(\{\mathbf{z}_i^{L-1}\}_{i=1}^m, \mathbf{W}_L)\mathbf{W}_L$.

Proof: The proof appears in Appendix A.5. ■

3.4 Extensions

We have analysed the standard feed-forward DNNs and their classification margin in the preceding sections. We now briefly discuss how our results extend to the case where the training set is not perfectly separable by a DNN and how the results extend to other DNN architectures and to different margin metrics.

3.4.1 Beyond Feed-Forward DNNs

There are various DNN architectures such as Residual Networks [29, 38], Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) networks [51] that are used frequently in practice. It turns out that our analysis – which is based on the network's JM – can also be easily extended to such DNN architectures. In fact, the proposed framework encompasses all DNN architectures for which the JM can be computed. For example, we can compute the JM of a Residual Network.

Recall that the Residual Networks introduce short-cut connections between layers. In particular, the l-th layer of a Residual Network is given as

$$\mathbf{z}^l = \mathbf{z}^{l-1} + \phi(\mathbf{z}^{l-1}, \theta_l), \tag{3.66}$$

where $\phi(\mathbf{z}^{l-1}, \theta_l)$ is a non-linear transform. We denote by $\mathbf{J}_l(\mathbf{z}^{l-1})$ the JM of

$\phi(\mathbf{z}^{l-1}, \theta_l)$. Then the JM of the l-th layer is

$$\frac{d\mathbf{z}^l}{d\mathbf{z}^{l-1}} = \mathbf{I} + \mathbf{J}_l(\mathbf{z}^{l-1}), \tag{3.67}$$

and the JM of a Residual Network is of the form

$$\mathbf{J}_{SM}(\mathbf{z}^{L-1}) \cdot \left(\mathbf{I} + \sum_{l=1}^{L} \mathbf{J}_l(\mathbf{z}^{l-1}) \left(\prod_{i=1}^{l-1} (\mathbf{I} + \mathbf{J}_{l-i}(\mathbf{z}^{l-2})) \right) \right), \tag{3.68}$$

where $\mathbf{J}_{SM}(\mathbf{z}^{L-1})$ denotes the JM of the soft-max layer. In particular, the right element of the product in (3.68) can be expanded as

$$\begin{aligned} &\mathbf{I} + \mathbf{J}_1(\mathbf{x}) \\ &+ \mathbf{J}_2(\mathbf{z}^1) + \mathbf{J}_2(\mathbf{z}^1)\mathbf{J}_1(\mathbf{x}) \\ &+ \mathbf{J}_3(\mathbf{z}^2) + \mathbf{J}_3(\mathbf{z}^2)\mathbf{J}_2(\mathbf{z}^1)\mathbf{J}_1(\mathbf{x}) + \mathbf{J}_3(\mathbf{z}^2)\mathbf{J}_2(\mathbf{x}) + \mathbf{J}_3(\mathbf{x})\mathbf{J}_1(\mathbf{x}) \\ &+ \dots \end{aligned}$$

This is a sum of JMs of all the possible sub-networks of a Residual Network. In particular, there are L elements of the sum consiting of only one 1-layer sub-networks and there is only one element of the sum consisting of a L-layer sub-network. This observation is consistent with the claims in [102], which states that Residual Networks resemble an ensemble of relatively shallow networks.

3.4.2 Non-separable Training Sets

All GE bounds established in this chapter rely on the assumption that for every training sample s_i there exists a positive margin: $\gamma^d(s_i) > \gamma$. However, this may not always be the case in practice. If a DNN is not able to achieve a margin $\gamma > 0$ for every training sample we say that the training set is non-separable by the DNN. The GE bounds presented here can be easily extended to such cases by leveraging the notion of pseudo-robustness [23]. In particular, the notion of pseudo-robustness requires that a classification margin is positive only for a subset of training samples. One can then obtain the GE bounds of the same form with an additional penalty term that is a consequence of the fact that some of the training samples are not classified with a positive

margin.

3.4.3 Beyond the Euclidean Metric

The choice of the metric throughout this chapter was the Euclidean distance. However, we can also consider the geodesic distance on a manifold as a measure for margin instead of the Euclidean distance. The geodesic distance can be more appropriate than the Euclidean distance since it is a natural metric on the manifold. Moreover, the covering number of the manifold $\mathcal{X}$ may be smaller if we use the covering based on the geodesic metric balls, which will lead to tighter GE bounds. We outline the approach below.

Assume that $\mathcal{X}$ is a Riemannian manifold and $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$. Take a continuous, piecewise continuously differentiable curve $c(u)$, $u = [0, 1]$ such that $c(0) = \mathbf{x}$, $c(1) = \mathbf{x}'$ and $c(u) \in \mathcal{X}\ \forall u \in [0, 1]$. The set of all such curves $c(\cdot)$ is denoted by $\mathcal{C}$. Then the geodesic distance between $\mathbf{x}$ and $\mathbf{x}'$ is defined as

$$d_G(\mathbf{x}, \mathbf{x}') = \inf_{c(u) \in \mathcal{C}} \int_0^1 \left\| \frac{dc(u)}{du} \right\|_2 du. \tag{3.69}$$

Similarly as in Section 3.3.1, we can show that the JM of DNN is central to bounding the distance expansion between the signals at the DNN input and the signals at the DNN output.

Theorem 3.5 (Manifold distance expansion property)**.** *Take $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$, where $\mathcal{X}$ is a Riemannian manifold and take $c^\star(u)$, $u = [0, 1]$ to be a continuous, piecewise continuously differentiable curve connecting $\mathbf{x}$ and $\mathbf{x}'$ such that*

$$d_G(\mathbf{x}, \mathbf{x}') = \int_0^1 \left\| \frac{dc^\star(u)}{du} \right\|_2 du. \tag{3.70}$$

Then

$$\|f(\mathbf{x}') - f(\mathbf{x})\|_2 \leq \sup_{u \in [0,1]} \|\mathbf{J}(c^\star(u))\|_2 d_G(\mathbf{x}', \mathbf{x}) \tag{3.71}$$

Proof: The proof appears in Appendix A.6. ■

Note that we have established a relationship between the Euclidean distance of two points in the output space and the corresponding geodesic distance in the input

space. This is important because it implies that promoting a large Euclidean distance between points can lead to a large geodesic distance between the points in the input space. Moreover, the ratio between $\|f(\mathbf{x}') - f(\mathbf{x})\|_2$ and $d_G(\mathbf{x}, \mathbf{x}')$ is upper bounded by the maximum value of the spectral norm of the network's JM evaluated on the line $c^\star(u)$. This result is analogous to the results of Theorem 3.2 and Corollary 3.3. It also implies that regularizing the network's JM is beneficial also in the case when the classification margin is not measured in the Euclidean metric.

3.5 Experiments

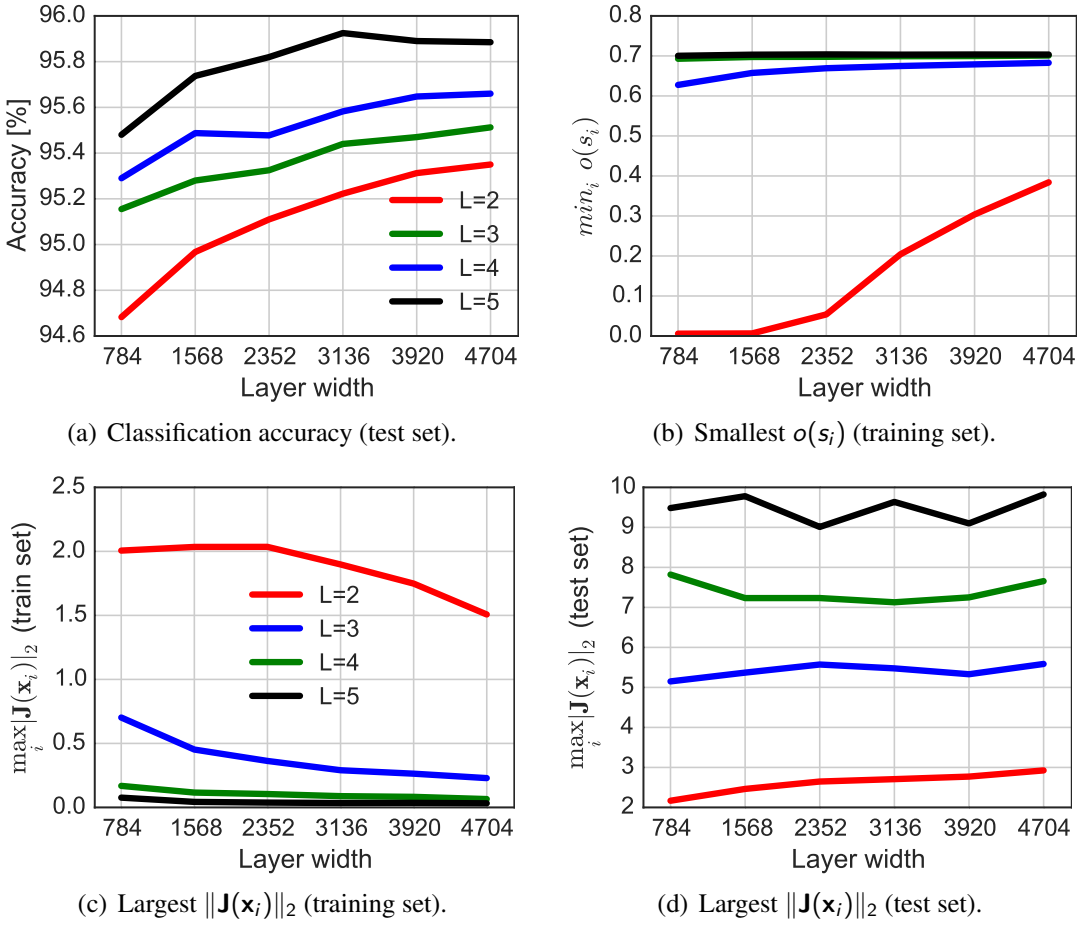
Next, we analyse the GE and the JM properties of a weight normalized DNN on the MNIST dataset [103] and relate the results to the theory presented in this chapter.

Setup

We train DNNs with a different number of fully connected layers ($L = 2, 3, 4, 5$) and different sizes of weight matrices ($M_l = \{784 \times i\}, i = 1, \ldots, 6, l = 1, \ldots, L-1$). ReLU non-linearities are used in all layers, the last layer is always the softmax layer, and the objective is the CCE loss. The networks were trained using the Stochastic Gradient Descent (SGD) with momentum, which was set to 0.9. Batch size was set to 128 and learning rate was set to 0.1 and reduced by factor 10 after every 40 epochs. The networks were trained for 120 epochs in total. Note that we train all networks for a fixed number of epochs and do not employ any form of early stopping [2] following many recent state-of-the-art works [29, 38, 61, 110]. All experiments are repeated 5 times with different random draws of a training set and different random weight initializations. We did not employ any additional regularization as our goal here is to explore the effects of the weight normalization on the DNN behaviour. We always use 5000 training samples.

Results

The classification accuracies on the test set are shown in Figure 3.3 (a) and the smallest classification score obtained on the training set is shown in Figure 3.3 (b). We have observed for all configurations that the training set accuracies were 100% (only exception is the case $L = 2$, $M_l = 784$ where the training accuracy was 99.6%). Therefore, the classification accuracies on the test set correspond directly to the GE. Note that the classification accuracy on the test set is increasing with the network depth and the



(a) Classification accuracy (test set).

(b) Smallest $o(s_i)$ (training set).

(c) Largest $\|\mathbf{J}(\mathbf{x}_i)\|_2$ (training set).

(d) Largest $\|\mathbf{J}(\mathbf{x}_i)\|_2$ (test set).

Figure 3.3: Weight normalized DNN with $L = 2, 3, 4, 5$ layers and different sizes of weight matrices (layer width). Plot (a) shows classification accuracy, plot (b) shows the smallest score of training samples, plot (c) shows the largest spectral norm of the network's JM evaluated on the training set and plot (d) shows the largest spectral norm of the network's JM evaluated on the testing set.

weight matrix size directly, which implies that the GE is smaller for deeper and wider DNNs. Note also that the score increases with the network depth and width. This is most obvious for the 2 and 3 layer DNNs, whereas for the 3 and 4 layer DNNs the score is close to $\sqrt{2}$ for all network widths.

Since the DNNs are weight normalized, the Frobenius norms of the weight matrices are equal to the square root of the weight matrix dimension, and the product of Frobenius norms of the weight matrices grows with the network depth and the weight matrix size. The increase of the minimum score with the network depth and network width does not offset the product of Frobenius norms, and clearly, the bound in (3.40) based on the margin bound in (3.38) and the bound based on the RC in (3.60), which leverage the Frobenius norms of the weight matrices, predict that the GE will increase

with the network depth and weight matrix size in this scenario. Therefore, the experiment indicates that these bounds are too pessimistic.

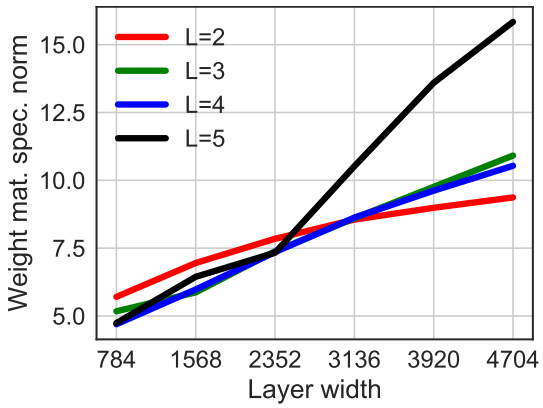


Figure 3.4: Maximum spectral norm of the weight matrices of a weight normalized DNN with $L = 1, 2, 3, 4$ layers and different sizes of weight matrices (layer width).

We have also inspected the spectral norms of the weight matrices of the trained networks, which are shown in Figure 3.4. In all cases the spectral norms were greater than one. We can argue that the bound in (3.40) based on the margin bound in (3.37) predicts that the GE will increase with network depth, as the product of the spectral norms grows with the network depth. We note however, that the spectral norms of the weight matrices are much smaller than the Frobenius norms of the weight matrices.

Finally, we look for a possible explanation for the success of the weight normalization in the bounds in (3.40) based on the margin bounds in (3.35) and (3.36), which are a function on the JM. The largest value of the spectral norm of the network's JM evaluated on the training set is shown in Figure 3.3 (c) and the largest value of the spectral norm of the network's JM evaluated on the testing set is shown in Figure 3.3 (d).

We can observe an interesting phenomena. The maximum value of the JM's spectral norm on the training set decreases with the network depth and width. On the other hand, the maximum value of the JM's spectral norm on the testing set increases with network depth (and slightly with network width). From the perspective of the constraint sets in (3.43) and (3.44) we note that in the case of the latter we have to take into account the worst case spectral norm of the JM for inputs in $\text{conv}(\mathcal{X})$. The maximum value of the spectral norm on the testing set indicates that this value increases with the network depth and implies that the bound based on (3.36) is still loose. On the other hand, the bound in (3.35) implies that we have to consider the JM in the neighbour-

hood of the training samples. As an approximation, we can take the spectral norms of the JMs evaluated at the training set. As it is shown in Figure 3.3 (c) this values decrease with the network depth and width. This implies that deeper and wider weight normalized DNNs achieve a larger classification margin and therefore generalize better.

3.6 Summary

We have presented a general framework for bounding the GE of DNNs based on their classification margin and the covering number of the input space. Moreover, we bound the classification margin in terms of the achieved separation between the training samples at the network output and the network's JM.

One of the hallmarks of our bounds relates to the fact that our characterization of the behaviour of the GE better follows the behaviour of the GE in practice than in the case of other bounds in the literature. Our bounds predict that the GE of DNNs can be independent of their depth and size whereas many other bounds say that the GE is exponential in the network width or depth.

The next chapter concentrates on further exploring how the data complexity and structure play a role in the GE of DNNs. In particular, we focus on data with symmetries and on DNNs that are invariant to such symmetries.

Chapter 4

Invariance and Generalization of Deep Neural Networks

We have shown in the previous chapter that the ability of Deep Neural Networks (DNNs) to generalize is a function of the classification margin that they achieve on the training examples and the complexity of the data, which is measured via its covering number. This implies that if a DNN is adapted to the data in such way that the "effective" complexity of the data is smaller, the generalization properties of the network should be improved. In fact, Deep Learning (DL) practitioners have been leveraging the knowledge about data structure in DNN design for a long time [35]. For example, Convolutional Neural Networks (CNNs), which use convolutional and pooling layers, are designed with the goal of being invariant to image plane translation.

In this chapter we extend the results of the previous section on the Generalization Error (GE) of DNNs to a particular scenario where the data possesses certain symmetries and a DNN is constructed in such a way that it is invariant to these symmetries. We show that in such scenarios the GE may be significantly smaller than in a general case.

