이학석사 학위논문

Robustness and Upper Bound of Generalization Error in Deep Neural Networks

(깊은 신경망에서 강건성과 일반화 오차의 상한)

2018년 8월

서울대학교 대학원

수 리 과 학 부

이 효 제

Robustness and Upper Bound of Generalization Error in Deep Neural Networks

(깊은 신경망에서 강건성과 일반화 오차의 상한)

지도교수 강 명 주

이 논문을 이학석사 학위논문으로 제출함

2017년 10월

서울대학교 대학원

수 리 과 학 부

이 효 제

이 효 제의 이학석사 학위논문을 인준함

2017년 12월

위 원 장 _____ (인)

부 위 원 장 _____ (인)

위 원 _____ (인)

Robustness and Upper Bound of Generalization Error in Deep Neural Networks

by

Hyoje Lee

A DISSERTATION

Submitted to the faculty of the Graduate School
in partial fulfillment of the requirements
for the degree of Master of Science
in the Department of Mathematical Sciences
Seoul National University
August 2018

Abstract

Robustness and Upper Bound of Generalization Error in Deep Neural Networks

Hyoje Lee

Department of Mathematical Sciences
The Graduate School
Seoul National University

The generalization of Deep Neural Networks(DNNs) is a crucial issue. While DNNs perform well in many domains, it is hard to explain theoretically why DNNs generalize well. The robustness, introduced in Xu, is a good notion to explain the generalization of DNNs. In this work, we branch off the notion of the robustness and generalization error. In addition, we assume that the input space is a bounded d-dimensional subspace of $\mathbb{R}^n$ and derive a new upper bound of the generalization error in DNNs. We attempt to demonstrate our work by using the Jacobian regularizer, suggested by Sokolic, based on the Jacobian matrix in DNNs. Also we visualize the result by using t-SNE.

Key words: Robustness, Generalization Error, Jacobian regularization, Deep Learning
Student Number: 2016-20246

Contents

Chapter 1

Introduction

In recent years, Deep Neural Networks(DNNs) have been achieving the state-of-the-art performance in many domains. These architectures have dramatically improved in visual object recognition, object detection, segmentation, speech recognition and many other fields[5, 4, 16, 8, 18, 12].

In the early stages of DNNs, *overfitting*, the problem that the learning algorithm provides a good performance for training data but not test data, was a serious obstacle. To solve this problem, there were many efforts. Dropout in [20] is a good way to prevent the overfitting problem. The idea is to randomly drop out units(hidden and visible) in a given neural network. It is helpful to prevent the overfitting problem. Batch normalization, suggested in [9], is also a good way to address the overfitting problem. Before the activating function, normalize the output of each layer. The weight decay [11] is also a universal method to prevent the overfitting problem. These techniques are relevant to generalize DNNs.

The issue of generalization of DNNs has attracted increasing amount of research interest. The work in [2] shows that the generalization of DNNs does not depend on the number of weights (or hidden units) in each layer but depends on the size of the weights in each layer. There are several works that provide theoretical or mathematical reasons to explain why DNNs generalize well. The *robustness*, introduced in Xu [22], is a good notion to explain the generalization of DNNs. The *robust* learning algorithm means that the algorithm provides a stable performance even after adding a noise to the training

dataset. Thus the generalization of DNNs is relevant to the robustness of DNNs.

The notion of robustness can be applied to various learning algorithms. Originally, Xu [22] derives the generalization bounds in various supervised learning, for example, SVM, LASSO, etc. There were also other attempts influenced by Xu [22] to apply the notion of robustness to other learning algorithms. Some work adapt the robustness to metric learning [3].

The work in Sokolic [19] suggests the notion of generalization error of DNNs via their classification margin. Their analysis shows that a bounded norm of *Jacobian matrix* is crucial for DNNs to generalize well. Moreover, Sokolic [19] suggests the *Jacobian regularizer* to lower the generalization error.

In this paper, we propose to study the notion of robustness of DNNs and show the related experiment results. We adapt the framework of Sokolic [19] on the other conditions. While Sokolic [19] assume that the input space is a C_M regular d-dimensional manifold, introduced in [21], we assume that the input space is a bounded d-dimensional subspace of $\mathbb{R}^n$. In this part, the crucial part is how to calculate the *covering number* [17] of input space. So we present here how to calculate the covering number of our input space. We already know heuristically that DNNs generalize well when we normalize the data before input. We try to explain why the normalization of input data is a good method to generalize DNNs by our generalization bound. Moreover, we show the related experiment results. We use the Jacobian regularizer and its *regularizer constant* λ. We use several constants and report the results correspondingly.

The rest of the paper is organized as follows. We introduce the notion of robustness and generalization error in **Chapter 2**, as well as suggesting our input space. We propose to explain the structure of DNNs and Jacobian regularizer in **Chapter 3**. The experiments and its results are in **Chapter 4**. We conclude in **Chapter 5**. Some explanations and proofs are in **Appendix**.

Chapter 2

Robustness in Deep Neural Networks

2.1 Robustness and Generalization Error

We will review the theory of robustness in this section. The notion is introduced in Xu [22]. First, we consider the input vector $\mathbf{x} \in \mathbb{X} \subseteq \mathbb{R}^n$ and the label $y \in \mathbb{Y}$, where $\mathbb{X}$ is the input space and $\mathbb{Y} = \{1, 2, 3, \ldots, N\}$ is the label space with N labels(classes). Then we can consider the sample space $\mathbb{S} = \mathbb{X} \times \mathbb{Y}$ and its element is $\mathbf{s} = (\mathbf{x}, y) \in \mathbb{S}$. Also we assume that samples are drawn from $\mathbb{S}$ with distribution P. In this paper, we use a notation $\mathbb{S}_m$ to denote a training sample set with m samples drawn from P by $\mathbb{S}_m = (\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_m)$. Note that $\mathbf{s}$ means an arbitrary sample in the sample space $\mathbb{S}$ and $\mathbf{s}_i$ means a training sample in the training sample set $\mathbb{S}_m$ in this paper.

