SPARSE DEEP NEURAL NETWORKS USING $L_{1,\infty}$-WEIGHT NORMALIZATION

Ming Wen[*], Yixi Xu[*], Yunling Zheng, Zhouwang Yang, Xiao Wang

University of Science and Technology of China and Purdue University

Abstract:

Deep neural networks have recently demonstrated an amazing performance on many challenging tasks. Overfitting is one of the notorious features for DNNs. Empirical evidence suggests that inducing sparsity can relieve overfitting, and weight normalization can accelerate the algorithm convergence. In this paper, we study $L_{1,\infty}$-weight normalization for deep neural networks with bias neurons to achieve the sparse architecture. We theoretically establish the generalization error bounds for both regression and classification under the $L_{1,\infty}$-weight normalization. It is shown that the upper bounds are independent of the network width and $\sqrt{k}$-dependence on the network depth k, which are the best available bounds for networks with bias neurons. These results provide theoretical justifications on the usage of such weight normalization to reduce the generalization error. We also develop an easily implemented gradient projection descent algorithm to practically obtain a sparse neural network. We perform various experiments to validate our theory and demonstrate the effectiveness of the resulting approach.

Key words and phrases: Deep neural networks, Generalization, Overfitting, Rademarcher complexity, Sparsity.

[*]Equal contribution.

1. Introduction

Deep neural networks (DNNs) have recently attracted a lot of attentions due to their successful applications on many real-word applications (Goodfellow et al., 2016). On the one hand, the new advancement on optimization with stochastic gradient descent (SGD) and graphical processing units (GPUs) makes the DNN training easy to scale to millions of parameters (Krizhevsky et al., 2012; Jaderberg et al., 2015; Szegedy et al., 2015). On the other hand, overfitting becomes a notorious feature of DNNs, which may lead to poor generalization. Recent works (Han et al., 2016; Louizos et al., 2017; Molchanov et al., 2017) show that the networks can be pruned significantly without any loss in accuracy. In the meanwhile, many other methods have also been developed to address the issue of overfitting, which includes early stopping, weight penalties of various kinds such as L_1 and L_2 regularizations, weight sharing (Nowlan and Hinton, 1992), and dropout (Srivastava et al., 2014).

Empirical evidence suggests that inducing sparsity can relieve overfitting and save computation resources. A common strategy is to apply sparsity-inducing regularizers such as L_0 penalty (Louizos et al., 2018) or the total number of parameters in the network (Srinivas et al., 2017). However, theoretical investigations or justifications on sparse DNNs are less explored in the literature.

Weight normalization, by bounding the Euclidean norm of the incoming weights of each unit, has shown to be able to accelerate the convergence of stochastic gradient descent optimization across many applications (Salimans and Kingma, 2016). In this paper, we borrow the strength of weight normalization and induce the sparsity by bounding the $L_{1,\infty}$ norm of the weight matrix (including bias) for each layer. By doing this, we are able to induce the sparsity in a systematic way. Furthermore, we have developed capacity control for such models. It is shown that the generalization error upper bounds are independent of the network width and $\sqrt{k}$-dependence on the depth k of the network, which are the best available bounds for networks *with bias neurons*. Our results provide theoretical justifications on the usage of such weight normalization, which leads to a sparse DNN. At the same time, the generalization error has the minimal dependence on the network architecture. $L_{1,\infty}$ norm-constrained fully connected DNNs *without bias neurons* were investigated in prior studies (Bartlett, 1998; Neyshabur et al., 2015; Sun et al., 2016; Golowich et al., 2018). Rademacher complexity bounds in (Bartlett, 1998; Neyshabur et al., 2015; Sun et al., 2016) is 2^k times larger than our result in Theorem 1 even without bias neurons in each hidden layer. Furthermore, it is hard to extend the work of (Golowich et al., 2018) to the fully connected DNNs with bias neurons, especially when the activation function, such as the tanh activation function, fails to map the bias neuron to one. As a comparison, our result is applicable to all Lipschitz continuous activation functions.

The overall contributions of the paper are: (a). We have theoretically established the generalization error bounds for both regression and classification under the $L_{1,\infty}$-weight normalization for networks *with bias neurons*; (b). We have developed an easily implemented gradient projection descent algorithm to practically obtain a sparse neural network; (c). We have performed various experiments to validate our theory and demonstrate the effectiveness of the resulting approach.

The paper is organized as follows. In Section 2, we define the sparse DNNs. Section 3 gives the Rademacher complexities, the generalization bounds for both regression and classification. In Section 4, we propose a gradient projection descent algorithm. Section 5 includes both synthetic and real world experiments to validate our theoretical findings.

2. The Model

In this section, we define the general prediction problem and introduce the sparse deep neural networks (DNNs).

2.1 The General Prediction Problem

Assume that $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ are n independent random variables on $\mathcal{X} \subseteq \mathbb{R}^{m_1}$, $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_n$ are on $\mathcal{Z} \subseteq \mathbb{R}^{m_2}$, $y_1, \ldots, y_n$ are on $\mathcal{Y} \subseteq \mathbb{R}$, and the noise $\varepsilon_1, \ldots, \varepsilon_n$ are independent while satisfying that $\mathbb{E}(\varepsilon_i) = 0$. The general prediction problem is defined as

$$\begin{aligned} \boldsymbol{z}_i &= f(\boldsymbol{x}_i) + \varepsilon_i \\ y_i &= t(\boldsymbol{z}_i), \end{aligned} \tag{2.1}$$

where $t : \mathcal{Z} \to \mathcal{Y}$ is a fixed function related to the prediction problem, and $f : \mathcal{X} \to \mathcal{Z}$ is an unknown function. We provide two examples in order to illustrate how to adapt equation (2.1) to different settings. For regression, we have $m_2 = 1$, $\mathcal{Z} = \mathcal{Y}$, and $t(z) = z$. While for classification, we could define m_2 as the number of classes, $\mathcal{Y} = \{1, 2, \cdots, m_2\}$, and $t = \text{argmax}$.