4.1 Setup

Similarly as in Chapter 3, we are interested in the GE of DNN classifiers. We will consider a DNN f that maps an input space $\mathcal{X}$ to an output space $\mathcal{Z}$:

$$f : \mathcal{X} \to \mathcal{Z}\,, \tag{4.1}$$

where the input space is a subset of the real N dimensional space $\mathbb{R}^N$: $\mathcal{X} \subseteq \mathbb{R}^N$ and the output space $\mathcal{Z}$ is the real $N_\mathcal{Y}$ dimensional space: $\mathcal{Z} = \mathbb{R}^{N_\mathcal{Y}}$, where $N_\mathcal{Y}$ represents the number of classes. A DNN classifier is defined as

$$g(\mathbf{x}) = \arg\max_i (f(\mathbf{x}))_i \,, \tag{4.2}$$

where $(f(\mathbf{x}))_i$ corresponds to the i-th entry of the vector $f(\mathbf{x})$.

We will assume that the DNN is trained on a training set

$$S_m = \{s_i\}_{i=1}^m = \{(\mathbf{x}_i, y_i)\}_{i=1}^m \,, \tag{4.3}$$

where $\mathbf{x}_i \in \mathcal{X}$, $i = 1, \ldots, m$, $y_i \in \{1, 2, \ldots, N_\mathcal{Y}\}$, $i = 1, \ldots, m$ and m is the number of training samples. The GE is defined as

$$GE(g) = |\ell_{\text{exp}}(g) - \ell_{\text{emp}}(g)| \tag{4.4}$$

where $\ell_{\text{exp}}(g)$ is the expected loss defined in (2.8), and $\ell_{\text{emp}}(g)$ is the empirical loss defined in (2.7). The classification performance will be measured by the 0-1 loss: $\ell_{\text{0-1}}(g(\mathbf{x}), y) = \mathbb{1}(g(\mathbf{x}) \neq y)$, where $\mathbb{1}(\cdot)$ is the indicator function.

4.1.1 Sample Space Covering Number and Data Complexity

Recall that the covering number of the samples space $\mathcal{S} = \mathcal{X} \times \mathcal{Y}$, which is a product of the input space $\mathcal{X}$ and the label space $\mathcal{Y}$, can be interpreted as a measure of the data complexity, as discussed in more detail in Section 3.1.1. In particular, the ρ-covering number of $\mathcal{S}$ corresponds to the smallest number of (pseudo-)metric balls of radius ρ needed to cover $\mathcal{S}$, and it is denoted by $\mathcal{N}(\mathcal{S}; d, \rho)$, where d denotes the (pseudo-)metric (see Definition 2.5). The space $\mathcal{S}$ is the Cartesian product of a continuous input space $\mathcal{X}$ and a discrete label space $\mathcal{Y}$, and we can write [23]

$$\mathcal{N}(\mathcal{S}; d, \rho) \leq N_\mathcal{Y} \cdot \mathcal{N}(\mathcal{X}; d, \rho) \,, \tag{4.5}$$

where $N_{\mathcal{Y}}$ corresponds to the number of classes. The metric d is assumed in this chapter is the Euclidean metric

$$d(\mathbf{x}, \mathbf{x}') = \|\mathbf{x} - \mathbf{x}'\|_2, , \quad \mathbf{x}, \mathbf{x}' \in \mathbb{R}^N . \tag{4.6}$$

4.1.2 Data Symmetries and Robust and Invariant DNNs

In addition to the setup presented above, which follows the setup in Chapter 3, we assume that there are certain symmetries in the data that do not change the underlying class label. See for example Figure 4.1. We model these symmetries with a set of transformations and say that the classification task is invariant to such transformations. Similarly, we say that a DNN is invariant if its output is identical for all possible transformations – belonging to the set of transformations – of the input signal. We formalize these notions below.

(a) Original image. (b) 90° rotation. (c) 180° rotation. (d) 270° rotation.

Figure 4.1: In a classification task all the above images have the same label – *bird* – irrespective of the rotation.

4.1.2.1 Structured Input Space

The input space is denoted by $\mathcal{X}$. As noted in the previous paragraph, $\mathcal{X}$ often exhibits symmetries that may reduce its "effective" complexity and, therefore, also reduce the GE. We formalize this intuition in this section.

To capture the additional structure present in the data, we model the input space $\mathcal{X}$ as a product of a base space $\mathcal{X}_0$ and a set of transformations $\mathcal{T}$:

$$\mathcal{X} = \mathcal{T} \times \mathcal{X}_0 \coloneqq \{t(\mathbf{x}) : t \in \mathcal{T}, \mathbf{x} \in \mathcal{X}_0\}, \tag{4.7}$$

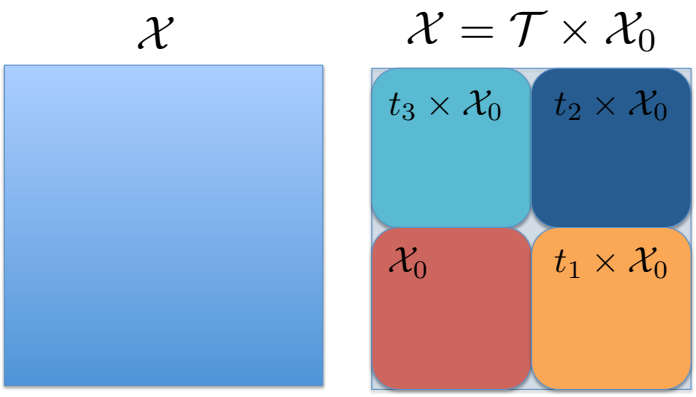
where $\mathcal{X}_0 \subseteq \mathbb{R}^N$, $\mathcal{T} = \{t_0, t_2, \dots t_{T-1}\}$, and T corresponds to the size of $\mathcal{T}$ and trans-

formation $t \in \mathcal{T}$ is a map[1]:

$$t : \mathbb{R}^N \to \mathbb{R}^N. \tag{4.8}$$

We take t_0 to be the identity, i.e., $t_0(\mathbf{x}) = \mathbf{x}$ throughout this chapter. For example, if $\mathcal{X}_0$ is a set of images and $\mathcal{T}$ is a set of translations, then $\mathcal{X}$ will be the set of all possible translations of the images in $\mathcal{X}_0$. These concepts are also illustrated in Figure 4.2.

In addition, we assume that the classification task is invariant to the set of transformations $\mathcal{T}$, i.e., the label of $t(\mathbf{x})$ is the same for all $t \in \mathcal{T}$. Intuitively, given the set of transformations $\mathcal{T}$, the complexity of the classification task is a function of the base space $\mathcal{X}_0$ and not the input space $\mathcal{X}$.



(a) Input space. (b) Input space decomposition.

Figure 4.2: Theorem 4.1 shows that the size of the input space $\mathcal{X}$ determines the GE of a robust DNN. The input space can often be constructed as a product of a simpler base space $\mathcal{X}_0$ and a set of transformations $\mathcal{T}$, where the transformations in $\mathcal{T}$ preserve the class labels. Theorem 4.2 shows that the GE of an invariant and robust DNN is determined by the size of the base space $\mathcal{X}_0$. The size of the base space $\mathcal{X}_0$ can be much smaller than the size of the input space $\mathcal{X}$.

We will measure the complexity of the base space $\mathcal{X}_0$ via its covering number in the same way that we measure the complexity of the input space $\mathcal{X}$.

[1]Note that the discrete representation of this set is not limiting in practice. Also note that if $\mathcal{T}$ is a group we may talk about about action of group $\mathcal{T}$ on $\mathcal{X}$, however, our formulation is more general.

4.1.2.2 Robust DNN and its Generalization Error

As we have seen in Chapter 3, the DNN's Jacobian Matrix (JM):

$$\mathbf{J}(\mathbf{x}) = \frac{df(\mathbf{x})}{d\mathbf{x}}, \tag{4.9}$$

plays an important role in bounding the classification margin and the GE of the DNN. In particular, a bounded spectral norm of the JM ensures that separation at the output of the DNN translates into a large classification margin at the network input. To simplify the discussion in this chapter we will assume that

$$\max_{\mathbf{x}\in\mathbb{R}^N} \|\mathbf{J}(\mathbf{x})\|_2 \leq 1. \tag{4.10}$$

This assumption allows us to focus on the role of invariance in the GE of DNNs while assuming that the DNN is robust.

Recall that the score of the training sample $s_i = (\mathbf{x}_i, y_i)$ is defined as

$$o(s_i) = \min_{j\neq y_i} \sqrt{2}(\boldsymbol{\delta}_{y_i} - \boldsymbol{\delta}_j)^T f(\mathbf{x}_i), \tag{4.11}$$

where $\boldsymbol{\delta}_i \in \mathbb{R}^{N_\mathcal{Y}}$ is the Kronecker delta vector with $(\boldsymbol{\delta}_i)_i = 1$ and other elements equal to zero (see Definition 3.2). A general GE bound for a robust DNN is given below.

Corollary 4.1. *Assume that a DNN classifier $g(\mathbf{x})$ satisfies* (4.10) *and that there exists a constant ω such that the score of each training samples is bounded as*

$$o(s_i) \geq \omega \quad \forall s_i \in S_m, \tag{4.12}$$

and consider the 0-1 loss. Then, for any $\delta > 0$, with probability at least $1-\delta$,

$$GE(g) \leq \sqrt{\frac{2\log(2)\cdot N_\mathcal{Y}\cdot \mathcal{N}(\mathcal{X}; d, \omega/2)}{m}} + \sqrt{\frac{2\log(1/\delta)}{m}}. \tag{4.13}$$

Proof: Proof follows directly from Theorems 3.1 and 3.3. ■

The role of the data complexity in (4.13) appears via the covering number $\mathcal{N}(\mathcal{X}; d, \omega/2)$. Note that the covering number is the function of the input space $\mathcal{X}$ and the achieved output score ω.

4.1.2.3 Invariant DNN

Given the set of transformations $\mathcal{T}$ it is reasonable to use an invariant DNN.[2]

Definition 4.1 (Invariant DNN)**.** *A DNN f is invariant to the set of transformations $\mathcal{T}$ if:*

$$f(t_i(\mathbf{x})) = f(t_j(\mathbf{x})) \quad \forall \mathbf{x} \in \mathcal{X}_0, \forall t_i, t_j \in \mathcal{T}\,. \tag{4.14}$$

We will denote such an invariant DNN by $f_{\mathcal{T}}$ and the corresponding DNN classifier by $g_{\mathcal{T}}$.

Next, we consider the GE of a robust and invariant DNN.

4.2 Generalization Error Bounds for Robust and Invariant Deep Neural Network Classifiers

In this section we provide bounds to the GE of invariant DNNs. The invariance of the DNN leads to mapping of different subsets of data space $\mathcal{X}$ into the same point and consequently leads to a more "efficient" covering of the input space $\mathcal{X}$, which translates into a lower GE.

The GE of a robust and invariant DNN can be bounded as follows:

Theorem 4.1 (GE of an invariant DNN)**.** *Assume that an invariant DNN classifier $g_{\mathcal{T}}$ satisfies* (4.10) *and that there exists a constant ω such that the score of each training sample is bounded as*

$$o(s_i) \geq \omega \quad \forall s_i \in S_m\,, \tag{4.15}$$

and consider the 0-1 loss. Then, for any $\delta > 0$, with probability at least $1-\delta$,

$$GE(g_{\mathcal{T}}) \leq \sqrt{\frac{2\log(2)\cdot N_{\mathcal{Y}}\cdot \mathcal{N}(\mathcal{X}_0; d, \omega/2)}{m}} + \sqrt{\frac{2\log(1/\delta)}{m}}\,. \tag{4.16}$$

Proof: The proof appears in Appendix B.1. ■

[2]Here we define a notion of absolute invariance. It is easy to extend it to approximate invariance, where in $\mathcal{X}$ we have transformed versions of $\mathcal{X}_0$ plus small/bounded noise; and also to extend the GE bounds in a similar manner for approximately invariant learning algorithms.

Note that the GE bound in Theorem 4.1 is of the same form as the GE bound in Corollary 4.1 and the main difference is in the employed covering number. In particular, the ratio between the bounds is (neglecting the term $\sqrt{\frac{2\log(1/\delta)}{m}}$)

$$r(\mathcal{X}_0, \mathcal{X}; d, \epsilon) = \left(\frac{\mathcal{N}(\mathcal{X}_0; d, \epsilon)}{\mathcal{N}(\mathcal{X}; d, \epsilon)} \right)^{1/2}, \tag{4.17}$$

where $\epsilon = \omega/2$.

We are especially interested in the scenario where the GE bound of an invariant DNN may be much smaller than the GE bound of a non-invariant DNN. This happens when $r(\mathcal{X}_0, \mathcal{X}; d, \epsilon) \ll 1$. We now establish a set of sufficient conditions on $\mathcal{X}_0$, $\mathcal{X}$, $\mathcal{T}$, d and ϵ such that $r(\mathcal{X}_0, \mathcal{X}; d, \epsilon) \ll 1$.

Theorem 4.2 (Covering number ratio bounds)**.** *Assume that $\mathcal{X} = \mathcal{T} \times \mathcal{X}_0$ and choose an $\epsilon < 1$. Then*

$$d(t(\mathbf{x}), t'(\mathbf{x}')) > 2\epsilon \quad \forall \mathbf{x}, \mathbf{x}' \in \mathcal{X}_0, t \neq t' \in \mathcal{T} \tag{4.18}$$

and

$$d(t(\mathbf{x}), t(\mathbf{x}')) \geq d(\mathbf{x}, \mathbf{x}') \quad \forall \mathbf{x}, \mathbf{x}' \in \mathcal{X}_0, t \in \mathcal{T} \tag{4.19}$$

$$\implies r(\mathcal{X}_0, \mathcal{X}; d, \epsilon) \leq 1/\sqrt{T}, \tag{4.20}$$

where T is the number of elements in $\mathcal{T}$. On the other hand,

$$d(t(\mathbf{x}), t'(\mathbf{x})) = 0 \quad \forall \mathbf{x} \in \mathcal{X}_0, t \neq t' \in \mathcal{T} \tag{4.21}$$

$$\implies r(\mathcal{X}_0, \mathcal{X}; d, \epsilon) = 1. \tag{4.22}$$

Proof. The proof appears in Appendix B.2. □

Theorem 4.2 establishes, via conditions on the geometry of the base space $\mathcal{X}_0$, and the effect of transformations in $\mathcal{T}$ on it, that the ratio $r(\mathcal{X}_0, \mathcal{X}; d, \epsilon)$ can be smaller or equal to $1/\sqrt{T}$. We now discuss the conditions (4.18), (4.19) and (4.21) in more detail:

- The condition in (4.18) can be stated as follows. Take any pair of vectors $\mathbf{x}, \mathbf{x}'$ in the base space $\mathcal{X}_0$ and transform them by two transformations t, t' in $\mathcal{T}$ that are not equal. Then the distance between the pair of transformed vectors $t(\mathbf{x}), t'(\mathbf{x}')$

must be at least 2ϵ. In other words, any two distinct transformations in $\mathcal{T}$ must transform any two vectors $\mathbf{x}, \mathbf{x}'$ in the base space $\mathcal{X}_0$ into sufficiently distinct vectors (distance at least 2ϵ). Intuitively, satisfying (4.18) means that each transformations in $\mathcal{T}$ transforms vectors from base space $\mathcal{X}_0$ in such a way that the transformed versions are not similar to transformed versions obtained by a different transformation.

- The condition in (4.19) ensures that the transformations in $\mathcal{T}$ do not contract the distances between the vectors in $\mathcal{X}_0$. This way the transformations in $\mathcal{T}$ do not reduce the complexity (measured by the covering number) of the base space $\mathcal{X}_0$. For example, a transformation that maps any $\mathbf{x} \in \mathcal{X}_0$ into itself violates (4.19) and leads to $\mathcal{X} = \mathcal{X}_0$, as formalized by (4.21).