Let $\mathcal{C} : \mathbb{X} \to \mathbb{Y}$ be a classifier. Our purpose is to obtain a good classifier which satisfies $\mathcal{C}(\mathbf{x})=y$ by learning algorithm. In this paper, our classifier is based on DNNs. To measure a quality of the classifier, we measure the loss of outputs by using the loss function $\mathfrak{L}(\mathcal{C}(\mathbf{x}), y)$, where $\mathfrak{L}$ is a loss function which measure a loss between the predicted label and the true label, $\mathbf{x}$ is an input vector and y is a corresponding label. Now we define the following losses and error:

Definition 2.1 (in Xu [22])**.** *Let $\mathbb{S}$ be the sample space and $\mathbb{S}_m$ a training sample set with m samples. Let $\mathbf{s}=(\mathbf{x}, y)$, $\mathbf{s}_i=(\mathbf{x}_i, y_i)$ be elements of $\mathbb{S}$ and*

$\mathbb{S}_m$, *respectively. For a classifier* $\mathcal{C}$, *the* ***empirical loss*** *of training set* $\mathbb{S}_m$, *denoted* $\mathfrak{L}_{emp}$, *is defined by*

$$\mathfrak{L}_{emp}(\mathcal{C}) = \frac{1}{m}\sum_{i=1}^{m}\mathfrak{L}(\mathcal{C}(\mathbf{x}_i), y_i), \tag{2.1}$$

the ***expected loss*** *(true loss) over distribution P, denoted* $\mathfrak{L}_{exp}$, *is defined by*

$$\mathfrak{L}_{exp}(\mathcal{C}) = \mathbb{E}_{\mathbf{s}\sim P}[\mathfrak{L}(\mathcal{C}(\mathbf{x}), y)], \tag{2.2}$$

and the ***generalization error***, *denoted* $\mathrm{GE}(\mathcal{C})$, *is defined by*

$$\mathrm{GE}(\mathcal{C}) = |\mathfrak{L}_{exp} - \mathfrak{L}_{emp}|. \tag{2.3}$$

Of course, we cannot measure the expected loss because we cannot know the distribution P of sample space $\mathbb{S}$. Our goal of learning is to predict the expected loss by using the empirical loss. It means that if we want to generalize the learning algorithms, then the generalization error need to be small. Hence we want to leverage the generalization error when algorithm is learning. Now we dwell on the bound of generalization error in Xu [22].

First, we define the *robustness* of a learning algorithm.

Definition 2.2 (in Xu [22])**.** *Let* $\mathbb{S}$ *be a sample space and* $\mathbb{S}_m$ *a training sample set with m samples. Let* $\mathbf{s}$=($\mathbf{x}$, y), $\mathbf{s}_i$=($\mathbf{x}_i$, y_i) *be elements of* $\mathbb{S}$ *and* $\mathbb{S}_m$, *respectively. A classifier* $\mathcal{C}$ *of a learning algorithm is* ***(K,*** $\epsilon(\mathbb{S}_m)$***)-robust*** *, for* $K \in \mathbb{N}, \epsilon : \mathbb{S}^m \to \mathbb{R}$, *if the sample space* $\mathbb{S}$ *can be partitioned K disjoint sets, denoted by* $\{C_j\}_{j=1}^{K}$, *such that for all* $\mathbf{s}_i \in \mathbb{S}_m$ *and all* $\mathbf{s} \in \mathbb{S}$,

$$\mathbf{s}_i, \mathbf{s} \in C_j \implies |\mathfrak{L}(\mathcal{C}(\mathbf{x}_i), y_i) - \mathfrak{L}(\mathcal{C}(\mathbf{x}), y)| < \epsilon(\mathbb{S}_m). \tag{2.4}$$

The above definition means that if a classifier is (K, $\epsilon(\mathbb{S}_m)$)-robust, then the difference of losses between the samples which are in the same set C_j is less than the $\epsilon(\mathbb{S}_m)$, i.e., two samples in same set C_j are close. In Xu [22], they suggest that the bound of the generalization error by using the notion of robustness.

Theorem 2.3 (*Theorem 3* in Xu [22]). *If a classifier $\mathcal{C}$ of learning algorithm is (K, $\epsilon(\mathbb{S}_m)$)-robust and $\mathfrak{L}(\mathcal{C}(\mathbf{x}), y) \leq M$ for all $\mathbf{s}=(\mathbf{x}, y) \in \mathbb{S}$, then for any $\delta > 0$, with probability at least $1 - \delta$,*

$$\mathrm{GE}(\mathcal{C}) \leq \epsilon(\mathbb{S}_m) + M\sqrt{\frac{2K \cdot \log 2 + 2\log(\frac{1}{\delta})}{m}}. \tag{2.5}$$

Note that our purpose of learning is to minimize the generalization error. So we need to leverage the upper bound of generalization error. Of course, the upper bound decrease as m $\to \infty$, where m is the number of training samples. However it is difficult in practical, so we leverage the other terms.

Now we define the *classification margin* as follows:

Definition 2.4. *The **classification margin** of a training sample $\mathbf{s}_i = (\mathbf{x}_i, y_i)$ is defined by*

$$\Gamma(\mathbf{s}_i) = \sup\{l : \| \mathbf{x}_i - \mathbf{x} \|_2 \leq l \implies f(\mathbf{x}) = y_i \quad \forall \mathbf{x}\}. \tag{2.6}$$

Above the classification margin of $\mathbf{s}_i$ means that if we consider the ball of radius Γ centered at $\mathbf{x}_i$, then all samples $\mathbf{x}$ in the ball will be classified the same label y_i, where y_i is the label corresponding to $\mathbf{x}_i$.