2.2 Overfitting

We first provide a formal definition on overfitting. Let $L(f(\cdot), \cdot) : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$ be the loss function. Define the expected and empirical risks, respectively, as

$$\mathbb{E}_L(f) = \mathbb{E}_{(\boldsymbol{x},y)\sim\mathcal{D}}[L(f(\boldsymbol{x}), y)], \quad \widehat{\mathbb{E}}_L(f) = \frac{1}{n}\sum_{i=1}^{n} L(f(\boldsymbol{x}_i), y_i),$$

where $\mathcal{D}$ is the underlying distribution of $(\boldsymbol{x}, y)$. In practice, the empirical risk is corresponding to the training error, and the expected risk is corresponding to the testing error. Generally, a learning algorithm is said to *overfit* if it is more accurate in fitting known data but less accurate in predicting new data. Mathematically, the difference between the expected risk and the empirical risk, called *generalization error*, is a measure of how accurately an algorithm is able to predict outcome values for previously unseen data. Let

$$\mathcal{E}_L(f) = \Big| \ \mathbb{E}_L(f) - \widehat{\mathbb{E}}_L(f) \ \Big|.$$

The performance of DNNs relies on the balance between the empirical risk and generalization error. On the one hand, complex structures usually overfit the data, while achieving a low empirical risk. On the other hand, simple models generalize well, but might underfit the data. In practice, it is common to use a neural net with billions parameters (e.g.

Simonyan and Zisserman (2015)), and the training error could be even reduced to exactly 0 in many cases. Thus, in this paper, we will focus on the generalization error only.

Our goal is to control the generalization error $\mathcal{E}_L(f)$ and make it less sensitive to the network architecture when adopting a DNN model. This generalization error bound can be studied using *Rademacher complexity* by some standard techniques in Mohri et al. (2012). Under some mild conditions, it is sufficient to bound Rademacher complexity in order to control the generalization error. In this paper, we establish non-asymptotic uniform error bounds for $\mathcal{E}_L(f)$ when f is a sparse DNN. These bounds have the minimal dependence on the network structure.

2.3 Sparse DNNs

We begin with some notations for fully-connected neural networks. A neural network on $\mathbb{R}^{d_0} \to \mathbb{R}^{d_{k+1}}$ with k hidden layers is defined by a set of $k+1$ affine transformations $T_1 : \mathbb{R}^{d_0} \to \mathbb{R}^{d_1}, T_2 : \mathbb{R}^{d_1} \to \mathbb{R}^{d_2}, \cdots, T_{k+1} : \mathbb{R}^{d_k} \to \mathbb{R}^{d_{k+1}}$ and an activation function σ. In this paper, we consider activation functions satisfying that $\sigma(0) = 0$. Note that this condition holds for widely used activation functions including ReLU and tanh. The affine transformations are parameterized by $T_\ell(\mathbf{u}) = \boldsymbol{W}_\ell^T \mathbf{u} + \mathbf{B}_\ell$, where $\boldsymbol{W}_\ell \in \mathbb{R}^{d_{\ell-1} \times d_\ell}$ and $\mathbf{B}_\ell \in \mathbb{R}^{d_\ell}$ for $\ell = 1, \cdots, k+1$. The function represented by this neural network is

$$f(x) = T_{k+1} \circ \sigma \circ T_k \circ \cdots \circ \sigma \circ T_1 \circ \boldsymbol{x}.$$

Before introducing the sparse DNNs, we build an augmented layer for each hidden layer by appending the bias neuron 1 to the original layer, and then combine the weight matrix and the bias vector to form a new matrix. We define the first hidden layer as

$$f_1(\boldsymbol{x}) = T_1 \circ \boldsymbol{x} \triangleq \langle \tilde{\mathbf{V}}_1, (1, \boldsymbol{x}^T)^T \rangle,$$

where $\tilde{\mathbf{V}}_1 = (\mathbf{B}_1, \boldsymbol{W}_1^T)^T \in \mathbb{R}^{(d_0+1)\times d_1}$.

Sequentially for $\ell = 2, \cdots, k$, define the ℓth hidden layer as

$$f_\ell(\boldsymbol{x}) = T_\ell \circ \sigma \circ f_{\ell-1}(\boldsymbol{x}) \triangleq \langle \tilde{\mathbf{V}}_\ell, (1, \sigma \circ f_{\ell-1}^T(\boldsymbol{x}))^T \rangle,$$

where $\tilde{\mathbf{V}}_\ell = (\mathbf{B}_\ell, \boldsymbol{W}_\ell^T)^T \in \mathbb{R}^{(d_{\ell-1}+1)\times d_\ell}$. The output layer is

$$f(\boldsymbol{x}) = T_{k+1} \circ \sigma \circ f_k(\boldsymbol{x}) \triangleq \langle \tilde{\mathbf{V}}_{k+1}, (1, \sigma \circ f_k^T(\boldsymbol{x}))^T \rangle,$$

where $\tilde{\mathbf{V}}_{k+1} = (\mathbf{B}_{k+1}, \boldsymbol{W}_{k+1}^T)^T \in \mathbb{R}^{(d_k+1)\times d_{k+1}}$.