The results of this section can be summarized by the following remarks:

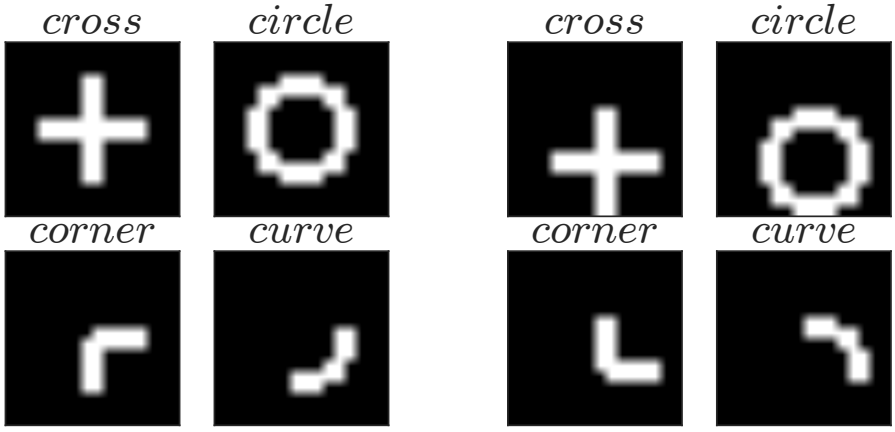
- Given an input space $\mathcal{X}$, which can be factored as $\mathcal{X} = \mathcal{T} \times \mathcal{X}_0$ according to (4.7) and the size of transformation set T, we have established that the GE of an invariant and robust DNN may be up to a factor $\sqrt{T}$ smaller than the GE of a non-invariant robust DNN.

- Note that both in the case of a non-invariant DNN in Corollary 4.1 and in the case of an invariant DNN in Theorem 4.1 we require the output score of the training examples to be at least ω, i.e., we require a DNN to be able to separate the training examples by a score of at least ω. A trivial invariant DNN that maps all points in $\mathcal{X}$ to the same point, will not be able to separate the training set and clearly, the results of Theorem 4.1 do not apply. In fact, we require from an invariant DNN to still be able to discriminate between different signals from different classes.

4.2.1 Illustration

To provide additional intuition related to Theorem 4.2, we present the following toy example. We consider four images of dimension $N \times N$, with $N = 16$, which are shown in Figure 4.3(a) and take d to be the Euclidean metric. The sets of transformations that we consider are:

- Translation set: The set of pixel-wise cyclic translations in any direction. The size of the set is N^2.



(a) Atoms. (b) Transformed atoms.

Figure 4.3: Examples of atoms and their transformations. (a) A set of atoms (*cross*, *circle*, *corner*, *curve*) used to construct the base space. (b) Examples of transformed atoms with a transformation from the trans-rotation set.

- Rotation set: The set of image rotations by 90°. The size of this set is 4.
- Trans-rotation set: A product of the translation and the rotation sets, where the rotation is applied first followed by a translation. The size of this set is $4 \times N^2$.

Note that all the transformations above can be implemented by permutation matrices which are orthonormal. This is important as it implies that all the considered sets satisfy the condition in (4.19). Examples of transformed atoms are shown in Figure 4.3(b).

We now provide an example of a base space $\mathcal{X}_0$ and a transformation set $\mathcal{T}$ for which $r(\mathcal{X}_0, \mathcal{X}; d, \epsilon) \leq 1/\sqrt{T}$; and then provide an example of a base space $\mathcal{X}_0$ and a transformation set $\mathcal{T}$ for which $r(\mathcal{X}_0, \mathcal{X}; d, \epsilon) = 1$.

Example for $r(\mathcal{X}_0, \mathcal{X}; d, \epsilon) \leq 1/\sqrt{T}$

Consider $\mathcal{X}_0 = \{cross, circle, corner, curve\}$ and $\mathcal{T}$ to be the translation set. The set $\mathcal{X} = \mathcal{T} \times \mathcal{X}_0$ then contains all possible translations of shapes in $\mathcal{X}_0$. We have verified by numerical computations that the condition in (4.18) is satisfied for all $\epsilon < 0.375$. Therefore, $R(\mathcal{X}_0, \mathcal{X}; d, \epsilon) \leq 1/\sqrt{T}$ for $\epsilon < 0.375$, where $\sqrt{T} = N = 16$ is the dimension of the images. Therefore, as suggested by Theorem 4.1, a translation invariant DNN can attain a GE with a factor N smaller than the GE of a non-invariant DNN. Similarly, if we take $\mathcal{X}_0 = \{corner, curve\}$ and $\mathcal{T}$ to be the trans-rotation set, we can establish $r(\mathcal{X}_0, \mathcal{X}; d, \epsilon) \leq 1/(2N)$ for $\epsilon < 0.26$.

Example for $r(\mathcal{X}_0, \mathcal{X}; d, \epsilon) = 1$

Now consider $\mathcal{X}_0 = \{cross, circle\}$ and $\mathcal{T}$ to be the rotation set. Therefore, $\mathcal{X} = \mathcal{T} \times \mathcal{X}_0$ contains all possible 90° rotations of *circle* and *cross* in Figure 4.3(a). It is clear that the *circle* and *cross* are already invariant to such rotation, i.e., they corresponds to exactly the same shape. Therefore, the condition in (4.21) holds and $r(\mathcal{X}_0, \mathcal{X}; d, \epsilon) = 1$. Clearly, in such cases, an invariant DNN is not expected to have a smaller GE than a non-invariant DNN.

4.3 Experiments

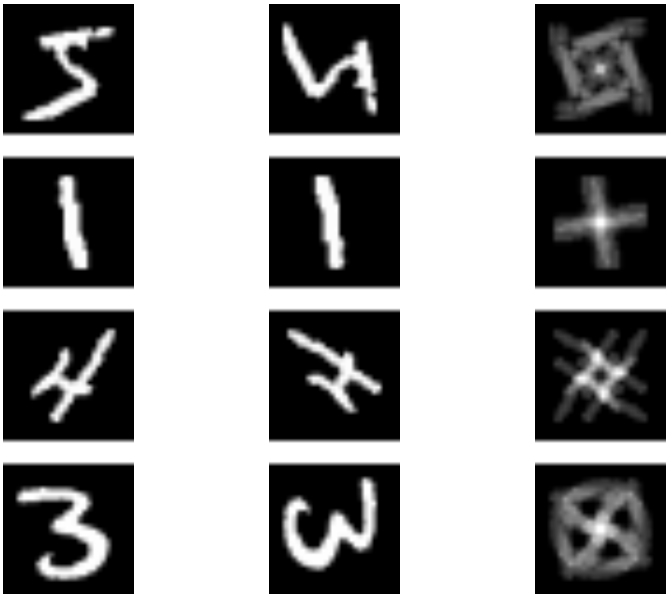
We now demonstrate the theoretical results with experiments on the MNIST dataset [103]. We compare a rotation invariant CNN and a conventional CNN on the rotated MNIST datasets.

Rotated MNIST dataset

The rotated MNIST-D dataset is constructed by rotating the digits by an angle $r \cdot D^\circ$, $r \in \{0, 1, 2, \ldots, 360/D - 1\}$, where the index r is chosen randomly for each image in the dataset. We use $D = 180, 90, 45$. Examples of digits from the original MNIST dataset are shown in Figure 4.4(a) and example of digits rotated by a random multiple of 90° are shown in Figure 4.4(b).

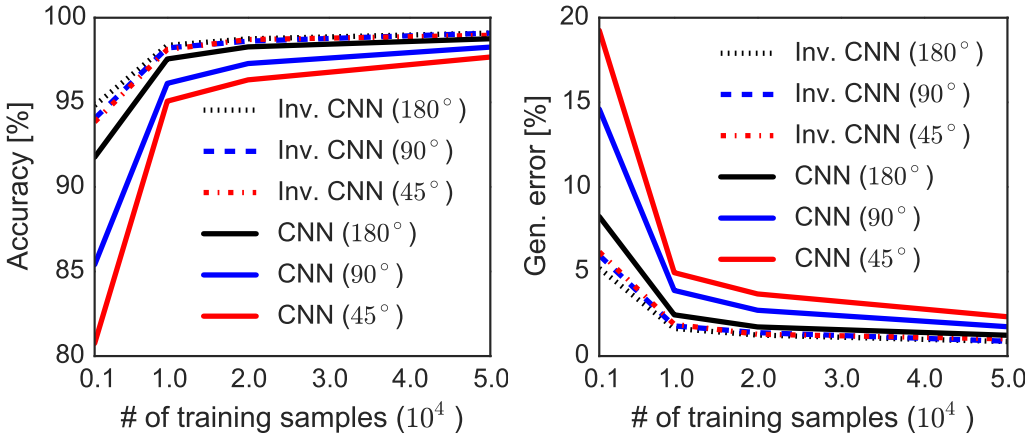
Setup

We use a 7 layer CNN architecture: $(32, 5, 5)$-conv, $(2, 2)$-max-pool, $(64, 5, 5)$-conv, $(2, 2)$-max-pool, $(128, 5, 5)$-conv followed by a global average pooling layer and a softmax layer, where (k, u, v)-conv denotes the convolutional layer with k filters of size $u \times v$, and (p, p)-max-pool denotes the max-pooling layer with pooling regions of size $p \times p$. The rotation invariant CNN is the same as the conventional CNN, but it includes a cyclic slice layer before the first convolutional layer and a cyclic pool layer before the softmax layer. Both, the cyclic slice layer and the cyclic pool layer were proposed in [40] and together they ensure that the CNN is invariant to rotations. In particular, the cyclic slice layer takes input image $\mathbf{x}$ and creates copies of $\mathbf{x}$, each rotated for $r \cdot D^\circ$, $r = 0, 1, 2, \ldots, 360/D - 1$, where D is the same as in the dataset MNIST-D. The copies are then passed through the CNN independently. At the end of the CNN, before the softmax layer, the outputs of the copies are averaged by a cyclic pool layer to obtain a rotation invariant representation.



(a) Original. (b) Rotated. (c) Invariant.

Figure 4.4: Examples of digits from the MNIST dataset (a), rotated digits by a random multiple of 90° (b), and the invariant dataset created by averaging of all 4 rotations ($\{0°, 90°, 180°, 270°\}$) of a single digit (c).



(a) Test set classification accuracy (b) Generalization error.

Figure 4.5: Classification accuracy (a) and the GE (b) of the rotation invariant CNN and the conventional CNN on the rotated MNIST dataset.

The networks are trained using the Stochastic Gradient Descent (SGD) with momentum, which was set to 0.9. The training objective is the standard Categorical Cross Entropy (CCE) loss. Batch size was set to 32 and learning rate was set to 0.01 and reduced by 10 after 100 epochs. The networks were trained for 150 epochs in total. We used training sets of sizes 10^3, 10^4, $2 \cdot 10^4$, $5 \cdot 10^4$.

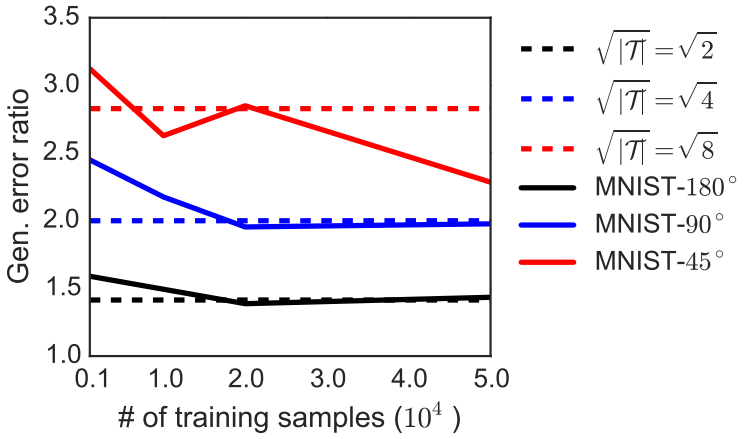


Figure 4.6: The ratio of the GEs of the rotation invariant CNN and the conventional CNN on the rotated MNIST datasets.

Results

The classification accuracies are reported in Figure 4.5(a) and the GE is reported in Figure 4.5(b). We may note that the (explicitly) rotation invariant CNN always has a higher classification accuracy than the conventional CNN. Moreover, the GE of the rotation invariant CNN is much smaller than the GE of the conventional CNN. The difference is most significant when the training set is small, which demonstrates the importance of invariance for the generalization of DNNs.

Note also that the GE of the rotation invariant CNNs on different datasets MNIST-D, $D = 180, 90, 45$, is roughly the same, whereas the conventional CNNs have a higher GE on the datasets with a smaller D. This can be explained by the fact that the rotated MNIST dataset with a smaller D is more complex due to the larger number of rotations. The sizes of the transformation sets for $D = 180, 90, 45$ are $2, 4$ and 8, respectively. Given the theoretical analysis in this chapter, we expect that the ratio of the GEs of an invariant and a non-invariant CNNs will be approximately equal to $\sqrt{|\mathcal{T}|}$. The actual GE ratios are shown in Figure 4.6. We can observe that the GE ratios obtained empirically roughly follow our predictions. However, when the training set is small, the conventional CNN generalizes worse than predicted by our theory and when the training set is large, the conventional CNN generalizes better than predicted by our theory. We conjecture that the conventional CNNs learn to be "partially" invariant when the number of training samples is large. Moreover, the current theory might not capture the relationship between invariant and non-invariant CNNs entirely.

Finally, we have also consider the rotation invariant MNIST dataset, where each

image $\mathbf{x}$ in the dataset is rotated by $r \cdot D^{\circ}$, $r \in \{0, 1, 2, \ldots, 360/D - 1\}$ and the $360/D$ copies are averaged to obtain a sample (see Figure 4.4(c)). As our theory suggests, the rotation invariant CNNs in this case do not have a lower GE than a conventional CNN because the dataset itself is rotation invariant. In fact, given the rotation invariant MNIST dataset, the rotation invariant CNN and the conventional CNN are equivalent. This can be easily established by observing that the cyclic slicing layer produces copies of the input that are identical. We have verified empirically that the rotation invariant and the non-invariant CNNs perform the same on the rotation invariant MNIST dataset.

4.4 Summary

In this chapter we have formally demonstrated that the GE of an invariant DNN can be much smaller than the GE of a non-invariant DNN, provided that the input space can be factorized into a product of a transformation set and a base space, where the covering number of the base space is much smaller than the covering number of the input space.

The next chapter will capitalize on the insights of this chapter and Chapter 3 to explore how to regularize DNNs in order to improve their robustness and invariance with the goal of achieving a lower GE.

Chapter 5

Jacobian and Invariance Regularizers for Deep Neural Networks

In Chapter 3 we have established novel theoretical foundations for the Generalization Error (GE) of Deep Neural Networks (DNNs) based on their classification margin. We have extended this theory to also take into account the symmetries in the data and the possible invariance of the DNN in Chapter 4. In this chapter we use these results to motivate two methods for DNN regularization that reduce the GE in practical applications.

First, we introduce the Jacobian regularizer, which is motivated by the classification margin bounds in Chapter 3. Second, we introduce the Invariance regularizer, which is motivated by the DNN invariance assumption in Chapter 4.

5.1 Jacobian Regularizer

Recall that according to Theorem 3.3 the classification margin $\gamma^d(s_i)$ of a training sample $s_i = (\mathbf{x}_i, y_i)$ can be lower bounded by

$$\gamma^d(s_i) \geq \frac{o(s_i)}{\sup_{\mathbf{x}:\|\mathbf{x}-\mathbf{x}_i\|_2 \leq \gamma^d(s_i)} \|\mathbf{J}(\mathbf{x})\|_2}, \tag{5.1}$$

where $o(s_i)$ is the score of the training sample s_i and $\mathbf{J}(\mathbf{x})$ is the Jacobian Matrix (JM) of the DNN.

The goal of training is to maximize the score $o(s_i)$ of all training samples, while ensuring that the DNN will generalize well. Based on the classification margin bound in (5.1), we suggest to regularize a DNN by bounding the norm of the network's JM

for the inputs close to $\mathbf{x}_i$. The intuition here is that while the loss function promotes the separation of the training samples at the network output, the regularization of the network's JM ensures that this separation at the output of the network will to a large classification margin, which is important for robustness and generalization.

We propose to penalize the norm of the network's JM evaluated at each training sample $\mathbf{x}_i$ by using the regularizer:

$$R_J(\Theta, S_m) = \frac{1}{m}\sum_{i=1}^{m} \|\mathbf{J}(\mathbf{x}_i)\|_2^2, \tag{5.2}$$

where Θ is the set of DNN parameters and $S_m = \{s_i\}_{i=1}^m = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$ is the training set. The implementation of such regularizer requires computation of its gradients or subgradients. In this case the computation of the subgradient of the spectral norm requires the calculation of a SVD [104], which makes the proposed regularizer computationally inefficient. To circumvent this, we propose a surrogate regularizer based on the Frobenius norm of the Jacobian matrix:

$$R_{JF}(\Theta, S_m) = \frac{1}{m}\sum_{i=1}^{m} \|\mathbf{J}(\mathbf{x}_i)\|_F^2. \tag{5.3}$$

Note that the Frobenius norm and the spectral norm can be bounded as follows:

$$1/\text{rank}(\mathbf{J}(\mathbf{x}_i))\|\mathbf{J}(\mathbf{x}_i)\|_F^2 \leq \|\mathbf{J}(\mathbf{x}_i)\|_2^2 \leq \|\mathbf{J}(\mathbf{x}_i)\|_F^2 . \tag{5.4}$$

The choice of the Frobenius norm instead of the spectral norm is justified by the fact that the Frobenius norm upper bounds the spectral norm. We will refer to $R_{JF}(\Theta, S_m)$ as the Jacobian regularizer.