However, the bound on the number of partitions K depends on the *covering number* of the input space $\mathbb{X}$. So we define the covering number as follows :

Definition 2.5 (in [17]). *Let $\mathbb{X}$ be (a subspace of) $\mathbb{R}^n$ and $\gamma \in \mathbb{R}$. We call a set $C \subset \mathbb{X}$ is an γ-**cover** of $\mathbb{X}$ if for each $\mathbf{x} \in \mathbb{X}$, there exists $\hat{\mathbf{x}} \in C$ such that $\| \mathbf{x} - \hat{\mathbf{x}} \|_2 \leq \gamma$. The γ-**covering number** of $\mathbb{X}$, denoted $\mathcal{N}(\mathbb{X}; \gamma)$, is defined by*

$$\mathcal{N}(\mathbb{X}; \gamma) = \min\{ |C| : C \text{ is a } \gamma\text{-cover of } \mathbb{X}\}. \tag{2.7}$$

Note that the sample space $\mathbb{S}$ is the Cartesian product of the input space $\mathbb{X}$ and the label space $\mathbb{Y}$ with $|\mathbb{Y}| = N$. Since the label space $\mathbb{Y}$ is discrete, $\mathcal{N}(\mathbb{S}; \gamma) \leq N \cdot \mathcal{N}(\mathbb{X}; \gamma)$, where N is the number of labels. Thus we have the following Lemma 2.6.

Lemma 2.6 (*Theorem 2* in Sokolic [19]). *If there exist* $\gamma > 0$ *such that*

$$\Gamma(\mathbf{s}_i) > \gamma \quad \forall \mathbf{s}_i \in \mathbb{S}_m, \tag{2.8}$$

then the classifier $\mathcal{C}$ *is* $(N \cdot \mathcal{N}(\mathbb{X}; \frac{\gamma}{2}), 0)$*-robust, where* N *is the number of labels, i.e.,* $|\mathbb{Y}|$*=*N*.*

By Theorem 2.3 and Lemma 2.6, we can conclude that $\mathrm{GE}(\mathcal{C})$ with a classification margin Γ is bounded as follows.

Theorem 2.7 (in Sokolic [19]). *Let* $\mathcal{C}$ *be a classifier with a classification margin* Γ*. Assume that there exist* $\gamma > 0$ *such that* $\Gamma(\mathbf{s}_i) > \gamma$ *for all* $\mathbf{s}_i \in \mathbb{S}_m$*. Then for any* $\delta > 0$*, with probability at least* $1 - \delta$*,*

$$\mathrm{GE}(\mathcal{C}) \leq M \sqrt{\frac{2N \cdot \mathcal{N}(\mathbb{X}; \frac{\gamma}{2}) \cdot \log 2 + 2\log(\frac{1}{\delta})}{m}}. \tag{2.9}$$

There are various loss functions. One of them, the 1–0 indicator function is a loss function that if the predicted label is equal to the true label, then it corresponds to 0, otherwise it corresponds to 1. If we take a loss function to be the 1–0 indicator function, then $\mathfrak{L}(\mathcal{C}(\mathbf{x}), y) \leq 1$. Also if we neglect the $2\log(\frac{1}{\delta})$ term, then we have the inequality

$$\mathrm{GE}(\mathcal{C}) \lesssim \sqrt{\frac{2N \cdot \mathcal{N}(\mathbb{X}; \frac{\gamma}{2}) \cdot \log 2}{m}}. \tag{2.10}$$

2.2 Upper Bound of Generalization Error

In Sokolic [19], the assumption of their work is that the input space $\mathbb{X}$ is a C_M-regular d-dimensional manifold with $\mathcal{N}(\mathbb{X}; \gamma) \leq (\frac{C_M}{\gamma})^d$, suggested in [21]. However, we suggest that let $\mathbb{X}$ be a bounded d-dimensional subspace of $\mathbb{R}^n$. In here, we present our adaptation of the covering number of $\mathbb{X}$. The concept is similar to [17].

Lemma 2.8 (adapted from *Example 27.1* in [17]). *Suppose that* $\mathbb{X}$ *is a bounded* d*-dimensional subspace of* $\mathbb{R}^n$ *and there exists* $\mu \in \mathbb{R}$ *such that*

$\| \mathbf{x} \|_2 \leq \mu$ *for all* $\mathbf{x} \in \mathbb{X}$. *Then the* γ*-covering number of* $\mathbb{X}$ *is bounded by* $\left(\frac{\mu\sqrt{d}}{\gamma}\right)^d$, *i.e.,*

$$\mathcal{N}(\mathbb{X}; \gamma) \leq \left(\frac{\mu\sqrt{d}}{\gamma}\right)^d. \tag{2.11}$$

Proof. The detailed proof is in Appendix A.2. □

Now we get the upper bound of the covering number of $\mathbb{X}$. Thus we can get the upper bound of the generalization error. The follow Corollary adapted from in Sokolic [19].

Corollary 2.9 (adapted from *Corollary 1* in Sokolic [19])**.** *Let* $\mathcal{C}$ *be a classifier with a classification margin* Γ. *Suppose that* $\mathbb{X}$ *is a bounded* d*-dimensional subspace of* $\mathbb{R}^n$ *and there exists* $\mu \in \mathbb{R}$ *such that* $\| \mathbf{x} \|_2 \leq \mu$ *for all* $\mathbf{x} \in \mathbb{X}$. *If there exist* $\gamma > 0$ *such that* $\Gamma(\mathbf{s}_i) > \gamma$ *for all* $\mathbf{s}_i \in \mathbb{S}_m$, *then for any* $\delta > 0$, *with probability at least* $1 - \delta$,

$$\mathrm{GE}(\mathcal{C}) \leq \sqrt{\frac{N \cdot 2^{d+1} \cdot \left(\mu\sqrt{d}\right)^d \cdot \log 2 + 2\log\left(\frac{1}{\delta}\right)}{\gamma^d \, m}}. \tag{2.12}$$

Proof. The Corollary is followed by Theorem 2.7 and Lemma 2.8. □

If we neglect the $2\log(\frac{1}{\delta})$ term of the inequality (2.12), then we have

$$\mathrm{GE}(\mathcal{C}) \lesssim \sqrt{\frac{N \cdot 2^{d+1} \cdot \left(\mu\sqrt{d}\right)^d \cdot \log\ 2}{\gamma^d \, m}}. \tag{2.13}$$

Since our purpose is to lower the generalization error, we need to leverage γ, so that the upper bound of $\mathrm{GE}(\mathcal{C})$ decreases in (2.12). By Corollary 2.9, we can get several results.