The sparsity of the DNN could be controlled by setting proper constraints for the $L_{1,\infty}$ norm of each hidden layer, where the $L_{1,\infty}$ norm of a $s_1 \times s_2$ matrix A is defined as

$$\|A\|_{1,\infty} = \max_j \left(\sum_{i=1}^{s_1} |a_{ij}| \right).$$

Specifically, define $\mathcal{SN}_{c,\boldsymbol{O}}^{k,\boldsymbol{d},\sigma}$ as the collection of all sparse DNNs $f(\boldsymbol{x}) = T_{k+1} \circ \sigma \circ T_k \circ \cdots \circ \sigma \circ T_1 \circ \boldsymbol{x}$ satisfying:

(a) It has k hidden layers;

(b) The number of neurons in the ℓth hidden layer is d_ℓ for $\ell = 1, 2, \cdots, k$. The dimension of input is d_0, and output d_{k+1};

(c) $\|T_\ell\|_{1,\infty} \triangleq \left\|\tilde{\mathbf{V}}_\ell\right\|_{1,\infty} \leq c$ for $\ell = 1, \cdots, k$;

(d) The L_1 norm of the jth column of $\tilde{\mathbf{V}}_{k+1}$ is bounded by the jth element of $\boldsymbol{o}$: $\left\|\tilde{\mathbf{V}}_{k+1}[\cdot, j]\right\|_1 \leq o_j$ for $j = 1, \cdots, d_{k+1}$.

We call c the normalization constant in the rest of the paper. Furthermore, define the collection of the sparse DNNs without any constraint on the output layer as

$$\mathcal{S}_c^{k,\boldsymbol{d},\sigma} = \cup_{\boldsymbol{o}\geq \mathbf{0}} \mathcal{SN}_{c,\boldsymbol{o}}^{k,\boldsymbol{d},\sigma}.$$

In order to obtain a sparse neural network, we need to transform our understanding of a problem into a loss function $L(\cdot,\cdot)$. Then it is equivalent to solve the optimization problem

$$\min_f \left\{ \sum_{i=1}^n L(f(\boldsymbol{x}_i), y_i) \,\middle|\, f \in \mathcal{S}_c^{k,\boldsymbol{d},\sigma} \right\}, \tag{2.2}$$

where $(\boldsymbol{x}_1, y_1), \cdots, (\boldsymbol{x}_n, y_n) \in \mathbb{R}^{m_1 \times 1}$ are the samples, k and $\boldsymbol{d}$ define the depth and widths of the DNNs, and the normalization constant c controls the sparsity. We focus more on the influence of c on the generalization behavior as well as the sparsity of the DNN, and we will provide the generalization bounds for the sparse DNNs with unconstrained output layers.

3. The Learning Theory

In this section, assume that $\mathcal{X} = [-1, 1]^{m_1}$, and the activation function σ is ρ_σ-Lipschitz continuous. Note that ReLU and tanh are both 1-Lipschitz continuous.

3.1 Rademacher Complexities

The *empirical Rademacher complexity* of the hypothesis class $\mathcal{F}$ with respect to a data set $S = \{z_1 \dots z_n\}$ is defined as:

$$\widehat{\mathfrak{R}}_S(\mathcal{F}) = \mathbb{E}_\epsilon \left[\sup_{f \in \mathcal{F}} \left(\frac{1}{n} \sum_{i=1}^{n} \epsilon_i f(z_i) \right) \right],$$

where $\epsilon = \{\epsilon_1 \dots \epsilon_n\}$ are n independent Rademacher random variables. The *Rademacher complexity* of the hypothesis class $\mathcal{F}$ with respect to n samples is defined as:

$$\mathfrak{R}_n(\mathcal{F}) = \mathbb{E}_{S \sim \mathcal{D}^n} \left[\widehat{\mathfrak{R}}_S(\mathcal{F}) \right].$$

In the following theorem, we bound the Rademacher complexity of $\mathcal{SN}_{c,o}^{k,\boldsymbol{d},\sigma}$ when the output dimension is one, which will be used later to obtain the generalization bounds for both regression and classification.

Theorem 1. *Fix the depth $k \geq 0$, the normalization constant $c > 0$, the output layer constraint $o > 0$, the widths $d_\ell \in \mathbb{N}_+$ for $\ell = 1, \cdots, k$, and $d_{k+1} = 1$, then for any set $S = \{\boldsymbol{x}_1, \cdots, \boldsymbol{x}_n\} \subseteq \mathcal{X}$, we have*

$$\widehat{\mathfrak{R}}_S(\mathcal{SN}_{c,o}^{k,\boldsymbol{d},\sigma}) \leq o\sqrt{\frac{(k+1)\log 16}{n}} \left(\sum_{\ell=0}^{k} (c\rho_\sigma)^\ell + (c\rho_\sigma)^k \right) + o(c\rho_\sigma)^k \sqrt{\frac{2\log(2m_1)}{n}}.$$

Furthermore, if $c\rho_\sigma \geq 1$,

$$\widehat{\mathfrak{R}}_S(\mathcal{SN}_{c,o}^{k,\boldsymbol{d},\sigma}) \leq \frac{1}{\sqrt{n}} o(c\rho_\sigma)^k (\sqrt{(k+3)\log 4} + \sqrt{2\log(2m_1)}).$$

Remark 1. When $\log(m_1)$ is small, we briefly summarize the dependence of the above bound on k under different choice of c:

- $c\rho_\sigma < 1$: $O(\sqrt{k}\frac{1-(c\rho_\sigma)^{k+1}}{1-c\rho_\sigma})$
- $c\rho_\sigma \geq 1$: $O(\sqrt{k}(c\rho_\sigma)^k)$

The complexity bound does not depend on the width of the network. In addition to the product of the $L_{1,\infty}$ norms of each layer, the complexity bound depends on the depth k by $O(\sqrt{k})$ when $c\rho_\sigma < 1$, and $\sqrt{k}(c\rho_\sigma)^k$ otherwise.