5.1.1 Computation of Gradients and Efficient Implementation

Recall that a DNN f that maps an input space $\mathcal{X}$ to an output space $\mathcal{Z}$: $f : \mathcal{X} \to \mathcal{Z}$, where the input space is a subset of the real N dimensional space $\mathbb{R}^N$: $\mathcal{X} \subseteq \mathbb{R}^N$ and the output space $\mathcal{Z}$ is the real $N_\mathcal{Y}$ dimensional space: $\mathcal{Z} = \mathbb{R}^{N_\mathcal{Y}}$, where $N_\mathcal{Y}$ represents the number of classes. A L layer DNN is obtained as

$$f(\mathbf{x}) = \phi_L(\phi_{L-1}(\cdots \phi_1(\mathbf{x}, \theta_1), \cdots \theta_{L-1}), \theta_L), \tag{5.5}$$

where

$$\mathbf{z}^l = \phi_l(\mathbf{z}^{l-1}, \theta_l) \tag{5.6}$$

represents the l-th layer with parameters θ_l, output $\mathbf{z}^l \in \mathbb{R}^{M_l}$ and input $\mathbf{z}^{l-1} \in \mathbb{R}^{M_{l-1}}$, $l = 1, \ldots, L$. The input layer corresponds to $\mathbf{z}^0 = \mathbf{x}$ and the output of the last layer is denoted by $\mathbf{z} = f(\mathbf{x})$. A typical network layer has the form

$$\mathbf{z}^l = [\hat{\mathbf{z}}^l]_\sigma = [\mathbf{W}_l \mathbf{z}^{l-1} + \mathbf{b}_l]_\sigma \,, \tag{5.7}$$

where $[\hat{\mathbf{z}}^l]_\sigma$ represents the element-wise non-linearity σ applied to each element of $\hat{\mathbf{z}}^l \in \mathbb{R}^{M_l}$, and $\hat{\mathbf{z}}^l$ represents the affine transformation of the layer input: $\hat{\mathbf{z}}^l = \mathbf{W}_l \mathbf{z}^{l-1} + \mathbf{b}_l$. The layer parameters $\theta_l = \{\mathbf{W}_l, \mathbf{b}_l\}$ are the weight matrix $\mathbf{W}_l \in \mathbb{R}^{M_l \times M_{l-1}}$ and the bias vector $\mathbf{b}_l \in \mathbb{R}^{M_l}$.

The JM of a DNN f is defined as

$$\mathbf{J}(\mathbf{x}) = \frac{df(\mathbf{x})}{d\mathbf{x}} = \prod_{l=0}^{L} \frac{d\phi_l(\mathbf{z}^{l-1})}{d\mathbf{z}^{l-1}} = \prod_{l=1}^{L} \frac{d\phi_l(\mathbf{z}^{l-1})}{d\mathbf{z}^{l-1}} \cdot \frac{d\phi_1(\mathbf{x})}{d\mathbf{x}}, \tag{5.8}$$

where we recall that $\mathbf{z}^0 = \mathbf{x}$ and corresponds to the DNN input. The k-th row of $\mathbf{J}(\mathbf{x}_i)$ corresponds to the gradient of the k-th element of $f(\mathbf{x})$, i.e., $(f(\mathbf{x}))_k$, $k = 1, \ldots, N_\mathcal{Y}$, with respect to the input $\mathbf{x}$ evaluated at $\mathbf{x}_i$ and it is denoted by $\mathbf{g}_k(\mathbf{x}_i) = \frac{d(f(\mathbf{x}))_k}{d\mathbf{x}}|_{\mathbf{x}=\mathbf{x}_i}$. Now we can write

$$R_{JF}(\Theta, S_m) = \frac{1}{m} \sum_{i=1}^{m} \sum_{k=1}^{N_\mathcal{Y}} \mathbf{g}_k(\mathbf{x}_i) \mathbf{g}_k(\mathbf{x}_i)^T \,. \tag{5.9}$$

As the regularizer will be minimized by a gradient descent algorithm we need to compute its gradient with respect to the DNN parameters. First, we express $\mathbf{g}_k(\mathbf{x}_i)$ as

$$\mathbf{g}_k(\mathbf{x}_i) = \mathbf{g}_k^l(\mathbf{x}_i) \mathbf{W}_l \mathbf{J}^{l-1}(\mathbf{x}_i) \tag{5.10}$$

where $\mathbf{g}_k^l(\mathbf{x}_i) = \frac{d(f(\mathbf{x}))_k}{d\hat{\mathbf{z}}^l}|_{\mathbf{x}=\mathbf{x}_i}$ is the gradient of $(f(\mathbf{x}))_k$ with respect to $\hat{\mathbf{z}}^l$ evaluated at the input $\mathbf{x}_i$ and $\mathbf{J}^{l-1}(\mathbf{x}_i) = \frac{d\mathbf{z}^{l-1}}{d\mathbf{x}}|_{\mathbf{x}=\mathbf{x}_i}$ is the JM of $l-1$-th layer output $\mathbf{z}^{l-1}$ evaluated

at the input $\mathbf{x}_i$. The gradient of $\mathbf{g}_k(\mathbf{x}_i)\mathbf{g}_k(\mathbf{x}_i)^T$ with respect to $\mathbf{W}_l$ is then given by [105]

$$\nabla_{\mathbf{W}_l}\left(\mathbf{g}_k(\mathbf{x}_i)\mathbf{g}_k(\mathbf{x}_i)^T\right) = 2\mathbf{g}_k^l(\mathbf{x}_i)^T\mathbf{g}_k^l(\mathbf{x}_i)\mathbf{W}_l\mathbf{J}^{l-1}(\mathbf{x}_i).$$

The gradient of $\mathbf{g}_k(\mathbf{x}_i)\mathbf{g}_k(\mathbf{x}_i)^T$ with respect to $\mathbf{b}_l$ is zero. The computation of the gradient of the regularizer at layer l requires the computation of gradients $\mathbf{g}_k^l(\mathbf{x}_i)$, $k = 1,\ldots,N_y$, $i = 1,\ldots,m$, and the computation of the Jacobian matrices $\mathbf{J}^{l-1}(\mathbf{x}_i)$, $i = 1,\ldots,m$. The computation of the gradient of a typical loss used for training DNN usually involves a computation of m gradients with computational complexity similar to the computational complexity of $\mathbf{g}_k^l(\mathbf{x}_i)$. Therefore, the computation of gradients required for an implementation of the Jacobian regularizer can be very expensive.

5.1.1.1 Per-layer Jacobian Regularizer

To avoid excessive computational complexity we propose a simplified version of the regularizer (5.3), which we name per-layer Jacobian regularizer. The per-layer Jacobian regularizer is defined as

$$R_{JF}^l(\Theta, S_m) = \frac{1}{m}\sum_{l=1}^{L}\sum_{i=1}^{m}\tilde{\mathbf{g}}_{\pi(i)}^{l-1}(\mathbf{x}_i)(\tilde{\mathbf{g}}_{\pi(i)}^{l-1}(\mathbf{x}_i))^T, \tag{5.11}$$

where $\tilde{\mathbf{g}}_{\pi(i)}^{l-1}(\mathbf{x}_i) = \frac{d(f(\mathbf{x}))_{\pi(i)}}{d\mathbf{z}^{l-1}}|_{\mathbf{x}=\mathbf{x}_i}$, and $\pi(i) \in \{1,\ldots,N_y\}$ is a random index and L is the number of layers. Compared to (5.3) we have made two simplifications. First, we assumed that input of layer l is fixed. This way we do not need to compute the JM $\mathbf{J}^{l-1}(\mathbf{x}_i)$ between the output of the layer $l-1$ and the input. Second, by choosing only one index $\pi(i)$ per training sample we have to compute only one additional gradient per training sample. This significantly reduces the computational complexity. The gradient of $\tilde{\mathbf{g}}_{\pi(i)}^{l-1}(\mathbf{x}_i)(\tilde{\mathbf{g}}_{\pi(i)}^{l-1}(\mathbf{x}_i))^T$ with respect to $\mathbf{W}_l$ is simply

$$\nabla_{\mathbf{W}_l}\left(\tilde{\mathbf{g}}_{\pi(i)}^{l-1}(\mathbf{x}_i)(\tilde{\mathbf{g}}_{\pi(i)}^{l-1}(\mathbf{x}_i))^T\right) = 2\mathbf{g}_{\pi(i)}^l(\mathbf{x}_i)^T\mathbf{g}_{\pi(i)}^l(\mathbf{x}_i)\mathbf{W}_l.$$

The gradient of $\tilde{\mathbf{g}}_{\pi(i)}^{l-1}(\mathbf{x}_i)(\tilde{\mathbf{g}}_{\pi(i)}^{l-1}(\mathbf{x}_i))^T$ with respect to $\mathbf{b}_l$ is zero. We demonstrate the effectiveness of these regularizers next.

5.2 Experimental Evaluation of the Jacobian Regularizer

We now demonstrate the effectievness of the Jacobian regularizer in (5.3) and the per-layer Jacobian regularizer in (5.11) with a series of experiments on the MNIST [103], CIFAR-10 [106], LaRED [107] and ImageNet (ILSVRC2012) [108] datasets. The Jacobian regularizer is applied to various DNN architectures such as feed-forward DNN, Convolutional Neural Network (CNN) and Residual Network [29]. We use the ReLU non-linearity in all considered DNNs as this is currently the most commonly used non-linearity.

The objective function in all experiments is of the form:

$$\mathcal{L}(\Theta, S_m) = \ell_{\text{emp}}(\Theta, S_m) + \lambda R(\Theta, S_m)\,, \tag{5.12}$$

where Θ is the set of DNN parameters, S_m is the training set, $\ell_{\text{emp}}(\Theta, S_m)$ is the empirical loss and $R(\Theta, S_m)$ is one of the regularizers: the weight decay in (2.50), the Jacobian regularizer in (5.3) or the per-layer Jacobian regularizer in (5.11). The regularization factor λ balances the contributions of the loss and the regularizer in the objective function.

5.2.1 Feed-forward Deep Neural Networks

First, we compare standard feed-forward DNNs trained with the weight decay and with the Jacobian regularizer (5.3) on the MNIST and CIFAR-10 datasets.

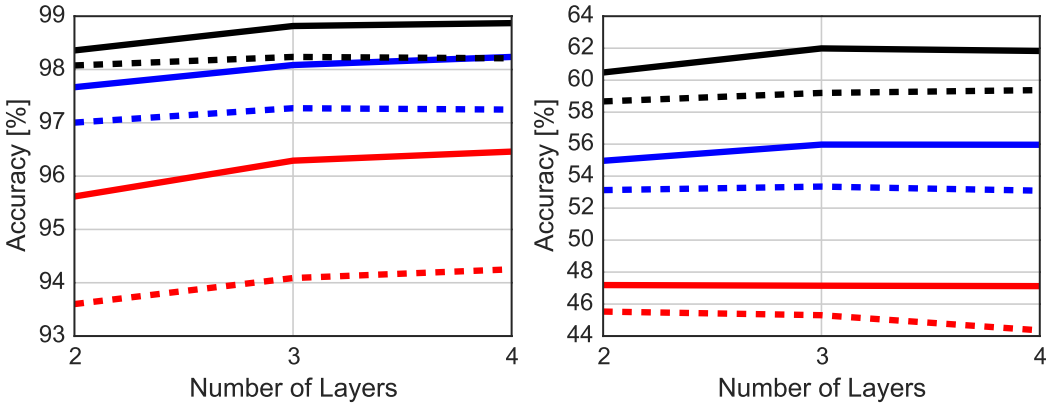
Setup

Different number of training samples are used (5000, 20000, 50000). We consider DNNs with 2, 3 and 4 fully connected layers where all layers, except the last one, have dimension equal to the input signal dimension, which is 784 in case of MNIST and 3072 in case of CIFAR-10. The last layer is always the softmax layer and the objective is the Categorical Cross Entropy (CCE) loss. The networks were trained using the Stochastic Gradient Descent (SGD) with momentum, which was set to 0.9. Batch size was set to 128 and learning rate was set to 0.01 and reduced by a factor 10 after every 40 epochs. The networks were trained for 120 epochs in total. The weight decay and the Jacobian regularization factors λ were chosen on a separate validation set. The

experiments were repeated with the same regularization parameters on 5 random draws of training sets and weight matrix initializations.

Results

Classification accuracies averaged over different experimental runs are shown in Figure 5.1. We observe that the proposed Jacobian regularizer always outperforms the



(a) MNIST (b) CIFAR-10

Figure 5.1: Classification accuracy of DNNs trained with the Jacobian regularizer (solid lines) and the weight decay (dashed lines). Different numbers of training samples are used: 5000 (red), 20000 (blue) and 50000 (black).

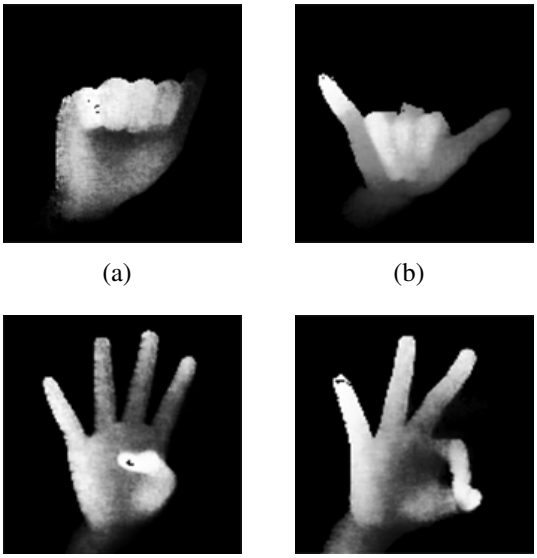
weight decay. This is also in-line with the theoretical results in Section 3.3.3, which predict that the JM is crucial for the control of (the bound to) the GE. Interestingly, in the case of MNIST, a 4 layer DNN trained with the Jacobian regularizer and using 20000 training samples (solid blue line if Figure 5.1 (a)) performs on par with the DNN trained with the weight decay and using 50000 training samples (dashed black line Figure 5.1 (a)), which means that using the Jacobian regularizer can lead to the same performance with significantly less training samples.

5.2.2 Convolutional Neural Networks

In this section we compare the performance of CNNs regularized with the Jacobian regularizer or the weight decay. We also show that the Jacobian regularizer can be applied to batch normalized DNNs. We will use the standard MNIST and CIFAR-10 dataset and the LaRED dataset which is briefly described below.

The LaRED dataset contains depth images of 81 distinct hand gestures performed by 10 subjects with approximately 300 images of each gesture per subject. We extracted the depth images of the hands using the masks provided in [107] and resized the images

to 32×32. Examples of depth images are shown in Figure 5.2. The images of the first 6 subjects were used to create non-overlapping training and testing sets. In addition we also constructed a testing set composed from the images of the last 4 subjects in the dataset in order to test generalization across different subjects. The goal is classification of gestures based on the depth image.



(a) (b)

(c) (d)

Figure 5.2: Example of depth images of different gestures in the LaRED dataset.

5.2.2.1 Comparison of the Jacobian Regularizer and the Weight Decay

Setup

We use a 4 layer CNN with the following architecture: $(32, 5, 5)$-conv, $(2, 2)$-max-pool, $(32, 5, 5)$-conv, $(2, 2)$-max-pool followed by a softmax layer, where (k, u, v)-conv denotes the convolutional layer with k filters of size $u \times v$ and (p, p)-max-pool denotes max-pooling with pooling regions of size $p \times p$. The training procedure follows the one described in the previous paragraphs.

Results

The results are reported in Table 5.1 and Figure 5.3 for MNIST and in Tables 5.2 and 5.3, and in Figure 5.4 for LaRED. We observe that training with the Jacobian regularizer outperforms the weight decay in all cases. This is most obvious at smaller training set sizes. For example, on the MNIST dataset, the CNN trained using 1000 training samples and regularized with the weight decay achieves classification accuracy of 94%

and the CNN trained with the Jacobian regularizer achieves classification accuracy of 96.3%.

Table 5.1: Classification accuracy [%] of CNN on MINST dataset.