First, the normalization before input the data could be helpful to generalization of DNNs. If we normalize the data before input, then the norm of data is bounded. So μ of the input space $\mathbb{X}$ could be small. Thus we have less upper bound of generalization error.

Second, if we could leverage γ, then we would get less upper bound of the generalization error. Thus now, we dwell the relation between the Jacobian matrix and the classification margin in the next chapter.

Chapter 3

Deep Neural Networks and Jacobian regularizer

3.1 Jacobian Matrix in Deep Neural Networks

Let f be a Deep Neural Network(DNN) in a classification problem. In a classification problem, $\mathcal{C}(\mathbf{x}) = \arg\max_i(f(\mathbf{x})_i)$, where $\mathbf{x}$ is a input vector and $f(\mathbf{x})_i$ is a i-th component of $f(\mathbf{x})$. In here, f can be considered as a function $f : \mathbb{R}^n \to \mathbb{R}^N$, where n is the dimension of input vector and N is the number of labels. So we can consider the *Jacobian matrix* of f. Since DNN consists of several or many layers.

In this section, we state the Jacobian matrix of each layer which is described in Sokolic [19]. In here, the output of l-th layer is denoted by $\mathbf{u}^l$ where $l = 1, 2, \ldots, L$; input layer and last layer are denoted by $\mathbf{u}^0 = \mathbf{x}$ and $\mathbf{u} = f(\mathbf{x})$.

Linear and Softmax Layers

In DNN, there are linear layers, especially the last layer. It can be described as follows:

$$\mathbf{u}^L = \hat{\mathbf{u}}, \quad \hat{\mathbf{u}} = \mathbf{W}^L\mathbf{u}^{L-1} + \mathbf{b}^L, \tag{3.1}$$

where $\mathbf{W}_L$ is a weight matrix, $\mathbf{b}_L$ is a bias vector in last layer and $\mathbf{u}^L$ is a output of a L-th layer. We note that Jacobian matrix of a linear layer is :

$$\frac{d\mathbf{u}^L}{d\mathbf{u}^{L-1}} = \mathbf{W}^L. \tag{3.2}$$

In other words, Jacobian matrix of a linear layer is equal to its weight matrix. For last layer, we usually use the $softmax$ layer. It is described as follows:

$$\mathbf{u}^L = [\hat{\mathbf{u}}]_{softmax}, \quad \hat{\mathbf{u}} = \mathbf{W}^L\mathbf{u}^{L-1} + \mathbf{b}^L, \tag{3.3}$$

where $[\cdot]_{softmax} : \mathbb{R}^n \to \mathbb{R}^n$ is the *softmax function* such that its i-th component $([\mathbf{u}]_{softmax})_i$ is :

$$([\mathbf{u}]_{softmax})_i = \frac{e^{\mathbf{u}_i}}{\sum_{j=1}^{n} e^{\mathbf{u}_j}}, \quad \mathbf{u}_i : i\text{-th component of } \mathbf{u}. \tag{3.4}$$

Then its Jacobian matrix is :

$$\frac{d\mathbf{u}^L}{d\mathbf{u}^{L-1}} = \frac{d\mathbf{u}^L}{d\hat{\mathbf{u}}} \cdot \frac{d\hat{\mathbf{u}}}{d\mathbf{u}^{L-1}} \tag{3.5}$$

$$= (-[\hat{\mathbf{u}}]_{softmax} \cdot [\hat{\mathbf{u}}]^t_{softmax} + \mathrm{diag}([\hat{\mathbf{u}}]_{softmax})) \cdot \mathbf{W}^L. \tag{3.6}$$

We defer the detailed explanation for equation (3.6) to Appendix A.1.

Non-Linear Layers

In non-linear layers of DNN, we use a non-linear function σ, for example, Rectified Linear Unit(ReLU), Sigmoid, Hyperbolic tangent, etc. These non-linear functions are well explained in [7]. A l-th non-linear layer can be described as follows:

$$\mathbf{u}^l = [\hat{\mathbf{u}}]_\sigma, \quad \hat{\mathbf{u}} = \mathbf{W}^l\mathbf{u}^{l-1} + \mathbf{b}^l, \tag{3.7}$$

where $[\cdot]_\sigma : \mathbb{R}^n \to \mathbb{R}^n$ is a function such that its i-th component $([\mathbf{u}]_\sigma)_i$ is :

$$([\hat{\mathbf{u}}]_\sigma)_i = \sigma(\hat{\mathbf{u}}_i), \quad \hat{\mathbf{u}}_i : i\text{-th component of } \hat{\mathbf{u}}. \tag{3.8}$$

It is easy to know that

$$
\begin{aligned}
(\frac{d\mathbf{u}}{d\hat{\mathbf{u}}})_{ii} &= \frac{d\sigma(\hat{\mathbf{u}}_i)}{d\hat{\mathbf{u}}_i} = \sigma'(\hat{\mathbf{u}}_i) \\
(\frac{d\mathbf{u}}{d\hat{\mathbf{u}}})_{ij} &= 0 \qquad\qquad\qquad\qquad \text{if } i \neq j
\end{aligned}
$$

So the Jacobian matrix of a non-linear layer is :

$$
\begin{aligned}
\frac{d\mathbf{u}^l}{d\mathbf{u}^{l-1}} &= \frac{d\mathbf{u}^l}{d\hat{\mathbf{u}}} \cdot \frac{d\hat{\mathbf{u}}}{d\mathbf{u}^{l-1}} && (3.9) \\
&= \text{diag}(\{\sigma'(\hat{\mathbf{u}}_i^l) : i = 1, 2, \cdots\}) \cdot \mathbf{W}^L. && (3.10)
\end{aligned}
$$

Pooling Layer

A pooling layer is described as follows :

$$\mathbf{u}^l = \mathbf{P}^l(\mathbf{u}^{l-1})z^{l-1}, \tag{3.11}$$

where $\mathbf{P}^l(\mathbf{u}^{l-1})$ is the pooling matrix(average pooling, max pooling, etc). Since the pooling matrix is linear, so its Jacobian matrix is equal to itself :

$$\frac{d\mathbf{u}^l}{d\mathbf{u}^{l-1}} = \mathbf{P}^l(\mathbf{u}^{l-1}). \tag{3.12}$$

3.2 Jacobian Regularizer

In this section, we state the Jacobian regularizer which is suggested in Sokolic [19]. Before define the Jacobian regularizer, we need the notion of *score*.