3.2 Generalization Bounds for Regression

In this section, consider a specific case of equation (2.1), where t is an identity transformation, and $m_2 = 1$. Assume the following conditions in this section:

(A1). $(\boldsymbol{x}, y)$ is a random variable of support $\mathcal{X}\times\mathcal{Y}$ and distribution $\mathcal{D}$, and $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$ is a dataset of n i.i.d. samples drawn from $\mathcal{D}$.

(A2). The normalization constant $c > 0$, the number of hidden layers $k \in [0, \infty)$, and widths $\boldsymbol{d} \in \mathbb{N}_+^{k+2}$ with $d_0 = m_1$ and $d_{k+1} = 1$.

(A3). The loss function $L(f(\boldsymbol{x}), y)$ is 1-Lipschitz continuous on its first argument. In addition, $|L(f(\boldsymbol{x}), y)| \leq 1$.

The following theorem shows the generalization bound that holds uniformly for any sparse DNN in $\mathcal{S}_c^{k,\boldsymbol{d},\sigma}$. Note that the sample $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$ is a dataset randomly drawn from $\mathcal{D}$, the following statement holds with a high probability over the choice of the sample S.

Theorem 2. *Assume Condition (A1)-(A3) hold and* $c\rho_\sigma \geq 1$. *Fix* $\delta \in (0,1)$, *then with probability at least* $1-\delta$ *over the choice of the sample* S, *for* **every** *sparse DNN* $f_T \in \mathcal{S}_c^{k,\boldsymbol{d},\sigma}$, *we have*

$$\begin{aligned}\mathcal{E}_L(f_T) \leq & \sqrt{\frac{\log(\frac{2}{\delta}) + 2\log(\|T_{k+1}\|_1 + 2)}{2n}} + \\ & \frac{2}{\sqrt{n}}(\|T_{k+1}\|_1 + 1)(c\rho_\sigma)^k(\sqrt{(k+3)\log 4} + \sqrt{2\log(2m_1)}).\end{aligned} \tag{3.1}$$

Remark 2. The upper bound is a summation of $\sqrt{\frac{\log(\frac{2}{\delta})+2\log(\|T_{k+1}\|_1+2)}{2n}}$ and $\frac{2}{\sqrt{n}}(\|T_{k+1}\|_1 + 1)(c\rho_\sigma)^k(\sqrt{(k+3)\log 4} + \sqrt{2\log(2m_1)})$. Both terms depend on the sample size n by $n^{-1/2}$. Note that the L_1 norm of the output layer $\|T_{k+1}\|_1$ is data-dependent. In addition to this, the first term depends on the probability $1-\delta$ by $\sqrt{\log(1/\delta)}$. The second term depends on the depth by $\sqrt{k}(c\rho_\sigma)^k$, and the input dimension by $\sqrt{\log m_1}$. The case when $c\rho_\sigma < 1$ is discussed in the supplementary material.

Remark 3. Define the truncated mean square error and the truncated mean absolute error by

$$L_S(f(\boldsymbol{x}), y) = \min\left(\left(\frac{y - f(\boldsymbol{x})}{2}\right)^2, 1\right)$$

and

$$L_A(f(\boldsymbol{x}), y) = \min\left(|y - f(\boldsymbol{x})|, 1\right)$$

accordingly. It is easy to verify that both loss functions satisfy Condition (A3). Thus, we could apply Theorem 2 to both cases.

Remark 4. Condition (A3) is not always met in practice. It could be relaxed by assuming that the loss function $L(f(\boldsymbol{x}), y)$ is B_0-Lipschitz continuous on its first argument, and $|L(f(\boldsymbol{x}), y)| \leq B_0$, where B_0 is a constant. The generalization error could still be bounded by applying Theorem 2 to the loss function L/B_0.

3.3 Generalization Bounds for Classification

In this section, we consider the case of equation (2.1) when $t = \text{argmax}$ and $\mathcal{Y} = \{1, 2, \cdots, m_2\}$. In the rest of the paper, we define the jth element of a vector $\mathbf{z}$ by $z[j]$. In this subsection, assume the following conditions:

(B1). $(\boldsymbol{x}, y)$ is a random variable of support $\mathcal{X} \times \mathcal{Y}$ and distribution $\mathcal{D}$, and $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$ is a dataset of n i.i.d. samples drawn from $\mathcal{D}$.

(B2). The normalization constant $c > 0$, the number of hidden layers $k \in [0, \infty)$, and widths $\boldsymbol{d} \in \mathbb{N}_+^{k+2}$ with $d_0 = m_1$ and $d_{k+1} = m_2$.