# train samples	weight decay	Jacobian reg.	Per-layer Jacobian reg.
1000	94.00	**96.03**	94.81
5000	97.59	**98.20**	97.74
20000	98.60	**99.00**	98.77
50000	99.10	**99.35**	99.12

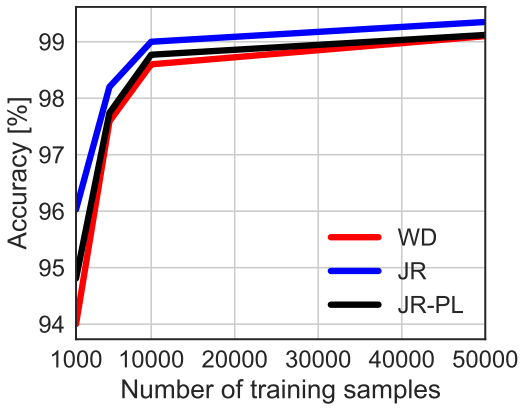


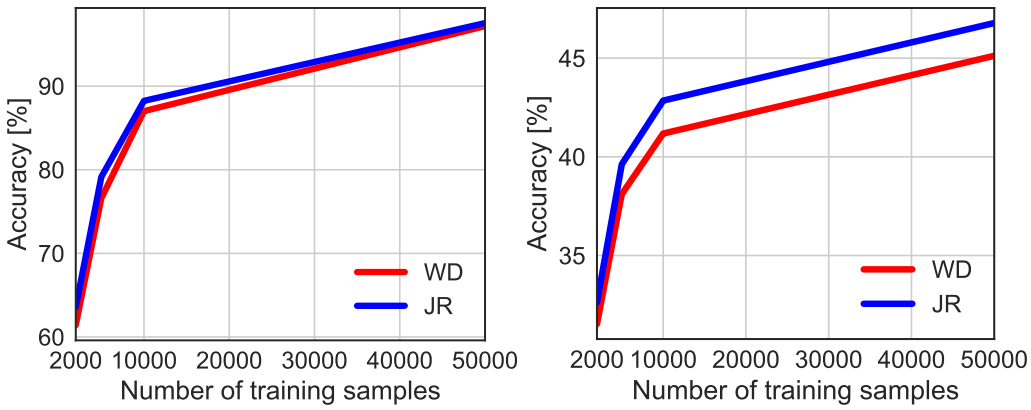
Figure 5.3: Classification accuracy of CNN on MNIST dataset. Different regularizers are used: weight decay (WD), Jacobian regularizer (JR) and per-layer Jacobian regularizer (JR-PL).

Table 5.2: Classification accuracy [%] of CNN on LaRED dataset using the same test subjects.

# train samples	weight decay	Jacobian reg.
2000	61.40	**63.56**
5000	76.59	**79.14**
10000	87.01	**88.24**
50000	97.18	**97.54**

Table 5.3: Classification accuracy [%] of CNN on LaRED dataset using different test subjects.

# train samples	weight decay	Jacobian reg.
2000	31.53	**32.62**
5000	38.11	**39.62**
10000	41.18	**42.85**
50000	45.12	**46.78**



(a) Training and testing set contain the same subjects.

(b) Training and testing set contain different subjects.

Figure 5.4: Classification accuracy of CNN on LaRED dataset. Different regularizers are used: weight decay (WD) and Jacobian regularization (JR).

Similarly, on the LaRED dataset, the Jacobian regularizer outperforms the weight decay with the difference most obvious at the smallest number of training samples. Note also that the generalization of the network to the subjects outside the training set is not very good; i.e., using 50000 training samples the classification accuracy on the testing set containing the same subjects is higher than 97% whereas the classification accuracy on the testing set containing different subjects is only 46%. The reason for this is that the samples from the training set do not reflect well the samples in the testing set as they are coming from different subjects. Nevertheless, the Jacobian regularizer outperforms the weight decay also on this testing set by a small margin.

Comparison of Jacobian and Per-layer Jacobian regularizers

Finally, we also compare the performance of the Jacobian regularizer (see (5.3)) and the per-layer Jacobian regularizer (see (5.11)) on the MNIST dataset. The results are reported in Table 5.1 and in Figure 5.3. We can observe that the Per-layer Jacobian regularizer still outperforms weight decay. However, it does not perform as well as the Jacobian regularizer. This is expected and can be explained by the fact that the Per-layer Jacobian regularizer only approximates the Jacobian regularizer.

5.2.2.2 Batch Normalization and the Jacobian Regularizer

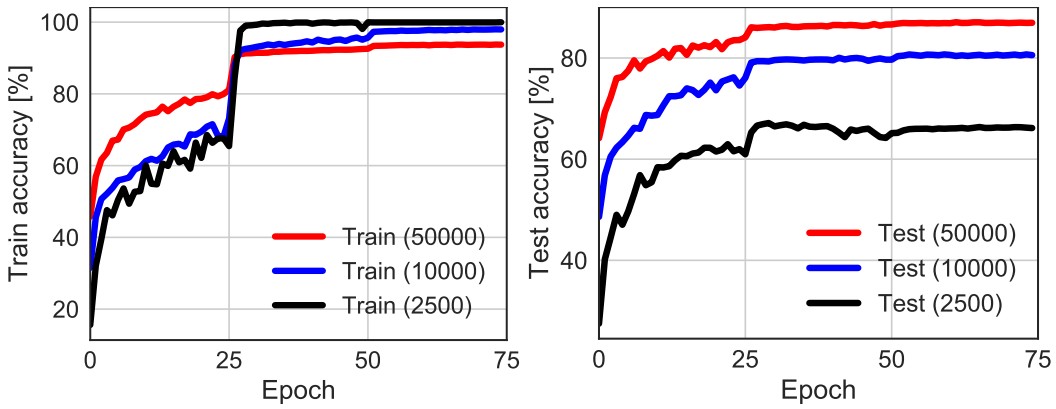
Now we show that the Jacobian regularization (5.3) can also be applied to a batch normalized DNN. Note that we have shown in Section 3.3.3 that the batch normalization has an effect of normalizing the rows of the weight matrices.

Table 5.4: Classification accuracy [%] of CNN on CIFAR-10 dataset.

# train samples	batch norm.	batch norm. + Jacobian reg.
2500	60.86	**66.15**
10000	76.35	**80.57**
50000	87.44	**88.95**

Setup

We use the CIFAR-10 dataset and the All-convolutional-DNN proposed in [109] (All-CNN-C) with 9 convolutional layers, an average pooling layer and a softmax layer. All the convolutional layers are batch normalized and the softmax layer is weight normalized. The networks were trained using the SGD with momentum, which was set to 0.9. Batch size was set to 64 and the learning rate was set to 0.1 and reduced by a factor 10 after every 25 epochs. The networks were trained for 75 epochs in total, which is sufficient for convergence as shown in Figure 5.5.



(a) Training set accuracy. (b) Testing set accuracy.

Figure 5.5: Training set and testing set accuracies during training of All-convolutional-DNN on CIFAR-10 dataset with 2500 (black), 10000 (blue) and 50000 (red) training samples.

Results

The classification accuracy results are presented in Table 5.4 for different sizes of training sets (2500, 10000, 50000). We can observe that the Jacobian regularization also leads to a smaller GE in this case. Note that in the case of normalized weight matrix rows the weight decay is not meaningful.

Table 5.5: Classification accuracy [%] of Residual Network on CIFAR-10 dataset.

# train samples	Residual Network	Residual Network + Jacobian reg.
2500	55.69	**62.79**
10000	71.79	**78.70**
50000 + aug.	93.34	**94.32**

5.2.3 Residual Networks

Now we demonstrate that the Jacobian regularizer is also effective when applied to Residual Networks. All Residual Networks in this section use the batch normalization [62]. We use the CIFAR-10 and ImageNet datasets. We use the per-layer Jacobian regularizer (5.11) for experiments in this section, due to its computational efficiency.

5.2.3.1 CIFAR-10

Setup

The Wide Residual Network architecture proposed in [110], which follows [38], but proposes wider and shallower networks which leads to the same or better performance than deeper and thinner networks, is used here. In particular, we use the Residual Network with 22 layers of width 5.

We follow the data normalization process of [110]. We also follow the training procedure of [110] except for the learning rate and use the learning rate sequence: (0.01, 5), (0.05, 20), (0.005, 40), (0.0005, 40), (0.00005, 20), where the first number in parenthesis corresponds to the learning rate and the second number corresponds to the number of epochs. We train Residual Network on small training sets (2500 and 10000 training samples) without augmentation and on the full training set with the data augmentation as in [110]. The regularization factor λ was set to 1 and 0.1 for the smaller training sets (2500 and 10000) and the full augmented training set, respectively.

Results

The results are presented in Table 5.5. In all cases the Residual Network trained with the Jacobian regularizer outperforms the baseline Residual Network. The effect of regularization is the strongest with the smaller number of training samples, as expected.

5.2.3.2 ImageNet

Setup

We use the 18 layer Residual Network [29] with identity connection [38]. The training

procedure follows [29] with the learning rate sequence: (0.1, 30), (0.01, 30), (0.001, 30). The Jacobian regularization factor λ was set to 1.

The images in the dataset are resized to 128×128. We run an experiment without data augmentation and with data augmentation following [27], which includes random cropping of images of size 112×112 from the original image and colour augmentation.

Results

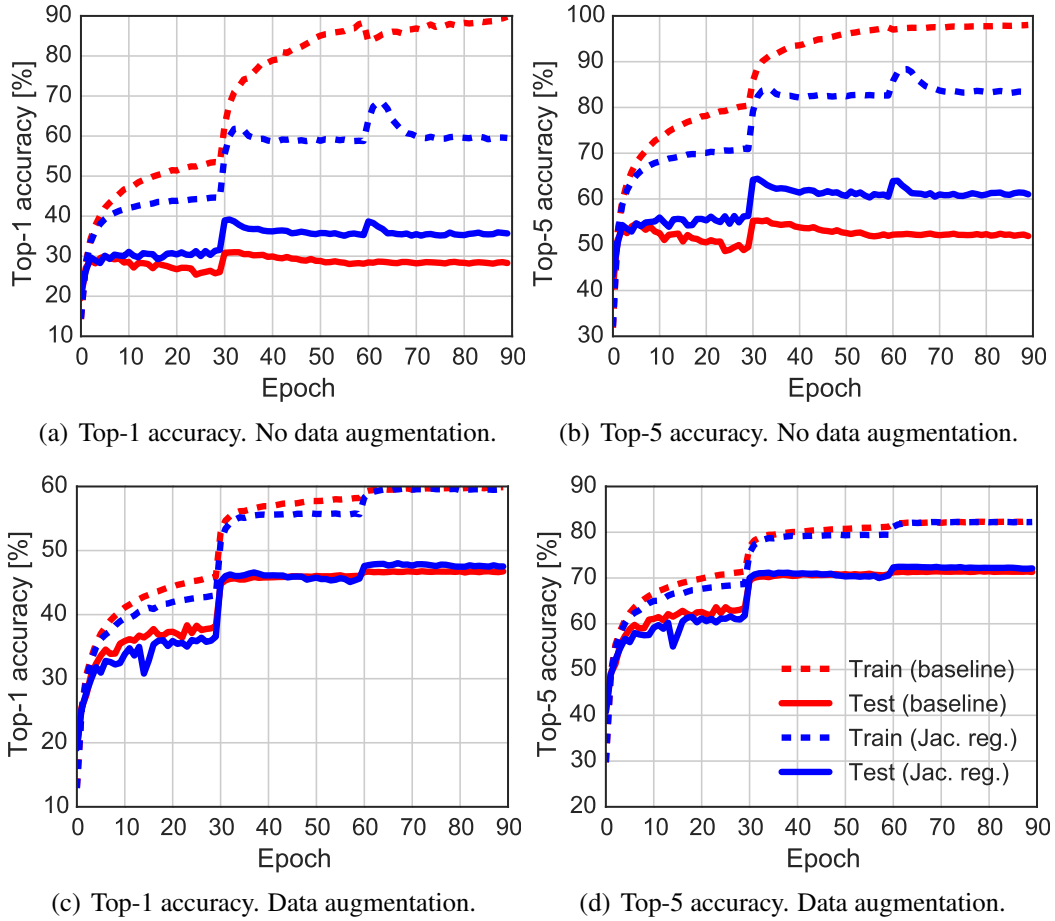
The classification accuracies during training are shown in Figure 5.6 and the final results are reported in Table 5.6. We first focus on training without data augmentation. The Residual Network trained with the Jacobian regularizer has a much smaller GE (23.83%) compared to the baseline Residual Network (61.53%). This again demonstrates that the Jacobian regularizer decreases the GE, and is in-line with the theory developed in Chapter 3. Note that the smaller GE of the Residual Network trained with the Jacobian regularizer partially transfers to a higher classification accuracy on the testing set. However, in practice DNNs are often trained with data augmentation, as discussed in Section 2.2.1.5. In this case the GE of a baseline Residual Network is much lower (13.14%) and is very close to the GE of the Residual Network trained with the Jacobian regularizer (12.03%). It is clear that data augmentation reduces the need for strong regularization. Nevertheless, note that the Residual Network trained with the Jacobian regularizer achieves a slightly higher testing set accuracy (47.51%) compared to the baseline Residual Network (46.75%).

Table 5.6: Classification accuracy [%] and GE [%] of Residual Network on ImageNet dataset.

Setup	Train	Test	GE
Baseline	89.82	28.29	61.53
Baseline + Jac. reg.	59.52	35.69	23.83
Baseline + aug.	59.89	46.75	13.14
Baseline + aug + Jac. reg.	59.54	**47.51**	**12.03**

5.2.4 Computational Time

Finally, we measure how the use of the Jacobian regularizer or the per-layer Jacobian regularizer affects training time of DNNs. We have implemented DNNs in Theano [111], which includes automatic differentiation and computation graph optimization, and is designed to run efficiently on Graphical Processing Units (GPUs). The experi-



(a) Top-1 accuracy. No data augmentation.

(b) Top-5 accuracy. No data augmentation.

(c) Top-1 accuracy. Data augmentation.

(d) Top-5 accuracy. Data augmentation.

Figure 5.6: Training set (dashed) and testing set (solid) classification accuracies during training. Blue curves correspond to the Residual Network trained with the Jacobian regularizer and red curves correspond to the baseline Residual Network. Top-1 and top-5 classification accuracies are reported for training without data augmentation (a,b) and for training with data augmentation (c,d).

ments are run on the Titan X GPU.

First, we compare the average computational time per batch for the CNN on the MNIST dataset in Section 5.2.2.1, where we have used the weight decay regularizer, the Jacobian regularizer and the per-layer Jacobian regularizer. The measurements are reported in Table 5.7. We can observe that using the Jacobian regularizer significantly increases the computational time (825%). This may not be critical if the number of training samples is small. On the other hand, the per-layer Jacobian regularizer has a much smaller cost (computational time increase of 75%). Due to its efficiency the per-layer Jacobian regularizer is more appropriate for large scale experiments where computational time is critical.

We also report the average batch processing time of the Residual Network on the

Table 5.7: Average batch processing time of a CNN on the MNIST dataset.

	Batch processing time [ms]	Increase w.r.t. weight decay
Weight decay	2.74	/
Jacobian reg.	25.35	825%
Per-layer Jacobian reg.	4.82	75%

ImageNet dataset in Section 5.2.3.2, where the per-layer Jacobian regularizer was used. Since the ImageNet dataset is much larger than the MNIST dataset the computational time of training is more critical. This is the reason the per-layer Jacobian regularizer was used in this experiments. The computational time measurements are reported in Table 5.8. We can observe that the per-layer Jacobian regularizer leads to only 58% increase in computation time on the ImageNet dataset compared to the baseline.

Table 5.8: Average batch processing time of a ResNet on ImageNet dataset.

	Batch processing time [s]	Increase w.r.t. baseline
Baseline	0.120	/
Per-layer Jacobian reg.	0.190	58%

5.3 Invariance Regularizer

In Chapter 4 we have focused on a scenario where there exists a set of transformations $\mathcal{T} = \{t_i\}_{i=0}^{T-1}$ that applied to an input $\mathbf{x}$, i.e., $t(\mathbf{x})$, $t \in \mathcal{T}$, do not change the class label. We have also defined the invariant DNN for which it holds:

$$f(t_i(\mathbf{x})) = f(t_j(\mathbf{x})) \quad \forall \mathbf{x} \in \mathcal{X}_0, \forall t_i, t_j \in \mathcal{T}\,. \tag{5.13}$$

The theory in Chapter 4 suggests that invariance of a DNN improves its generalization when there are symmetries present in the data. Invariance may be achieved in some cases by an appropriately designed DNN. However, it may be the case that we do not know how to design a DNN that is invariant or want to use an existing (non-invariant) DNN architecture. To achieve approximate invariance of such DNNs we propose the following invariance regularizer

$$R_I(\Theta, S_m) = \sum_{i=1}^{m} \sum_{t \in \mathcal{T}} \|f(\mathbf{x}_i) - f(t(\mathbf{x}_i))\|_2^2\,, \tag{5.14}$$

where Θ is the set of DNN parameters and S_m is the training set. The invariance regularizer promotes approximate invariance of a DNN by enforcing a similar representations at the DNN output (measured by the ℓ_2-norm) for a sample transformed by different transformations in $\mathcal{T}$. Note also that data augmentation, which is often used when training DNNs, does not apply explicit invariance regularization in (5.14).