Definition 3.1 (in Sokolic [19]). *The **score** of a training sample* $\mathbf{s}_i = (\mathbf{x}_i, y_i)$

$$o(\mathbf{s}_i) = \min_{j \neq y_i} \sqrt{2}[(f(x_i))_{y_i} - (f(x_i))_j]. \tag{3.13}$$

By the definition of the score, note that one of goal is maximizing the score in the training. Now we state the following Theorem.

Theorem 3.2 (in Sokolic [19]). *Assume that a DNN f classifies a training sample $\mathbf{s}_i$ with the score $o(\mathbf{s}_i) > 0$. Then the classification margin $\Gamma(\mathbf{s}_i)$ can be bounded by*

$$\Gamma(\mathbf{s}_i) \geq \frac{o(\mathbf{s}_i)}{\sup_{\mathbf{x}:\|\mathbf{x}-\mathbf{x}_i\|_2 \leq \Gamma(\mathbf{s}_i)} \parallel \mathbf{J}(\mathbf{x}) \parallel_2} \tag{3.14}$$

$$\geq \frac{o(\mathbf{s}_i)}{\sup_{\mathbf{x}\in \mathrm{conv}(\mathbb{X})} \parallel \mathbf{J}(\mathbf{x}) \parallel_2} \tag{3.15}$$

$$\geq \frac{o(\mathbf{s}_i)}{\prod_{\mathbf{W}\in\mathbb{W}} \parallel \mathbf{W} \parallel_2} \tag{3.16}$$

$$\geq \frac{o(\mathbf{s}_i)}{\prod_{\mathbf{W}\in\mathbb{W}} \parallel \mathbf{W} \parallel_F}, \tag{3.17}$$

where $\mathbb{W}$ is a set of weight matrices of the DNN f.

In the result of Theorem 3.2, we could obtain several facts. The equation (3.16) may be an answer why the weight decay [11] generalize the DNNs well. By the relation between inequalities (3.14) and (3.16), the norm of Jacobian matrix may be more effective than the norm of weight matrix if we focus on aspect to lower the generalization error. Some results to be shown later support these opinions.

Now we could conclude the following Corollary adapted from *Corollary 3* in Sokolic [19].

Corollary 3.3. *Let $\mathcal{C}$ be a classifier with a classification margin Γ and $\mathbb{X}$ a bounded d-dimensional subspace of $\mathbb{R}^n$. Suppose that there exists $\mu \in \mathbb{R}$ such that $\parallel \mathbf{x} \parallel_2 \leq \mu$ for all $\mathbf{x} \in \mathbb{X}$. Then for any $\delta > 0$, with probability at least $1 - \delta$,*

$$\mathrm{GE}(\mathcal{C}) \leq \sqrt{\frac{2^{d+1} \cdot N \cdot \left(\mu\sqrt{d}\right)^d \cdot \log 2 \cdot \left(\sup_{\mathbf{x}:\|\mathbf{x}-\mathbf{x}_i\|_2 \leq \Gamma(\mathbf{s}_i)} \parallel \mathbf{J}(\mathbf{x}) \parallel_2\right)^d}{o(\mathbf{s}_i)^d \, m}}, \tag{3.18}$$

if we neglect the term $\sqrt{\frac{2\log\left(\frac{1}{\delta}\right)}{m}}$.

Proof. The proof is followed by Corollary 2.9 and Theorem 3.2. □

Since the norm of Jacobian matrix can regularize the DNNs, Sokolic [19] suggests the *Jacobian regularizer* as follows :

$$\mathbf{R}_J(f) = \frac{1}{m} \sum_{i=1}^{m} \| \mathbf{J}(\mathbf{x}_i) \|_2^2, \tag{3.19}$$

where $\{\mathbf{x}_i\}_{i=1}^m$ are training samples and $\mathbf{J}(\mathbf{x}_i)$ is the Jacobian matrix of f at a training sample $\mathbf{x}_i$. Hence the loss of the DNN is

$$\textit{total loss} = \textit{original loss} + \mathbf{R}_J. \tag{3.20}$$

However, we suggest a constant λ, the *regularizer constant*, so that we could handle the effect of Jacobian regularizer. Therefore

$$\textit{total loss} = \textit{original loss} + \lambda \cdot \mathbf{R}_J. \tag{3.21}$$

In the next chapter, we use the Jacobian regularizer and attempt to demonstrate our theory.

Chapter 4

Experiments

In here, we apply the theory to real datasets. We use the MNIST [14] and the CIFAR10 [10]. The MNIST dataset is a database which consists of 28×28 handwritten digits with a training set of 60,000 examples and a test set of 10,000 examples. The CIFAR10 dataset is a database which consists of 32×32 color images in 10 classes with a training set of 50,000 examples and a test set of 10,000 examples. The classes are airplane, automobile, bird, cat, etc. These datasets are popular in DNNs, so it could be the yardstick how this experiments are reasonable and comparable to the other experiments of other works. We have implemented our experiments in Theano [1], which includes the Automatic Differentiation. Our experiments were implemented on Tesla K80.

We compare the performances of the Convolutional Neural Network(CNN) [13] with the Jacobian regularizer or with the L^2-regularization(weight decay) [11]. The L^2-regularization is a standard regularization method in DNNs. We propose to show the results of our experiments. In addition, We show the visualization by using t-SNE, the technique suggested by [15].

4.1 Model of Convolutional Neural Network

In our work, what we want to know is the effect of the Jacobian regularizer. Thus we do not use other techniques likewise the dropout [20], the Xavier initialization [6], etc. In here, we use a standard CNN likewise, *LeNet* [14].