The cross-entropy loss function is defined as

$$L_C(f(\boldsymbol{x}), y) = -\log \frac{\exp(f(\boldsymbol{x})[y])}{\sum_j \exp f(\boldsymbol{x})[j]}.$$

We then extend it to the sparse DNNs with unconstrained output layers. For any transformation $T(\mathbf{u}) = V^T(1, \mathbf{u}^T)^T$, define $T[j]$ as $V[, j]$: the jth column of V.

Theorem 3. *Assume Condition (B1)-(B2) hold and* $c\rho_\sigma \geq 1$. *Fix* $\delta \in (0,1)$. *Then, with probability at least* $1-\delta$ *over the choice of* S, *every* $f_T \in \mathcal{S}_c^{k,\boldsymbol{d},\sigma}$ *satisfies that*

$$\mathcal{E}_{L_C}(f_T) \leq O\left(\frac{(c\rho_\sigma)^k}{\sqrt{n}} \|T_{k+1}\|_{1,\infty}\left[\sqrt{\log\frac{1}{\delta}} + \frac{\|T_{k+1}\|_{1,1}}{\|T_{k+1}\|_{1,\infty}}\sqrt{m_2}(\sqrt{k}+\sqrt{\log m_1})\right]\right), \quad (3.2)$$

where $\|T\|_{1,1} = \sum_i \sum_j |v_{ij}|$, *if* $T(x) = \langle \mathbf{V}, (1, \boldsymbol{x})\rangle$.

Remark 5. The upper bound is the product of $(c\rho_\sigma)^k \|T_{k+1}\|_{1,\infty}/\sqrt{n}$ and

$$\sqrt{\log\frac{1}{\delta}} + \frac{\|T_{k+1}\|_{1,1}}{\|T_{k+1}\|_{1,\infty}}\sqrt{m_2}(\sqrt{k}+\sqrt{\log m_1}).$$

The first term includes $(c\rho_\sigma)^k \|T_{k+1}\|_{1,\infty}$, which reflects the range of the neural network f_T. The second term is the summation of $\sqrt{\log\frac{1}{\delta}}$ and $\frac{\|T_{k+1}\|_{1,1}}{\|T_{k+1}\|_{1,\infty}}\sqrt{m_2}(\sqrt{k}+\sqrt{\log m_1})$, where $1-\delta$, k, m_1 and m_2 are the probability, the depth, the input dimension, and the number of classes, respectively. The case when $c\rho_\sigma < 1$ is discussed in the supplementary material. Under this assumption, the generalization bound rely on $c\rho_\sigma$ by $O(\frac{1-(c\rho_\sigma)^{k+1}}{1-c\rho_\sigma})$.

Note that the above generalization bound is independent of the widths of the neural network. Our theoretical results could be easily extended to convolutional neural networks (CNNs), as a CNN could be viewed as a neural net with a sparse structure. Note that our generalization bound is independent of the widths of the neural network. When applying to CNNs, the bound will not depend on the number of kernels, which could be thousands in practice. In addition, it would be interesting to extend our theoretical results to residual neural networks in the future.

Algorithm 1 Gradient Projection Descent Algorithm

```
In each iteration:
Input: $\tilde{\mathbf{V}}^{(t)} = (\tilde{\mathbf{V}}_1^{(t)}, \cdots, \tilde{\mathbf{V}}_k^{(t)})$
for all $\ell = 1, \ldots, k$ do
    $\tilde{\mathbf{V}}_\ell^{(t+1)} := \tilde{\mathbf{V}}_\ell^{(t)} - \gamma_t \nabla L(\tilde{\mathbf{V}}_\ell^{(t)})$,
    where $\gamma_t$ is the stepsize at iteration $t$
    for all columns $\boldsymbol{v}$ in $\mathbf{V}_\ell^{(t+1)}$ do
        if $\|\boldsymbol{v}\|_1 > c$ then
            $\boldsymbol{v} = \text{proj}_{\|\cdot\|_1 \le c} \boldsymbol{v}$ by Algorithm 2
        end if
    end for
end for
Output: $\tilde{\mathbf{V}}^{(t+1)} = (\tilde{\mathbf{V}}_1^{(t+1)}, \cdots, \tilde{\mathbf{V}}_k^{(t+1)})$
```

4. The Algorithm

In this section, we propose a gradient projection descent algorithm to solve the optimization problem equation (2.2).

Recall that for a neural network $f(x) = T_{k+1} \circ \sigma \circ T_k \circ \cdots \circ \sigma \circ T_1 \circ \boldsymbol{x}$ with $T_\ell(\mathbf{u}) = \tilde{\mathbf{V}}_\ell^T (1, \mathbf{u}^T)^T$, we have $f \in \mathcal{S}_c^{k,\boldsymbol{d},\sigma}$ if and only if

$$\left\|\tilde{\mathbf{V}}_\ell\right\|_{1,\infty} \le c \quad \forall \ell \ or \ \left\|\tilde{\mathbf{V}}_\ell[\cdot, j]\right\|_1 \le c \quad \forall\ \ell, j.$$

Algorithm 2 Projection to L_1 norm ball (Duchi et al., 2008)

```
**Input:** $\boldsymbol{v} \in \mathbb{R}^s$, $c$
Sort abs($\boldsymbol{v}$) into $\mu : \mu_1 \geqslant \mu_2 \geqslant \cdots \geqslant \mu_s$
Find $p^* = \max\{p \in [s] : \mu_p - \frac{1}{p}(\sum_{q=1}^{p} \mu_q - c) > 0\}$
Define $\theta = \frac{1}{p^*}(\sum_{q=1}^{p^*} \mu_q - c)$
**Output:** $\boldsymbol{w}$ s.t. $\boldsymbol{w}_p = \mathrm{sgn}(\boldsymbol{v}_p) \cdot \max\{\mathrm{abs}(\boldsymbol{v}_p) - \theta, 0\}$
```

One idea is to solve the Lagrangian of equation (2.2) by a proximal minimization algorithm. However there is no closed form for the proximal operator with the $L_{1,\infty}$ norm. Another idea is to directly solve the constrained optimization problem by a gradient projection descent algorithm. Under this circumstance, the projection to an L_1 norm ball could be efficiently implemented while inducing the sparsity of its output (Duchi et al., 2008).