5.4 Experimental Evaluation of the Invariance Regularizer

We now evaluate the invariance regularizer of the CIFAR-10 dataset [106].

The objective function in all experiments is of the form:

$$\mathcal{L}(\Theta, S_m) = \ell_{\text{emp}}(\Theta, S_m) + \lambda_J R^l_{JF}(\Theta, S_m) + \lambda_I R_I(\Theta, S_m)\,, \tag{5.15}$$

where Θ is the set of DNN parameters, S_m is the training set, $\ell_{\text{emp}}(\Theta, S_m)$ is the empirical loss, $R^l_{JF}(\Theta, S_m)$ is the per-layer Jacobian regularizer (5.11) and $R_I(\Theta, S_m)$ is the invariance regularizer (5.14). The regularization factors λ_J and λ_I balance the contributions of the per-layer Jacobian regularizer and the invariance regularizer, respectively.

Setup

The CIFAR-10 dataset is normalized following [110], and the Wide Residual Network [110] with 13 layers of width 5 is used. The networks are trained using SGD and the learning rate is set to 0.01 for the first epoch and then to 0.05, 0.005 and 0.0005, each for 30 epochs. We use 10^3, 10^4, $2 \cdot 10^4$ and $5 \cdot 10^4$ training samples. The regularization factors λ_J is set to 0.1 with smaller training sets (2500, 5000, 10000) and 0.05 otherwise. Batch size is set to 128.

SGD batches are constructed as follows: the first half of the batch contains images from the training set and the other half of the batch contains transformed versions of the images in the first half of the batch where the transformations are chosen at random. The set of transformations contains shifts of $\pm\{1, 2, 3, 4\}$ pixels and horizontal flips, as in [110]. For the Invariance regularizer we chose to regularize the output of the last global pooling layer instead of the softmax output. Moreover, we do not sum over all transformations in $\mathcal{T}$ as in (5.14), but only over the original sample in the first half of the batch and a random transformation in the second half of the batch. The regularization

factor λ_I in all experiments is set to 10^{-4}.

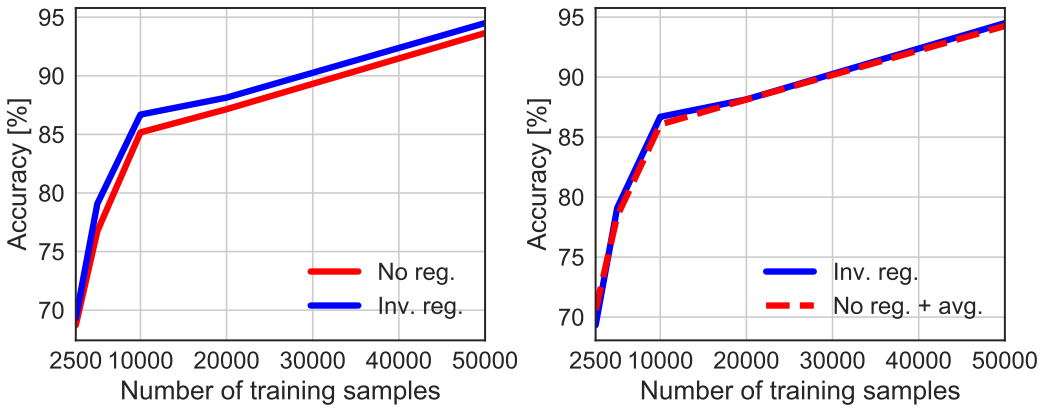
Results

Table 5.9 reports the standard test accuracy and the accuracy of the predictions averaged over the augmented test set (denoted by + avg.), which are obtained as follows: for each test image we average the softmax outputs for the original image, shifted images (9×9 shifts) and horizontally flipped images. Note that this method requires approximately 160 forward passes through a network to obtain a prediction and is often used in practice to improve the robustness of classification, as discussed in Section 2.2.1.5. Results are reported for the case when the invariance regularizer is not used, i.e. $\lambda_I = 0$, and for the case when the Invariance regularizer is used. The results are also plotted in Figure 5.7.

The training set accuracies were 100% or very close to 100% in all cases. First, we observe that invariance regularization leads to a lower GE (a higher accuracy) in all cases. Moreover, testing with predictions averaged over the augmented test set is even more robust and leads to a lower GE for both, the regularized and the non-regularized DNNs. However, DNNs trained with explicit invariance regularization (except when 2500 training samples are used) perform better or on par with a non-regularized network even when the predictions are averaged over the augmented test set (see Figure 5.7(b)). Moreover, averaging over the augmented test set is approximately 160 times more expensive than conventional testing, which requires to evaluate the DNN only once. This experiment verifies the hypothesis that enforcing the network invariance explicitly improves the robustness and leads to a smaller GE.

Table 5.9: Classification accuracy [%] on CIFAR-10 dataset.

	number of training samples				
	2500	5000	10000	20000	50000
No reg. ($\lambda_I = 0$)	68.71	76.74	85.17	87.15	93.65
Inv. Reg.	**69.32**	**79.08**	**86.69**	**88.14**	**94.50**
No reg. ($\lambda_I = 0$) + avg.	70.59	78.40	86.05	88.13	94.26
Inv. Reg. + avg.	**70.71**	**79.65**	**86.96**	**88.98**	**94.78**



(a) Comparison of performance with and without Invariance regularizer.

(b) Comparison of Invariance regularizer and averaging predictions over augmented testing set.

Figure 5.7: Classification accuracies of Wide Residual Network on CIFAR-10 dataset with and without Invariance regularizer and with and without prediction averaging.

5.5 Summary

In this chapter we have proposed and experimentally evaluated the Jacobian regularizer and the invariance regularizer. Both are motivated by the data structure based theory of generalization for DNNs presented in Chapters 3 and 4.

We have shown that the regularization of the network's JM, which in contrast to the weight decay may be combined with the weight normalization or the batch normalization, is an effective regularization method, especially in the limited training data regime, with moderate increase in computational complexity. Similarly, the invariance regularizer reduces the sensitivity of a DNN to input transformations, which consequently leads to a better generalization.

Chapter 6

Conclusions

In this thesis we have focused on the Generalization Error (GE) of Deep Neural Networks (DNNs). In spite of the great success of DNNs in practical applications, the GE of DNNs is one of the many theoretical aspects that are not well understood. We have proposed a novel theoretical approach for bounding the GE that leads to novel insights and regularization methods, and also offers interesting opportunities for further research.

6.1 Summary and Conclusions

We now briefly summarize the thesis.

- We have started the thesis by overviewing the most important work in the literature associated with DNNs in Chapter 2. We have focused on the work that studies the GE of DNNs and noted that many existing bounds do not apply to DNNs, since these GE bounds either scale exponentially with the DNN depth or grow with the number of DNN parameters. Both cases contradict practice where very deep or very wide DNNs have been demonstrated to generalize well [29, 38]. Since the existing bounds use tools that link the GE to the ability of a DNN to fit noise, and very deep or wide DNNs are very powerful approximators, these techniques unsurprisingly provide loose bounds.

- The literature review in Chapter 2 motivated us to seek novel GE bounds that are better aligned with practice. Our proposed GE bounds, which are based on the classification margin and the covering number of the data, are the topic of Chapter 3. The main result can be stated as follows: provided that a DNN achieves

some classification margin on a training set, the GE of the DNN will be proportional to the square root of the ratio of the covering number, where the radius of metric balls is given by the classification margin, and the number of the training samples. Therefore, given a finite covering number of the data, which as we discuss often scales not with the ambient dimension, but rather with the intrinsic dimension of the data, our generalization bounds imply that DNNs of arbitrary depth or width will generalize well. As discussed in Chapter 2, the fact that very deep or wide DNNs generalize well also holds in practice.

- In Chapter 3 we have also provided an analysis of geometrical properties of DNNs. In particular, we have shown that the network's Jacobian Matrix (JM) controls the expansion of the distance between signals at the DNN input and distance between signals at the DNN output. This provides a rationale for training DNNs by promoting separation of training samples at the DNN output while enforcing constraints on the JM in order to ensure that the separation at the DNN output is translated into a large classification margin at the DNN output. A large classification margin leads to a more robust DNN with a lower GE.

- Our proposed GE bounds in Chapter 3 relate the GE to the properties of the data via its covering number. This motivates us to further explore how symmetries in the data affect the GE bounds. This is the topic of Chapter 4, where we model symmetries in the data as a product of a discrete set of transformations and a base set of signals. We also assume that the classification task is invariant to these transformations, i.e., the class label of the input is the same for all transformations, and we define an invariant DNN for which the output for a given input is the same for all possible transformations of this input. We then show, using the tools from Chapter 3, that the GE bound of an invariant DNN does not scale with the covering number of the data, but rather with the covering number of the base set. We also provide a set of conditions on the geometry of the base set and the set of transformations that show that the ratio between the covering number of the data and the covering number of the base space is greater than or equal to the size of the set of transformations. In this case the ratio of GE bounds of a non-invariant DNN and an invariant DNN corresponds to the square root of the

size of the set of transformations. The conclusion is that an invariant DNN may have a much smaller GE than a non-invariant DNN.

- Finally, in Chapter 5 we propose two regularization schemes that are motivated by the theory developed in Chapters 3 and 4. First, we propose the Jacobian regularizer, where we constrain the norm of the JM during training. The intuition that follows from the classification margin bounds in Chapter 3 is the following: the training objective will increase the separation of different classes at the network output, while a bounded norm of the JM ensures that this separation at the output of the network leads to a large classification margin. The experiments show that the proposed method is effective and outperforms the standard weight decay, especially when the number of training samples is small.

  We also propose the invariance regularizer, which is motivated by the definition of the invariant DNN in Chapter 4. The invariance regularizer promotes a similar representation (measured in the Euclidean distance) of the same training sample transformed by different transformations in the set of transformations. We show experimentally that this reduces the sensitivity of the network to transformations of the input and leads to a lower GE.

6.2 Future Work

The work in this thesis also provides many potential future research directions.

- **Extension to regression and unsupervised learning.** The focus of this thesis are DNNs applied to classification problems. We believe that the current results may be extended to regression problems. However, even more interesting would be extension of this theory to generative models. For example, DNNs are used as generative models in Generative Adversarial Networks (GANs) and are trained in unsupervised fashion [112]. Clearly, there is a question of how many training samples are needed for a generative DNN to generalize well. Moreover, in Chapter 4 we have explored the role of data symmetries in the generalization of DNNs. We believe that the same principles should hold in generative DNNs. In particular, if some of the possible symmetries are encoded in a generative DNN it should be more straight forward to generate samples.

- **Better data models and covering numbers.** In Chapter 3 we have shown that GE of a DNN scales with the covering number of the data, provided that the DNN achieves a positive classification margin. We have shown examples of sparse models and manifold models where the covering number grow exponentially with the intrinsic dimension of the data and not with the ambient dimension. Although these models are quite general there is a possibility to explore other models and their covering numbers that would allow us to provide tighter GE bounds for particular classes of signals. For example, cartoon functions [113] may be used as a model for natural images and have been used to study the properties of the discrete Scattering transform [48]. Similarly, the multi-layer convolutional sparse coding model [114], which have been shown to closely model the behaviour of Convolutional Neural Networks (CNNs), may provide a better model for the data.

- **Extension to learned invariance.** There are many symmetries that are present in real world data, however, they can not be modelled explicitly. For example, a good face recognition DNN will be invariant to facial expressions. It seems to be the case in practice that DNNs learn to be at least approximately invariant (or less sensitive) to such transformations [115]. The current results in Chapter 4 are not able to capture the role of such invariances and it would be interesting to see if we can obtain similar bounds even for the case when the data symmetries are not predefined but are learned.

- **Relationship between the DNN optimization, the JM and the GE.** We have shown in Chapter 3 that a bounded JM of a DNN ensures that separation of training samples at the network output is transferred to its input. We also know from the works that study optimization of DNNs that a well behaved JM (singular values close to 1) are beneficial for optimization of DNNs. Therefore, the possible further direction is to explore if the methods that improve the optimization properties of DNNs also implicitly lead to a better classification margin and, therefore, to a smaller GE.

- **Transfer learning, multi-modal learning and the GE.** It has been observed in practice that a DNN trained on a large image classification training set can also

be used on other related, but different tasks. Usually, all the layers of the DNN except the last one are fixed, and the last one is adopted to the new task [116]. This is especially relevant when there is only a small training set available for the task at hand, and a pre-trained DNN significantly improves performance. This is an example of transfer learning.

Intuition behind the results in this thesis says that what is achieved by pre-training of a DNN is a reduction of the data complexity perceived by the new task, which leads to a much better generalization. A possible future work could, therefore, try to provide a principled theory of generalization for transfer learning by considering the similarity of data structure between the datasets.

Similar question may be asked in the case of multi-modal learning where we may observe two modalities of the same phenomena. Intuitively, if the modalities share some common structure, learning a DNN with one modality should reduce the sample complexity of learning the second modality.

Appendix A

Proofs of Chapter 3

A.1 Proof of the Jacobian Property (Theorem 3.2)

We first note that the line between $\mathbf{x}$ and $\mathbf{x}'$ is given by $\mathbf{x}+u(\mathbf{x}'-\mathbf{x})$, $u \in [0,1]$. We define the function $F(t) = f(\mathbf{x}+u(\mathbf{x}'-\mathbf{x}))$, and observe that $\frac{dF(u)}{du} = \mathbf{J}(\mathbf{x}+u(\mathbf{x}'-\mathbf{x}))(\mathbf{x}'-\mathbf{x})$. By the generalized fundamental theorem of calculus or the Lebesgue differentiation theorem we write

$$\begin{aligned} f(\mathbf{x}') - f(\mathbf{x}) &= F(1) - F(0) \\ &= \int_0^1 \frac{dF(u)}{du}\, du \\ &= \int_0^1 \mathbf{J}(\mathbf{x}+u(\mathbf{x}'-\mathbf{x}))\, du\, (\mathbf{x}'-\mathbf{x}) \,. \end{aligned} \tag{A.1}$$

This concludes the proof.

A.2 Proof of the Distance Expansion Bound (Corollary 3.3)

First note that $\|\mathbf{J}_{\mathbf{x},\mathbf{x}'}(\mathbf{x}'-\mathbf{x})\|_2 \leq \|\mathbf{J}_{\mathbf{x},\mathbf{x}'}\|_2 \|\mathbf{x}'-\mathbf{x}\|_2$ and that $\mathbf{J}_{\mathbf{x},\mathbf{x}'}$ is an integral of $\mathbf{J}(\mathbf{x}+u(\mathbf{x}'-\mathbf{x}))$. In addition, notice that we may always apply the following upper bound:

$$\|\mathbf{J}_{\mathbf{x},\mathbf{x}'}\|_2 \leq \sup_{\mathbf{x},\mathbf{x}' \in \mathcal{X}, u \in [0,1]} \|\mathbf{J}(\mathbf{x}+u(\mathbf{x}'-\mathbf{x}))\|_2 \,. \tag{A.2}$$

Since $\mathbf{x}+u(\mathbf{x}'-\mathbf{x}) \in \mathrm{conv}(\mathcal{X})\ \forall u \in [0,1]$, we get (3.20).

A.3 Proof of the Jacobian Matrix Spectral Norm Bounds (Lemma 3.1)

In all proofs we leverage the fact that for any two matrices $\mathbf{A}$, $\mathbf{B}$ of appropriate dimensions it holds $\|\mathbf{AB}\|_2 \leq \|\mathbf{A}\|_2\|\mathbf{B}\|_2$, where $\|\mathbf{A}\|_2$ is the spectral norm of $\mathbf{A}$ [105]. We also leverage the bound $\|\mathbf{A}\|_2 \leq \|\mathbf{A}\|_F$, where $\|\mathbf{A}\|_F$ is the Frobenious norm of $\mathbf{A}$ [105].