		# training samples				
		1,000	5,000	10,000	30,000	60,000
L^2-reg.	train	100.0	100.0	99.95	99.73	99.75
	test	93.85	97.18	98.03	98.63	98.92
$\lambda = 1.0$	train	97.52	95.23	94.18	94.62	99.99
	test	93.91	94.73	94.16	94.79	98.87
$\lambda = 0.5$	train	99.08	96.67	96.87	96.70	96.44
	test	**94.83**	96.19	96.75	96.91	97.14
$\lambda = 0.1$	train	100.0	99.67	99.50	99.25	98.85
	test	93.61	**97.61**	**98.29**	98.77	98.78
$\lambda = 0.05$	train	100.0	99.93	99.52	99.54	99.29
	test	93.76	97.54	98.16	98.78	**99.02**
$\lambda = 0.01$	train	100.0	100.0	99.62	99.98	99.41
	test	93.54	97.27	98.10	**98.97**	98.94

Table 4.1: Accuracy(%) on the MNIST. **1st row** : L^2-regularization(weight decay). **2nd $\sim$ 6th rows** : Jacobian regularizer with a constant λ. We show the train accuracy and test accuracy both.

We use a 4-layer CNN as follows: (5×5)-32 filters, (2×2)-max-pool, (5×5)-32 filters, (2×2)-max-pool, flat-layer and the network ends with a 10-way fully connected layer with a softmax. The cost function is the cross-entropy and we use the Stochastic Gradient Descent(SGD) algorithm with batch-size is 256. The activation function in all hidden layers is a ReLU, identically. Unlike Sokolic [19], we use the various regularizer constants of Jacobian regularizer.

4.2 Results

4.2.1 Result of Convolutional Neural Network

MNIST

Since the performance of CNNs on the MNIST dataset is originally very high, the difference between the accuracy of the L^2-regularization and the

		# training samples				
		1,000	5,000	10,000	25,000	50,000
L^2-reg.	train	99.21	100.0	99.99	81.05	80.43
	test	29.44	37.73	41.16	46.27	49.95
$\lambda = 1.0$	train	40.17	34.91	27.53	30.08	31.75
	test	30.26	33.99	29.43	27.62	30.74
$\lambda = 0.5$	train	52.48	41.98	34.67	35.03	36.15
	test	**33.06**	38.54	34.49	35.45	33.59
$\lambda = 0.1$	train	73.30	53.02	49.52	46.04	43.99
	test	31.35	**42.17**	42.74	44.82	40.78
$\lambda = 0.05$	train	85.02	62.10	54.47	49.87	47.24
	test	30.59	41.73	45.42	47.21	45.06
$\lambda = 0.01$	train	98.17	83.03	70.37	58.46	53.74
	test	29.54	40.32	**45.44**	**49.31**	**51.61**

Table 4.2: Accuracy(%) on the CIFAR10. **1st row** : L^2-regularization(weight decay). **2nd $\sim$ 6th rows** : Jacobian regularizer with a constant λ. We show the train accuracy and test accuracy both.

Jacobian regularizer is very small. However, if we only focused on the aspect of regularization of DNN, then the Jacobian regularizer provides a dramatic performance. Unlike Sokolic [19], we use various regularizer constants. We demonstrate that these are effective. The results are in Table 4.1.

We observe that the Jacobian regularizer provides a similar performance of the L^2-regularization. Some cases outperform the result of the L^2-regularization. According to the Jacobian regularizer constant λ, the performances are little different. For example, the CNN trained using 1,000 training samples provides the highest performance when $\lambda = 0.5$, 94.83%, and the CNN trained using 60,000 training samples provides the highest accuracy when $\lambda = 0.05$, 99.02%.

CIFAR10

While the MNIST dataset is a database which consists of simple figures, the CIFAR10 dataset is a database which consists of complex figures. Thus the accuracy of our simple CNN on the CIFAR10 is relatively low. Also it is easy to overfit our simple CNN on the CIFAR10. Thus we could observe the effect of regularization with the Jacobian regularizer in here. The results are in Table 4.2.

We observe that the Jacobian regularizer provides a dramatic performance on the aspect of regularization. While the CNN trained using relatively less training samples with the L^2-regularization reveals the overfitting problem, the CNN trained with the Jacobian regularizer does not, relatively. Note that the difference of the training accuracy and test accuracy in results trained by the Jacobian regularizer is relatively small. Also the Jacobian regularizer outperforms the L^2-regularization. For example, the CNN trained using 10,000, 25,000 and 50,000 training samples provides the highest accuracy when $\lambda = 0.01$, 45.44%, 49.31% and 51.61%, respectively.

4.2.2 Visualization using t-SNE

In here, we show the visualization of our result by using t-SNE. The t-SNE(Stochastic Neighbor Embedding) is a method, suggested by [15], to visualize high dimensional data in a low dimensional map. We use the feature map from the layer right before the fully connected layer, by utilizing 10,000 test data. So we can only use the result passed through convolutional layers. The number of scattered data on the map is 10,000 and the learning rate of t-SNE is 1000.

The visualization shows that the classified data are clustered well in each label. Thus we can observe how well classify given datasets in our experiments. The figure 4.1 shows that the scattered data trained by using the Jacobian regularizer are similar to the data trained by using the L^2-regularization. Some classified labels trained by the Jacobian regularizer may be better clustered than by the L^2-regularization.