Our gradient projection descent algorithm could be easily implemented as a variation of any gradient descent method, as shown in Algorithm 1. In each iteration of the original gradient descent method, we project its output to the sparse DNN function class by Algorithm 2. Note that the uniform convergence of the empirical risk to the true risk holds for any hypothesis defined in Theorems 2 and 3. Therefore, it also applies to the gradient projection descent algorithm output.

5. Numerical Results

In this section, we validate our theorem using both simulated and real data experiments. We first design two synthetic experiments to demonstrate the theoretical advantage of the sparse DNNs. Especially, we illustrate the power of $L_{1,\infty}$-weight normalization for high dimensional problems. Furthermore, we apply our algorithm on convolutional layers, and validate our theoretical findings on CIFAR-10 datasets. For each setting, we measure the training error, generalization error, test accuracy, and model sparsity in order to demonstrate c's influence on the generalization ability as well as the sparsity of the model. Recall that the generalization error is the difference between training error and test error. Note that we do not have access to the underlying distribution of the input x and output y. Thus, the generalization error refers to the empirical loss on the test set in all experiments. Besides, test accuracy is the classification accuracy for testing data. The sparsity rate is the ratio of the number of zero parameter estimates to the size of weight matrices. In this section, we will use the following format $d_0 - d_1 - \cdots - d_{k+1}$ to define the architecture of a neural network, where k is the number of hidden layer of the neural network, d_0 is the input dimension, d_{k+1} is the output dimension, and d_i denotes the number of neurons in the the ith hidden layer.

5.1 The Regression Experiment

We evaluate our algorithm on a high-dimensional linear regression problem $y = \boldsymbol{x}^T\boldsymbol{\beta} + \varepsilon$, where the coefficient $\boldsymbol{\beta}$ is a sparse vector. We sample 500 random samples $(\boldsymbol{x}_i, y_i) \in \mathbb{R}^{1000} \times \mathbb{R}, i = 1, \cdots, 500$ for training, and 1500 samples for testing from the distribution below.

1. Generate the coefficient $\boldsymbol{\beta}$ by $\beta_i \sim \text{Unif}(0.15, 150)$ for $i = 1, \cdots, 100$, while setting the rest of $\boldsymbol{\beta}$ to be zero.

2. For $\forall i$, first independently sample an auxiliary variable $\mathbf{z}_i \in \mathbb{R}^{1000}$ from $N(\mathbf{0}, \boldsymbol{I})$. Then generate $\boldsymbol{x}_i$ by $x_{i1} = z_{i1}$, and $x_{ij} = z_{ij} + 0.2(z_{i,j+1} + z_{i,j-1})$ for $j = 2, \cdots, 1000$. Finally sample y_i from $N(\boldsymbol{x}_i^T\boldsymbol{\beta}, 1)$

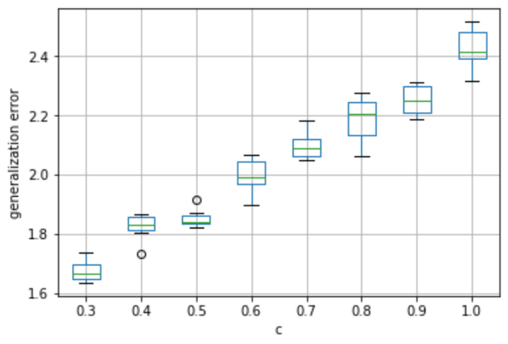
Note that we make the high-dimensional problem even more challenging in the presence of multicollinearity. We train the model with one fully connected layer with 300 output units using ReLU, and the loss function is mean square error. We summarize the results in Table 1, which are estimated by the mean of 10 repeated trials.

As shown in Figure 1a and Figure 1b, as c increases, weight matrices become denser, and the generalization error increases, which matches the conclusion of Theorem 2.

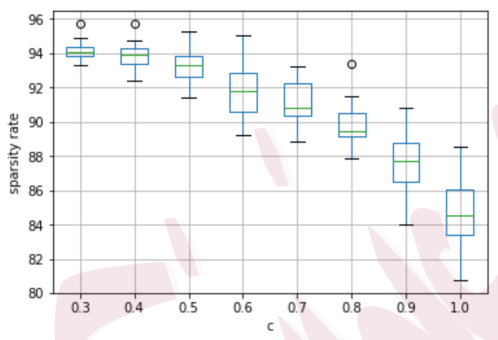
When $c = \infty$, or equivalently with no regularization, even the model fits the training data perfectly, it suffers from serious overfitting. The above problem could be solved by applying $L_{1,\infty}$ weight normalization with a proper c, as the test error would be decreased

	train err	test err	gen err	sparsity %
$c = \infty$	0.000	69.520	69.520	3.02%
$c = 10.00$	0.000	35.571	35.571	41.58%
$c = 2.00$	0.052	8.129	8.077	61.42%
$c = 1.00$	0.131	2.424	2.426	84.69%
$c = 0.90$	0.173	2.424	2.251	87.62%
$c = 0.80$	0.197	2.384	2.186	89.80%
$c = 0.70$	0.235	2.334	2.099	91.09%
$c = 0.60$	0.252	2.247	1.994	91.78%
$c = 0.50$	0.286	2.140	1.854	93.33%
$c = 0.40$	0.387	2.209	1.822	93.88%
$c = 0.30$	0.850	2.526	1.675	94.18%

Table 1: Training error, test error, generalization error and model sparsity for the regression experiment.