We start with the proof of statement 1). For the non-linear layer (3.29), we note that the Jacobian Matrix (JM) is a product of a diagonal matrix (3.30) and the weight matrix $\mathbf{W}_l$. Note that for all the considered non-linearities the diagonal elements of (3.30) are bounded by 1 (see derivatives in Table 3.1), which implies that the spectral norm of this matrix is bounded by 1. Therefore, the spectral norm of the JM is upper bounded by $\|\mathbf{W}_l\|_2$. The proof for the linear layer is trivial. In the case of the softmax layer (3.22) we have to show that the spectral norm of the softmax function $\left(-\zeta(\hat{\mathbf{z}})\zeta(\hat{\mathbf{z}})^T + \text{diag}(\zeta(\hat{\mathbf{z}})\right)$ is bounded by 1. We use the Gershgorin disc theorem, which states that the eigenvalues of $\left(-\zeta(\hat{\mathbf{z}})\zeta(\hat{\mathbf{z}})^T + \text{diag}(\zeta(\hat{\mathbf{z}})\right)$ are bounded by

$$\max_i (\zeta(\hat{\mathbf{z}}))_i (1 - (\zeta(\hat{\mathbf{z}}))_i) + (\zeta(\hat{\mathbf{z}}))_i \sum_{j \neq i} (\zeta(\hat{\mathbf{z}}))_j \,, \tag{A.3}$$

where $\zeta(\cdot)$ is the softmax function and $(\zeta(\hat{\mathbf{z}}))_i$ is the i-th element of $\zeta(\hat{\mathbf{z}})$.

Noticing that $\sum_{j \neq i} (\zeta(\hat{\mathbf{z}}))_j \leq 1$ leads to the upper bound

$$\max_i (\zeta(\hat{\mathbf{z}}))_i (2 - (\zeta(\hat{\mathbf{z}}))_i) \,. \tag{A.4}$$

Since $(\zeta(\hat{\mathbf{z}}))_i \in [0, 1]$ it is trivial to show that (A.4) is upper bounded by 1.

The proof of statement 2) is straightforward. Because the pooling regions are non-overlapping it is straightforward to verify that the rows of all the defined pooling operators $\mathbf{P}^l(\mathbf{z}^{l-1})$ are orthonormal. Therefore, the spectral norm of the JM is equal to 1.

A.4 Proof of the Classification Margin Bounds (Theorem 3.3)

Throughout the proof we will use the notation $o(s_i) = o(\mathbf{x}_i, y_i)$ and $\mathbf{v}_{ij} = \sqrt{2}(\boldsymbol{\delta}_i - \boldsymbol{\delta}_j)$. We start by proving the inequality in (3.35). Assume that the classification margin $\gamma^d(s_i)$ of training sample $(\mathbf{x}_i, y_i)$ is given and take $j^\star = \arg\min_{j \neq y_i} \mathbf{v}_{y_i j}^T f(\mathbf{x}_i)$. We then take a point $\mathbf{x}^\star$ that lies on the decision boundary between y_i and $j^\star$ such that $o(\mathbf{x}^\star, y_i) = 0$. Then

$$\begin{aligned} o(\mathbf{x}_i, y_i) &= o(\mathbf{x}, y_i) - o(\mathbf{x}^\star, y_i) \\ &= \mathbf{v}_{y_i j^\star}^T (f(\mathbf{x}_i) - f(\mathbf{x}^\star)) \\ &= \mathbf{v}_{y_i j^\star}^T \mathbf{J}_{\mathbf{x}_i, \mathbf{x}^\star} (\mathbf{x}_i - \mathbf{x}^\star) \leq \|\mathbf{J}_{\mathbf{x}_i, \mathbf{x}^\star}\|_2 \|\mathbf{x}_i - \mathbf{x}^\star\|_2. \end{aligned} \tag{A.5}$$

Note that by the choice of $\mathbf{x}^\star$, $\|\mathbf{x}_i - \mathbf{x}^\star\|_2 = \gamma^d(s_i)$ and similarly $\|\mathbf{J}_{\mathbf{x}_i, \mathbf{x}^\star}\|_2 \leq \sup_{\mathbf{x}: \|\mathbf{x} - \mathbf{x}_i\|_2 \leq \gamma^d(s_i)} \|\mathbf{J}(\mathbf{x})\|_2$. Therefore, we can write

$$o(s_i) \leq \sup_{\mathbf{x}: \|\mathbf{x} - \mathbf{x}_i\|_2 \leq \gamma^d(s_i)} \|\mathbf{J}(\mathbf{x})\|_2 \, \gamma^d(s_i), \tag{A.6}$$

which leads to (3.35). Next, we prove (3.36). Recall the definition of the classification margin in (3.9):

$$\begin{aligned} \gamma^d(s_i) &= \sup\{a : \|\mathbf{x}_i - \mathbf{x}\|_2 \leq a \implies g(\mathbf{x}) = y_i \, \forall \mathbf{x}\} \\ &= \sup\{a : \|\mathbf{x}_i - \mathbf{x}\|_2 \leq a \implies o(\mathbf{x}, y_i) > 0 \, \forall \mathbf{x}\}, \end{aligned}$$

where we leverage the definition in (3.34). We observe $o(\mathbf{x}, y_i) > 0 \iff \min_{j \neq y_i} \mathbf{v}_{y_i j}^T f(\mathbf{x}) > 0$ and

$$\min_{j \neq y_i} \mathbf{v}_{y_i j}^T f(\mathbf{x}) = \min_{j \neq y_i} \left(\mathbf{v}_{y_i j}^T f(\mathbf{x}_i) + \mathbf{v}_{y_i j}^T (f(\mathbf{x}) - f(\mathbf{x}_i)) \right).$$

Note that

$$\begin{aligned} \min_{j \neq y_i} \left(\mathbf{v}_{y_i j}^T f(\mathbf{x}_i) + \mathbf{v}_{y_i j}^T (f(\mathbf{x}) - f(\mathbf{x}_i)) \right) &\geq \min_{j \neq y_i} \mathbf{v}_{y_i j}^T f(\mathbf{x}_i) + \min_{j \neq y_i} \mathbf{v}_{y_i j}^T (f(\mathbf{x}) - f(\mathbf{x}_i)) \\ &= o(\mathbf{x}_i, y_i) + \min_{j \neq y_i} \mathbf{v}_{y_i j}^T (f(\mathbf{x}) - f(\mathbf{x}_i)). \end{aligned} \tag{A.7}$$

Therefore,

$$o(\mathbf{x}_i, y_i) + \min_{j \neq y_i} \mathbf{v}_{y_i j}^T (f(\mathbf{x}) - f(\mathbf{x}_i)) > 0 \implies o(\mathbf{x}, y_i) > 0 .$$

This leads to the bound of the classification margin

$$\gamma^d(s_i) \geq \sup\{a : \|\mathbf{x}_i - \mathbf{x}\|_2 \leq a \implies o(\mathbf{x}_i, y_i) + \min_{j \neq y_i} \mathbf{v}_{y_i j}^T (f(\mathbf{x}) - f(\mathbf{x}_i)) > 0 \, \forall \mathbf{x}\} .$$

Note now that

$$o(\mathbf{x}_i, y_i) + \min_{j \neq y_i} \mathbf{v}_{y_i j}^T (f(\mathbf{x}) - f(\mathbf{x}_i)) > 0 \tag{A.8}$$

$$\iff$$

$$o(\mathbf{x}_i, y_i) - \max_{j \neq y_i} \mathbf{v}_{y_i j}^T (f(\mathbf{x}_i) - f(\mathbf{x})) > 0 \tag{A.9}$$

$$\iff$$

$$o(\mathbf{x}_i, y_i) > \max_{j \neq y_i} \mathbf{v}_{y_i j}^T (f(\mathbf{x}_i) - f(\mathbf{x})) . \tag{A.10}$$

Moreover,

$$\max_{j \neq y_i} \mathbf{v}_{y_i j}^T (f(\mathbf{x}_i) - f(\mathbf{x})) \leq \sup_{\mathbf{x} \in \mathrm{conv}(\mathcal{X})} \|\mathbf{J}(\mathbf{x})\|_2 \|\mathbf{x}_i - \mathbf{x}\|_2 ,$$

where we have leveraged the fact that $\|\mathbf{v}_{ij}\|_2 = 1$ and the inequality (3.20) in Corollary 3.3. We may write

$$\gamma^d(s_i) \geq \sup\{a : \|\mathbf{x}_i - \mathbf{x}\|_2 \leq a \implies o(\mathbf{x}_i, y_i) > \sup_{\mathbf{x} \in \mathrm{conv}(\mathcal{X})} \|\mathbf{J}(\mathbf{x})\|_2 \|\mathbf{x}_i - \mathbf{x}\|_2 \, \forall \mathbf{x}\} .$$

a that attains the supremum can be obtain easily and we get:

$$\gamma^d(s_i) \geq \frac{o(\mathbf{x}_i, y_i)}{\sup_{\mathbf{x} \in \mathrm{conv}(\mathcal{X})} \|\mathbf{J}(\mathbf{x})\|_2} , \tag{A.11}$$

which proves (3.36). The bounds in (3.37) and (3.38) follow from the bounds provided in Lemma 3.1 and the fact that the spectral norm of a matrix product is upper bounded

by the product of the spectral norms. This concludes the proof.

A.5 Proof of the Batch Normalization Property (Theorem 3.4)

We denote by $\mathbf{W}_l^N$ the row normalized matrix obtained from $\mathbf{W}_l$ (in the same way as (3.63)). By noting that the ReLU and diagonal non-negative matrices commute, it is straight forward to verify that

$$[\mathbf{N}\left(\{\mathbf{z}_i^l\}_{i=1}^m, \mathbf{W}_l\right)\mathbf{W}_l\mathbf{z}^l]_\sigma = \mathbf{N}\left(\{\mathbf{z}_i^l\}_{i=1}^m, \mathbf{W}_l^N\right)[\mathbf{W}_l^N\mathbf{z}^l]_\sigma\,.$$

Note now that we can consider $\mathbf{N}(\{\mathbf{z}_i^l\}_{i=1}^m, \mathbf{W}_l^N)$ as the part of the weight matrix $\mathbf{W}_{l+1}$. Therefore, we can conclude that layer l has row normalized weight matrix. When the batch normalization is applied to layers, all the weight matrices will be row normalized. The exception is the weight matrix of the last layer, which will be of the form $\mathbf{N}(\{\mathbf{z}_i^{L-1}\}_{i=1}^m, \mathbf{W}_L)\mathbf{W}_L$.

A.6 Proof of the Manifold Distance Expansion Property (Theorem 3.5)

We begin by noting that

$$\begin{aligned} f(\mathbf{x}') - f(\mathbf{x}) &= f(c(1)) - f(c(0)) \\ &= \int_0^1 \frac{df(c(u))}{du}\,du \\ &= \int_0^1 \frac{df(c(u))}{dc(u)}\frac{dc(u)}{du}\,du\,, \end{aligned} \tag{A.12}$$

where the first equality follows from the generalized fundamental theorem of calculus, following the idea presented in the proof of Theorem 3.2. The second equality follows from the chain rule of differentiation. Finally, we note that $\frac{df(c(u))}{c(u)} = \mathbf{J}(c(u))$ and that

the norm of the integral is always smaller or equal to the integral of the norm and obtain

$$\begin{aligned}
\|f(\mathbf{x}') - f(\mathbf{x})\|_2 &= \left\| \int_0^1 \mathbf{J}(c(u)) \frac{dc(u)}{du}\, du \right\|_2 \\
&\leq \int_0^1 \|\mathbf{J}(c(u))\|_2 \left\| \frac{dc(u)}{du} \right\|_2 du \\
&\leq \sup_{u\in[0,1]} \|\mathbf{J}(c(u))\|_2 \int_0^1 \left\| \frac{dc(u)}{du} \right\|_2 du \\
&= \sup_{u\in[0,1]} \|\mathbf{J}(c(u))\|_2 d_G(\mathbf{x}, \mathbf{x}'), \qquad \text{(A.13)}
\end{aligned}$$

where we have noted that $\int_0^1 \left\| \frac{dc(u)}{du} \right\|_2 = d_G(\mathbf{x}, \mathbf{x}')$.

Appendix B

Proofs of Chapter 4

B.1 Proof of Generalization Error (GE) of an Invariant Deep Neural Network (DNN) (Theorem 4.1)

We show that under the assumptions of this theorem the DNN is $(\mathcal{N}(\mathcal{X}_0; d, \omega/2), 0)$-robust (see Definition 2.4). The GE bound then follows from theorems 3.1 and 3.3.

We construct a covering as follows. Take the covering that leads to the covering number $\mathcal{N}(\mathcal{X}_0; d, \omega/2)$ and denote the subsets of $\mathcal{X}_0$ by $\mathcal{K}_i$, $i = 1, \ldots, \mathcal{N}(\mathcal{X}_0; d, \omega/2)$. By the definition of $\mathcal{X}$ in (4.7) we can cover $\mathcal{X}$ by $\mathcal{N}(\mathcal{X}_0; d, \omega/2)$ sets of the form $\mathcal{T} \times \mathcal{K}_i$, $i = 1, \ldots, \mathcal{N}(\mathcal{X}_0; d, \omega/2)$.

Now take $\mathbf{x}_i$ in the training set and $\mathbf{x} \in \mathcal{X}$ such that $\mathbf{x}_i, \mathbf{x} \in \mathcal{T} \times \mathcal{K}_j$. Due to the invariance of f we have $\|f(\mathbf{x}_i) - f(\mathbf{x})\|_2 < \gamma$ and all $\mathbf{x}$ will lie in the same decision region as $\mathbf{x}_i$. This implies that stable and invariant learning algorithm is $(\mathcal{N}(\mathcal{X}_0; d, \omega/2), 0)$-robust. The GE bound follows from Theorem 3.1.

B.2 Proof of Covering number Ratio Bounds (Theorem 4.2)

Consider any covering of $\mathcal{X}_0$ that leads to the covering number $\mathcal{N}(\mathcal{X}_0; d, \epsilon)$. Denote the metric balls of radius ϵ that cover $\mathcal{X}_0$ by $\mathcal{C}_i$, $i = 1, \ldots, \mathcal{N}(\mathcal{X}_0; d, \epsilon)$. Denote the elements of $\mathcal{T}$ as t_j, $j = 1, \ldots, T$ and the transformed sets by $t_j(\mathcal{X}_0) = \{t_j(\mathbf{x}) : \mathbf{x} \in \mathcal{X}_0\}$, $j = 1, \ldots, T$.

First, we show that (4.18) implies that any possible metric ball of radius ϵ can only have non-empty intersection with one of the "copies" of $\mathcal{X}_0$. Denote by $\mathcal{B}$ an arbitrary

metric ball of radius ϵ. Then

$$\mathcal{B} \cap t_j(\mathcal{X}_0) \neq \emptyset \implies \mathcal{B} \cap t_k(\mathcal{X}_0) = \emptyset \quad \forall k \neq j. \tag{B.1}$$

To see this, observe that the definition of $\mathcal{B}$ implies that $d(\mathbf{x}, \mathbf{x}') \leq 2\epsilon$, $\forall \mathbf{x}, \mathbf{x}' \in \mathcal{B}$. Now take a point $\mathbf{x} \in \mathcal{B} \cap t_j(\mathcal{X}_0)$ and a point $\mathbf{x}' \in t_k(\mathcal{X}_0)$, $k \neq j$. Note that by (4.18) $d(\mathbf{x}, \mathbf{x}') > 2\epsilon$, which implies that $\mathbf{x}' \notin \mathcal{B}$ and, therefore, $\mathcal{B} \cap t_k(\mathcal{X}_0) = \emptyset$. This implies that the covering number of $\mathcal{X}$ with metric ball of radius ϵ is

$$\mathcal{N}(\mathcal{X}; d, \epsilon) = \sum_{j=1}^{T} \mathcal{N}(t_j(\mathcal{X}_0); d, \epsilon)\,. \tag{B.2}$$

Finally, it remains to be proven that $\mathcal{N}(t_j(\mathcal{X}_0); d, \epsilon) \geq \mathcal{N}(\mathcal{X}_0; d, \epsilon)$ $\forall t_j \in \mathcal{T}$, which is straightforward to establish given the condition (4.19). This proves (4.20). Proof of (4.22) is trivial as $\mathcal{X}_0 = \mathcal{X}$ when (4.21) holds.

References

[1] Michael I Jordan and Tom M Mitchell. Machine learning: Trends, perspectives, and prospects. *Science*, 349(6245):255–260, 2015.

[2] Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*. Springer, New York, 2006.

[3] Yann LeCun, Yoshua Bengio, and Geoffrey E Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[4] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828, 2013.

[5] Yoshua Bengio. Learning deep architectures for AI. *Foundations and trends in Machine Learning*, 2(1):1–127, 2009.

[6] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. *International Conference on Learning Representations (ICLR)*, Toulon, 2017.

[7] Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: from theory to algorithms*. Cambridge University Press, New York, 2014.

[8] Jure Sokolić, Raja Giryes, Guillermo Sapiro, and Miguel R D Rodrigues. Robust large margin deep neural networks. *IEEE Transactions on Signal Processing*, 65(16):4265–4280, 2017.

[9] Jure Sokolić, Raja Giryes, Guillermo Sapiro, and Miguel R D Rodrigues. Generalization error of invariant classifiers. *Conference on Artificial Intelligence and Statistics (AISTATS)*, 1094–1103, 2017.