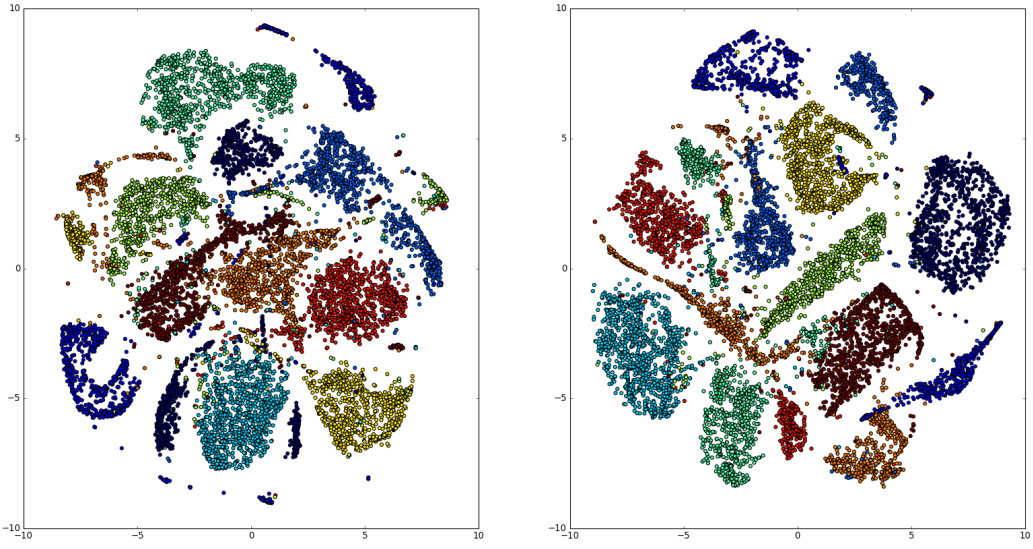

Figure 4.1: Visualization using t-SNE with a learning rate 1000 on MNIST. L^2-regularization (Left). Jacobian regularization with $\lambda = 0.1$ (Right).

Chapter 5

Conclusion

In this paper, we studied the notion of the robustness and generalization error in Deep Neural Networks. We assumed that the input space is a bounded d-dimensional subspace of $\mathbb{R}^n$ and obtained the covering number of the suggested input space. Thus we derived the new upper bound of the generalization error. By the upper bound of the generalization error, we tried to explain what factors have an effect to the generalization of Deep Neural Networks. We can come to the conclusion that the normalization of input data is helpful to generalize Deep Neural networks. Also the norm of the Jacobian matrix in Deep Neural Networks affect to the upper bound of the generalization error. Thus we observed that the Jacobian regularizer, based on the Jacobian matrix, leverages the generalization error.

Moreover, we provided the experimental analysis to support our theory. We compared the regularization methods with L^2-regularization and with Jacobian regularizer. The Jacobian regularization performs as well as L^2-regularization. In some case, the result of the Jacobian regularization outperforms the result of the L^2-regularization. We visualized the classified data to a two dimensional map by using t-SNE. Thus we demonstrated that the Jacobian regularization performs well in a classification problem.

Bibliography

[1] R. Al-Rfou, G. Alain, A. Almahairi, C. Angermueller, D. Bahdanau, N. Ballas, F. Bastien, J. Bayer, A. Belikov, A. Belopolsky, et al., *Theano: A python framework for fast computation of mathematical expressions*, arXiv preprint, (2016).

[2] P. L. Bartlett, *The sample complexity of pattern classification with neural networks: the size of the weights is more important than the size of the network*, IEEE transactions on Information Theory, 44 (1998), pp. 525–536.

[3] A. Bellet and A. Habrard, *Robustness and generalization for metric learning*, Neurocomputing, 151 (2015), pp. 259–267.

[4] R. Girshick, *Fast r-cnn*, in Proceedings of the IEEE international conference on computer vision, 2015, pp. 1440–1448.

[5] R. Girshick, J. Donahue, T. Darrell, and J. Malik, *Rich feature hierarchies for accurate object detection and semantic segmentation*, in Proceedings of the IEEE conference on computer vision and pattern recognition, 2014, pp. 580–587.

[6] X. Glorot and Y. Bengio, *Understanding the difficulty of training deep feedforward neural networks*, in Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, 2010, pp. 249–256.

[7] X. Glorot, A. Bordes, and Y. Bengio, *Deep sparse rectifier neural networks*, in Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, 2011, pp. 315–323.

[8] K. He, G. Gkioxari, P. Dollár, and R. Girshick, *Mask r-cnn*, arXiv preprint arXiv:1703.06870, (2017).

[9] S. Ioffe and C. Szegedy, *Batch normalization: Accelerating deep network training by reducing internal covariate shift*, in International Conference on Machine Learning, 2015, pp. 448–456.

[10] A. Krizhevsky and G. Hinton, *Learning multiple layers of features from tiny images*, (2009).

[11] A. Krogh and J. A. Hertz, *A simple weight decay can improve generalization*, in Advances in neural information processing systems, 1992, pp. 950–957.

[12] Y. LeCun, Y. Bengio, and G. Hinton, *Deep learning*, Nature, 521 (2015), pp. 436–444.

[13] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, *Backpropagation applied to handwritten zip code recognition*, Neural computation, 1 (1989), pp. 541–551.

[14] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, *Gradient-based learning applied to document recognition*, Proceedings of the IEEE, 86 (1998), pp. 2278–2324.

[15] L. v. d. Maaten and G. Hinton, *Visualizing data using t-sne*, Journal of Machine Learning Research, 9 (2008), pp. 2579–2605.

[16] S. Ren, K. He, R. Girshick, and J. Sun, *Faster r-cnn: Towards real-time object detection with region proposal networks*, in Advances in neural information processing systems, 2015, pp. 91–99.

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

[18] E. Shelhamer, J. Long, and T. Darrell, *Fully convolutional networks for semantic segmentation*, IEEE transactions on pattern analysis and machine intelligence, 39 (2017), pp. 640–651.

[19] J. Sokolic, R. Giryes, G. Sapiro, and M. R. Rodrigues, *Robust large margin deep neural networks*, IEEE Transactions on Signal Processing, (2017).

[20] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, *Dropout: a simple way to prevent neural networks from overfitting.*, Journal of machine learning research, 15 (2014), pp. 1929–1958.

[21] N. Verma, *Distance preserving embeddings for general n-dimensional manifolds*, The Journal of Machine Learning Research, 14 (2013), pp. 2415–2448.

[22] H. Xu and S. Mannor, *Robustness and generalization*, Machine learning, 86 (2012), pp. 391–423.