(a) Generalization error vs. c.



(b) Sparsity rate vs. c.

Figure 1: Boxplots of generalization error and sparsity rate with different c for the regression experiment.

by more than 95% if set $c = 0.20$.

5.2 The Classification Experiment

We first consider a high-dimensional nonlinear binary classification problem. We sample 500 random samples $(\boldsymbol{x}_i, y_i) \in \mathbb{R}^{500} \times \{0, 1\}, i = 1, \cdots, 500$ for training, and 1000 samples for testing from the distribution below.

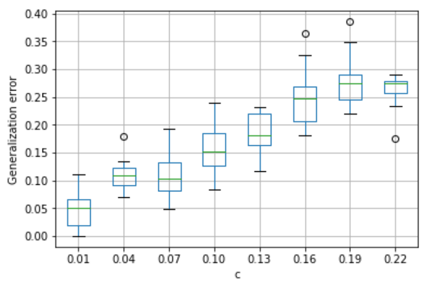
1. Generate $\alpha \sim N(0, 1)$.

2. For $\forall i$, independently sample x_{ij}: the jth element of $\boldsymbol{x}_i$, from $N(\frac{\alpha}{2}, \frac{1}{4})$ for $j = 1, \cdots, 500$, and

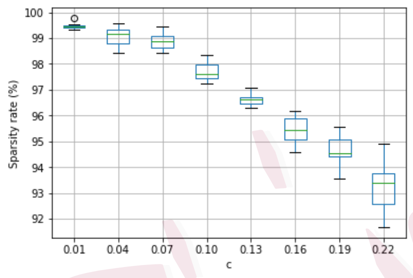
$$y_i = \begin{cases} 1, & e^{x_{i1}} + x_{i2}^2 + 5\sin(x_{i3}x_{i4}) - 3 > 0 \\ 0, & \text{otherwise} \end{cases}$$

	train err	gen err	test acc(%)	sparsity(%)
$c = \infty$	0.005	0.543	71.60	2.39
$c = 1.00$	0.016	0.624	83.50	66.71
$c = 0.50$	0.087	0.337	87.30	68.43
$c = 0.30$	0.053	0.297	88.40	90.78
$c = 0.22$	0.034	0.280	88.93	93.29
$c = 0.19$	0.046	0.273	89.10	94.53
$c = 0.16$	0.030	0.250	89.23	95.40
$c = 0.13$	0.077	0.177	89.93	96.60
$c = 0.10$	0.121	0.155	90.01	97.53
$c = 0.07$	0.207	0.102	90.12	98.98
$c = 0.04$	0.239	0.112	89.57	99.04
$c = 0.01$	0.265	0.068	88.48	99.49

Table 2: Training error, generalization error, test accuracy, and model sparsity for the classification experiment.



(a) Generalization error vs. c.

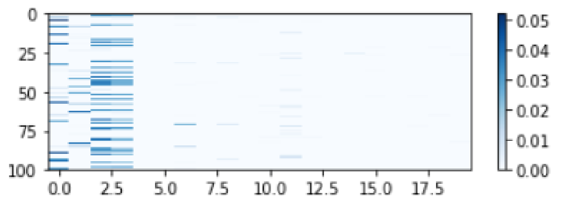


(b) Sparsity rate vs. c.

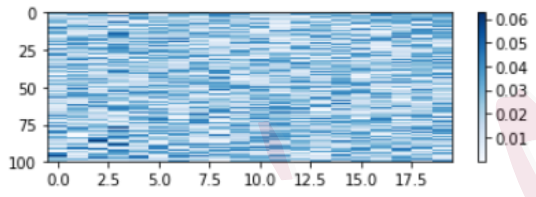
Figure 2: Boxplots of generalization error and sparsity rate with different c for the classification experiment.

We use a 500-100-50-20-2 fully connected neural network with ReLU, and the loss function is cross entropy. We report the results in Table 2, which are estimated by the mean of 10 repeated trials.

As illustrated in Figure 2a, the generalization error decreases as c decreases, which matches the conclusion of Theorem 3. In the meanwhile, the network becomes sparser with a smaller c, which is evident in Figure 2b. However, there is a trade-off between approximation and generalization ability. A smaller c leads to a smaller generalization error. On the other hand, a small c limits the expressive power of the neural network. For example, decreasing c from 0.10 to 0.07 nearly doubles the training error. With no regularization, the model fits the training data perfectly, however it performs poorly on the test dataset. We could improve the test accuracy by 19.4% using $L_{1,\infty}$-weight normalization with $c = 0.07$, while the resulting weight matrix are much sparser as shown



(a) $L_{1,\infty}$-weight normalization with $c = 0.07$.