[10] Jure Sokolić, Raja Giryes, Guillermo Sapiro, and Miguel R D Rodrigues. Generalization error of deep neural networks: Role of classification margin and data structure. *International Conference on Sampling Theory and Applications (SampTA) (to appear)*, 2017.

[11] Jure Sokolić, Majid Zamani, Andreas Demosthenous, and Miguel R D Rodrigues. A feature design framework for hardware efficient neural spike sorting. *International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, 1516–1519, 2015.

[12] Jure Sokolić, Francesco Renna, Robert Calderbank, and Miguel R D Rodrigues. Mismatch in the classification of linear subspaces: Upper bound to the probability of error. *IEEE International Symposium on Information Theory (ISIT)*, 2201–2205, 2015.

[13] Jure Sokolić, Francesco Renna, Robert Calderbank, and Miguel R D Rodrigues. Mismatch in the classification of linear subspaces: Sufficient conditions for reliable classification. *IEEE Transactions on Signal Processing*, 64(12):3035–3050, 2016.

[14] Jure Sokolić, Qiang Qiu, Miguel R D Rodrigues, and Guillermo Sapiro. Learning to Identify while Failing to Discriminate. *International Workshop on Cross-domain Human Identification at International Conference on Computer Vision (ICCV) (to appear)*, 2017.

[15] Olivier Chapelle, Bernhard Schölkopf, and Alexander Zien. *Semi-supervised learning*. The MIT Press, Cambridge, 2006.

[16] Vladimir N Vapnik. An overview of statistical learning theory. *IEEE Transactions on Neural Networks*, 10(5):988–999, 1999.

[17] Peter L Bartlett and Shahar Mendelson. Rademacher and Gaussian complexities: Risk bounds and structural results. *Journal of Machine Learning Research (JMLR)*, 3:463–482, 2002.

[18] Vladimir N Vapnik and Alexey J Chervonenkis. The necessary and sufficient conditions for consistency of the method of empirical risk. *Pattern Recognition and Image Analysis*, 1(3):284–305, 1991.

[19] Vladimir N Vapnik and Alexey J Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its Applications*, 16(2):264–280, 1971.

[20] Corinna Cortes and Vladimir N Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–279, 1995.

[21] Thomas Hofmann, Bernhard Schölkopf, and Alexander J Smola. Kernel methods in machine learning. *The Annals of Statistics*, 36(3):1171–1220, 2008.

[22] Olivier Bousquet and André Elisseeff. Stability and generalization. *Journal of Machine Learning Research (JMLR)*, 2:499–526, 2002.

[23] Huan Xu and Shie Mannor. Robustness and generalization. *Machine Learning*, 86(3):391–423, 2012.

[24] Aad W van der Vaart and Jon A Wellner. *Weak Convergence and Empirical Processes*. Springer, New York, 2000.

[25] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–538, 1986.

[26] Frank Rosenblatt. Principles of neurodynamics: Perceptrons and the theory of brain mechanisms. Technical Report Report No. VG-II96-G-8, Cornell Aeronautical Laboratory, Inc., New York, 1962.

[27] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems (NIPS)*, 1097–1105, 2012.

[28] Geoffrey E Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, and Brian Kingsbury. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.

[29] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 770–778, 2016.

[30] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015.

[31] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted Boltzmann machines. *International Conference on Machine Learning (ICML)*, 807–814, 2010.

[32] Joan Bruna, Arthur Szlam, and Yann LeCun. Learning stable group invariant representations with convolutional networks. *International Conference on Learning Representations (ICLR)*, Scottsdale, 2013.

[33] Y-Lan Boureau, Jean Ponce, and Yann LeCun. A theoretical analysis of feature pooling in visual recognition. *International Conference on Machine Learning (ICML)*, 111–118, 2010.

[34] David H Hubel and Torsten N Wiesel. Receptive fields of single neurones in the cat's striate cortex. *The Journal of Physiology*, 148:574–591, 1959.

[35] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, 1989.

[36] Kunihiko Fukushima. Neocognitron: A hierarchical neural network capable of visual pattern recognition. *Neural networks*, 1(2):119–130, 1988.

[37] Maximilian Riesenhuber and Tomaso Poggio. Hierarchical models of object recognition in cortex. *Nature Neuroscience*, 2(11):1019–1025, 1999.

[38] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. *European Conference on Computer Vision (ECCV)*, 630–645, 2016.

[39] Taco S Cohen and Max Welling. Group equivariant convolutional networks. *International Conference on Machine Learning (ICML)*, 2990–2999, 2016.

[40] Sander Dieleman, Jeffrey De Fauw, and Koray Kavukcuoglu. Exploiting cyclic symmetry in convolutional neural networks. *International Conference on Machine Learning (ICML)*, 1889–1898, 2016.

[41] Yanzhao Zhou, Qixiang Ye, Qiang Qiu, and Jianbin Jiao. Oriented response networks. *arXiv:1701.01833*, 2017.

[42] Max Jaderberg, Karen Simonyan, Andrew Zisserman, and Koray Kavukcuoglu. Spatial transformer networks. *Advances in Neural Information Processing Systems (NIPS)*, 2017–2025, 2015.

[43] João F Henriques and Andrea Vedaldi. Warped convolutions: Efficient invariance to spatial transformations. *arXiv:1609.04382*, 2016.

[44] Jifeng Dai, Haozhi Qi, Yuwen Xiong, Yi Li, Guodong Zhang, Han Hu, and Yichen Wei. Deformable convolutional networks. *arXiv:1703.06211*, 2017.

[45] Stéphane Mallat. Group invariant scattering. *Communications on Pure and Applied Mathematics*, 65(10):1331–1398, 2012.

[46] Joan Bruna and Stéphane Mallat. Invariant scattering convolution networks. *IEEE Transactions on Pattern Analysis and Machine Intellignce*, 35(8):1872–1886, 2012.

[47] Thomas Wiatowski and Helmut Bölcskei. A mathematical theory of deep convolutional neural networks for feature extraction. *arXiv:1512.06293*, 2015.

[48] Thomas Wiatowski, Michael Tschannen, Aleksandar Stanić, Philipp Grohs, and Helmut Bölcskei. Discrete deep feature extraction: A theory and new architectures. *International Conference on Machine Learning (ICML)*, 2149–2158, 2016.

[49] Edouard Oyallon and Stéphane Mallat. Deep roto-translation scattering for object classification. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2865–2873, 2014.

[50] Andrej Karpathy, Justin Johnson, and Fei-Fei Li. Visualizing and understanding recurrent networks. *International Conference on Learning Representations (ICLR) -workshop track*, San Juan, 2016.

[51] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[52] Ilya Sutskever, James Martens, George Dahl, and Geoffrey E Hinton. On the importance of initialization and momentum in deep learning. *International Conference on Machine Learning (ICML)*, 1139–1147, 2013.

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

[54] Diederik P Kingma and Jimmy L Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations (ICLR)*, San Diego, 2015.

[55] Anna Choromanska, Mikael Henaff, Michael Mathieu, Gérard B Arous, and Yann LeCun. The loss surfaces of multilayer networks. *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 192–204, 2015.

[56] Benjamin D Haeffele and René Vidal. Global optimality in tensor factorization, deep learning, and beyond. *arXiv:1506.07540*, 2015.

[57] Andrew M Saxe, James L McClelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *International Conference on Learning Representations (ICLR)*, Banff, 2014.

[58] Yann Ollivier. Riemannian metrics for neural networks I: feedforward networks. *Information and Inference*, 4(2):108–153, 2015.

[59] Guillaume Desjardins, Karen Simonyan, Razvan Pascanu, and Koray Kavukcuoglu. Natural neural networks. *Advances in Neural Information Processing Systems (NIPS)*, 2071–2079, 2015.

[60] Behnam Neyshabur, Ruslan Salakhutdinov, and Nathan Srebro. Path-sgd: Path-normalized optimization in deep neural networks. *Advances in Neural Information Processing Systems (NIPS)*, 2422–2430, 2015.

[61] Tim Salimans and Diederik P Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. *Advances in Neural Information Processing Systems (NIPS)*, 901–909, 2016.

[62] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *International Conference on Machine Learning (ICML)*, 448–456, 2015.

[63] Anders Krogh and John A Hertz. A simple weight decay can improve generalization. *Advances in Neural Information Processing Systems (NIPS)*, 950–957, 1992.

[64] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Iyla Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research (JMLR)*, 15(1):1929–1958, 2014.

[65] Yarin Gal and Zoubin Ghahramani. Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. *International Conference on Machine Learning (ICML)*, 1050–1059, 2016.

[66] Yichuan Tang. Deep learning using linear support vector machines. *Workshop on Representational Learning, International Conference on Machine Learning (ICML)*, 2013.

[67] Shizhao Sun, Wei Chen, Liwei Wang, and Tie-Yan Liu. Large margin deep neural networks: Theory and algorithms. *arXiv:1506.05232*, 2015.

[68] Senjian An, Munawar Hayat, Salman H Khan, Mohammed Bennamoun, Farid Boussaid, and Ferdous Sohel. Contractive rectifier networks for nonlinear maximum margin classification. *IEEE Conference on Computer Vision (CVPR)*, 2515–2523, 2015.

[69] Salah Rifai, Pascal Vincent, Xavier Muller, Xavier Glorot, and Yoshua Bengio. Contractive auto-encoders: Explicit invariance during feature extraction. *International Conference on Machine Learning (ICML)*, 833–840, 2011.

[70] Jiaji Huang, Qiang Qiu, Guillermo Sapiro, and Robert Calderbank. Discriminative robust transformation learning. *Advances in Neural Information Processing Systems (NIPS)*, 1333–1341, 2015.

[71] Stephan Zheng, Yang Song, Thomas Leung, and Ian Goodfellow. Improving the robustness of deep neural networks via stability training. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 4480–4488, 2016.

[72] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control Signals and Systems*, 2(4):303–314, 1989.

[73] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251–257, 1991.

[74] Guido Montúfar, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio. On the number of linear regions of deep neural networks. *Advances in Neural Information Processing Systems (NIPS)*, 2924–2932, 2014.

[75] Nadav Cohen, Or Sharir, and Amnon Shashua. On the expressive power of deep learning: A tensor analysis. *Conference on Learning Theory (COLT)*, 698–728, 2016.

[76] Nadav Cohen, Or Sharir, Ronen Tamari, David Yakira, Yoav Levine, and Amnon Shashua. Analysis and design of convolutional networks via hierarchical tensor decompositions. *arXiv:1705.02302*, 2017.

[77] Matus Telgarsky. Benefits of depth in neural networks. *Conference on Learning Theory (COLT)*, 1517–1539, 2016.

[78] Stefano Soatto. Actionable information in vision. *International Conference on Computer Vision (ICCV)*, 2138–2145, 2009.

[79] Stefano Soatto and Alessandro Chiuso. Visual representations: Defining properties and deep approximations. *International Conference on Learning Representations (ICLR)*, San Juan, 2016.

[80] Tomaso Poggio, Jim Mutch, Joel Leibo, Lorenzo Rosasco, and Andrea Tacchetti. The computational magic of the ventral stream: Sketch of a theory (and why some deep architectures work). Technical report, Massachusetts Institute of Technology, Cambridge, 2012.

[81] Fabio Anselmi, Joel Z Leibo, Lorenzo Rosasco, Jim Mutch, Andrea Tacchetti, and Tomaso Poggio. Unsupervised learning of invariant representations with low sample complexity: The magic of sensory cortex or a new framework for machine learning? *arXiv:1311.4158v5*, 2014.

[82] Fabio Anselmi, Lorenzo Rosasco, and Tomaso Poggio. On invariance and selectivity in representation learning. *Information and Inference*, 5(2):134–158, 2016.

[83] Kihyuk Sohn and Honglak Lee. Learning invariant representations with local transformations. *International Conference on Machine Learning (ICML)*, 1311–1318, 2012.

[84] Patrice Y Simard, Dave Steinkraus, and John C Platt. Best practices for convolutional neural networks applied to visual document analysis. *International Conference on Document Analysis and Recognition (ICDAR)*, 958–962, 2003.

[85] Laurent Sifre and Stéphane Mallat. Rotation, scaling and deformation invariant scattering for texture discrimination. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 1233–1240, 2013.

[86] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. *European Conference on Computer Vision (ECCV)*, 818–833, 2014.

[87] Aravindh Mahendran and Andrea Vedaldi. Understanding deep image representations by inverting them. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 5188–5196, 2015.

[88] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. Image style transfer using convolutional neural networks. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2414–2423, 2016.

[89] Joan Bruna, Arthur Szlam, and Yann LeCun. Signal recovery from pooling representations. *International Conference on Machine Learning (ICML)*, 307–315, 2014.

[90] Raja Giryes, Guillermo Sapiro, and Alex M Bronstein. Deep neural networks with random Gaussian weights: A universal classification strategy? *IEEE Transactions on Signal Processing*, 64(13):3444–3457, 2016.

[91] Thomas Wiatowski, Philipp Grohs, and Helmut Bölcskei. Energy Propagation in Deep Convolutional Neural Networks. *arXiv:1704.03636*, 2017.

[92] Noga Alon, Shai Ben-David, Nicolo Cesa-Bianchi, and David Haussler. Scale-sensitive dimensions, uniform convergence, and learnability. *Journal of the ACM*, 44(4):615–631, 1997.

[93] Behnam Neyshabur, Ryota Tomioka, and Nathan Srebro. Norm-based capacity control in neural networks. *Conference on Learning Theory (COLT)*, 1376–1401, 2015.

[94] Yuchen Zhang, Jason D Lee, Martin J Wainwright, and Michale I Jordan. Learning halfspaces and neural networks with random initialization. *arXiv:1511.07948*, 2015.

[95] Nadav Cohen and Amnon Shashua. Inductive bias of deep convolutional networks through pooling geometry. *International Conference on Learning Representations (ICLR)*, Toulon, 2017.

[96] Yaser S Abu-Mostafa. Hints and the VC dimension. *Neural Computation*, 5(2):278–288, 1993.

[97] Ilya Soloveychik, Dmitry Trushin, and Ami Wiesel. Group symmetric robust covariance estimation. *IEEE Transactions on Signal Processing*, 64(1):244–257, 2016.

[98] Parikshit Shah and Venkat Chandrasekaran. Group symmetry and covariance regularization. *Electronic Journal of Statistics*, 6:1600–1640, 2012.

[99] Shahar Mendelson, Alain Pajor, and Nicole Tomczak-Jaegermann. Uniform uncertainty principle for Bernoulli and subgaussian ensembles. *Constructive Approximation*, 28(3):277–289, 2008.

[100] Nakul Verma. Distance preserving embeddings for general n-dimensional manifolds. *Journal of Machine Learning Research (JMLR)*, 14(1):2415–2448, 2013.

[101] Behnam Neyshabur, Ryota Tomioka, Ruslan Salakhutdinov, and Nathan Srebro. Data-dependent path normalization in neural networks. *International Conference on Learning Representations (ICLR)*, San Diego, 2015.

[102] Andreas Veit, Michael Wilber, and Serge Belongie. Residual networks are exponential ensembles of relatively shallow networks. *Advances in Neural Information Processing Systems (NIPS)*, 550–558, 2016.

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

[104] Alistair G Watson. Characterization of the subdifferential of some matrix norms. *Linear Algebra and its Applications*, 170:33–45, 1992.

[105] Kaare Brandt Petersen and Michael Syskind Pedersen. *The matrix cookbook*. Technical University of Denmark, Kongens Lyngby, 2012.

[106] Alex Krizhevsky and Geoffrey E Hinton. Learning multiple layers of features from tiny images. Technical report, University of Toronto, Toronto, 2009.

[107] Yuan-Sheng Hsiao, Jordi Sanchez-Riera, Tekoing Lim, Kai-Lung Hua, and Wen-Huang Cheng. LaRED: A large RGB-D extensible hand gesture dataset. *ACM Multimedia Systems Conference*, 53–58, 2014.

[108] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C Berg, and Li Fei-Fei. ImageNet large scale visual recognition challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[109] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. *International Conference on Learning Representations (ICLR) - workshop track*, San Diego, 2015.

[110] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *British Machine Vision Conference*, York, 2016.

[111] Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv:1605.02688*, 2016.

[112] Ian J Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in Neural Information Processing Systems (NIPS)*, 2672–2680, 2014.

[113] David L Donoho. Sparse components of images and optimal atomic decompositions. *Constructive Approximation*, 17(3):353–382, 2001.

[114] Vardan Papyan, Yaniv Romano, and Michael Elad. Convolutional neural networks analyzed via convolutional sparse coding. *arXiv:1607.08194*, 2016.

[115] Stéphane Mallat. Understanding deep convolutional networks. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 374(2065), 2016.

[116] Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. CNN features off-the-shelf: An astounding baseline for recognition. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, 806–813, 2014.