Appendix A

Appendix

A.1 Equation

In here, we state the equation (3.6) in detail. It is easy to know that $\frac{d\hat{\mathbf{u}}^L}{d\mathbf{u}^{L-1}} = \mathbf{W}^L$. So it suffices to state that $\frac{d\mathbf{u}^L}{d\hat{\mathbf{u}}^L}$. For convenience, we omit L. Note that $\frac{d\mathbf{u}}{d\hat{\mathbf{u}}}$ is the Jacobian matrix. We denote $(\frac{d\mathbf{u}}{d\hat{\mathbf{u}}})_{ij}$ by a (i,j)-th component of $\frac{d\mathbf{u}}{d\hat{\mathbf{u}}}$. So we have

$$(\frac{d\mathbf{u}}{d\hat{\mathbf{u}}})_{ii} = \frac{e^{\hat{\mathbf{u}}_i} \cdot (\sum_k e^{\mathbf{u}_k}) - e^{\hat{\mathbf{u}}_i} \cdot e^{\hat{\mathbf{u}}_i}}{(\sum_k e^{\mathbf{u}_k})^2} = \frac{-e^{2\hat{\mathbf{u}}_i}}{(\sum_k e^{\mathbf{u}_k})^2} + \frac{e^{\hat{\mathbf{u}}_i}}{\sum_k e^{\mathbf{u}_k}},$$

and for i, j satisfying $i \neq j$,

$$(\frac{d\mathbf{u}}{d\hat{\mathbf{u}}})_{ij} = \frac{-e^{\hat{\mathbf{u}}_i} \cdot e^{\hat{\mathbf{u}}_j}}{(\sum_k e^{\mathbf{u}_k})^2}$$

Therefore

$$\frac{d\mathbf{u}^L}{d\mathbf{u}^{L-1}} = (-[\hat{\mathbf{u}}^L]_{softmax} \cdot [\hat{\mathbf{u}}^L]^T_{softmax} + \mathrm{diag}([\hat{\mathbf{u}}^L]_{softmax})) \cdot \mathbf{W}^L.$$

A.2 Proof of Lemma

In here, we prove the Lemma 2.8. The idea is similar to [17]. Let $\{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_d\}$ be an orthonormal basis of $\mathbb{X}$. For $\mathbf{x} \in \mathbb{X}$, $\mathbf{x}$ can be uniquely expressed by $\mathbf{x} = \sum_{i=1}^{d} a_i \mathbf{v}_i$. Note that $\max_{i\in[d]} |a_i| = \| \mathbf{a} \|_\infty \leq \| a \|_2 = \| \mathbf{x} \|_2 \leq \mu$ since $\{\mathbf{v}_i\}_{i=1}^{d}$ is an orthonormal basis. Let $\epsilon \in \mathbb{R}$ and consider a subset C of $\mathbb{X}$ defined by

$$C = \{\sum_{i=1}^{d} a_i \mathbf{v}_i : a_i \in \{-\mu, -\mu+\epsilon, -\mu+2\epsilon, \ldots, \mu-\epsilon, \mu\}\} \tag{A.1}$$

We claim that for all $\mathbf{x} \in \mathbb{X}$, there exists $\hat{\mathbf{x}} \in C$ such that $\| \mathbf{x} - \hat{\mathbf{x}} \|_2 \leq \frac{\epsilon\sqrt{d}}{2}$. Note that we can choose $\hat{\mathbf{x}} \in C$ satisfying the follow property :

$$\begin{aligned} \| \mathbf{x} - \hat{\mathbf{x}} \|_2^2 = \| \sum_{i=1}^{d} (a_i - \hat{a}_i)\mathbf{v}_i \|_2^2 &\leq \sum_{i=1}^{d} |a_i - \hat{a}_i|^2 \| \mathbf{v}_i \|_2^2 \\ &\leq \sum_{i=1}^{d} \left(\frac{\epsilon}{2}\right)^2 \| \mathbf{v}_i \|_2^2 \leq \frac{\epsilon^2}{4} \sum_{i=1}^{d} \| \mathbf{v}_i \|_2^2 = \frac{\epsilon^2 d}{4} \end{aligned}$$

So there exists $\hat{\mathbf{x}} \in C$ with $\| \mathbf{x} - \hat{\mathbf{x}} \|_2 \leq \frac{\epsilon\sqrt{d}}{2}$. Thus we can conclude that C is a $\frac{\epsilon\sqrt{d}}{2}$-cover of $\mathbb{X}$. Now let ϵ be $\frac{2\gamma}{\sqrt{d}}$, then $\frac{\epsilon\sqrt{d}}{2} = \gamma$. Therefore we have

$$\mathcal{N}(\mathbb{X}; \gamma) \leq |C| \leq \left(\frac{2\mu}{\epsilon}\right)^d = \left(\frac{\mu\sqrt{d}}{\gamma}\right)^d. \tag{A.2}$$

국문초록

깊은 신경망의 일반화는 중요한 문제이다. 깊은 신경망이 여러 분야에서 좋은 성능을 내고 있지만 깊은 신경망이 일반화가 잘 되는 이유를 이론적으로 설명하기는 어렵다. Xu에 의해서 제시된 강건성은 깊은 신경망의 일반화에 대해 설명할 수 있는 좋은 개념이다. 이 논문에서 우리는 Xu에 의해 제시된 강건성과 일반화 오차를 알아보고 여기서 파생된 내용을 이야기한다. 추가적으로 우리는 입력 공간이 $\mathbb{R}^n$에서의 유계인 d 차원 부분공간이라고 가정하고, 깊은 신경망에서의 일반화 오차의 상한을 새롭게 유도한다. Sokolic이 제시했던 깊은 신경망에서의 야코비 행렬을 기반으로 한 야코비 정칙화를 이용하여 우리의 이론을 실험을 통해 확인하고자 한다. 추가로 t–SNE를 이용하여 실험결과를 시각화한다.

주요어휘: 강건성, 일반화 오차, 야코비 정칙화, 깊은 신경망
학번: 2016-20246