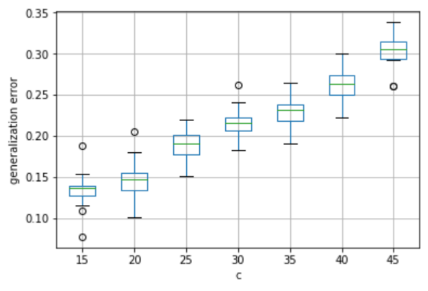
(b) With no regularization.

Figure 3: Visualization of the first twenty columns of the resulting weight matrix representing the first hidden layer with/without $L_{1,\infty}$-weight normalization.

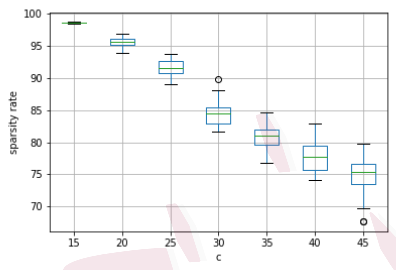
in Figure 3. We also observe that the first 4 columns of the resulting weight matrix are dense, while the others are sparse. It is because that only the first four elements of the input are included in the true model.

5.3 CIFAR-10

We extend our method to convolutional neural networks in the second experiment. CIFAR-10 (Krizhevsky, 2009) consists of 60000 32×32 color images in 10 classes. A small kernel size itself assumes the local sparsity, thus it is not necessary to apply $L_{1,\infty}$-weight normalization to convolutional layers with small kernel sizes. We use a modified VGG-16 to train the model, where the first two 3×3 convolutional layers are replaced by two 21×21 convolutional layers. Note that VGG-16 is a convolutional neural network model proposed by Simonyan and Zisserman (2015). We have done a 20-fold cross-validation to test the models ability to predict new data.



(a) Generalization error vs. c.



(b) Sparsity rate vs. c.

Figure 4: Boxplots of generalization error and sparsity rate with different c for CIFAR-10 experiment.

As shown in Figure 4b, when c increases, the network becomes denser. For example, when $c = 15$, the connection is very sparse as more than 95% of its elements are learned to be zero. However, if we increase c from 15 to 45, the sparsity rate will be reduced by almost 25%. Furthermore, Figure 4a indicates that picking a larger c might result in poorer generalization. For instance, the generalization error doubles when c is increased from 15 to 45. Such observation matches the conclusion of Theorem 3.

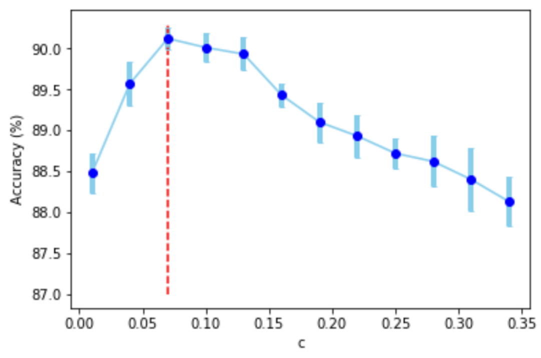
5.4 Selection of the Normalization Constant

The optimal normalization constant is chosen by k-fold cross validation (Kohavi et al., 1995; Tou and Gonzalez, 1974).

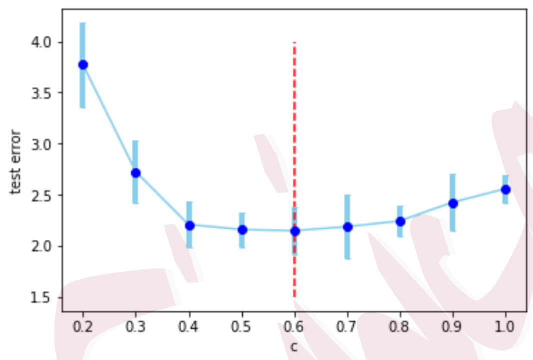
We give examples of the selection of c on the classification and regression problem in Section 5.1 and 5.2. As shown in Figure 5, we plot the average cross validation score

Regularizer	Cla. Exp	MNIST	CIFAR10
No	71.6	97.85	93.34
L_1, $\lambda = 0.1$	50.0	13.87	48.32
L_1, $\lambda = 0.01$	50.0	11.35	92.53
L_1, $\lambda = 0.001$	**81.6**	97.33	93.43
L_1, $\lambda = 0.0001$	73.2	97.99	93.64
L_2, $\lambda = 0.1$	70.8	80.30	80.41
L_2, $\lambda = 0.01$	73.4	92.30	90.38
L_2, $\lambda = 0.001$	73.2	94.43	91.41
L_2, $\lambda = 0.0001$	71.8	97.90	**93.72**
Dropout, $r_d = 0.5$	73.5	**98.11**	93.42
Our Approach	**91.0**	**98.03**	**93.75**

Table 3: Comparison of different regularization on previous classification experiment, MNIST, and CIFAR10. Bold results are the best two methods in each experiment.



(a) Accuracy on test dataset vs. c in the classification experiment.



(b) Test error vs. c in the regression experiment

Figure 5: Examples on the selection of c

against the normalization constant c for both of the experiments. Then choose the optimal c that corresponds to the largest average cross validation score.

5.5 Comparison with Other Regularizers

While establishing strong theoretical foundations for $L_{1,\infty}$-weight normalization, we show that our method performs well in practice by comparing with popular regularization techniques including L_1, L_2 (weight decay), and dropout regularizations on previous classification example, MNIST, and CIFAR-10 in terms of classification accuracy. These additional experiments are not intended to show the supreme of some well-tuned neural network architectures, but to illustrate the comparative performance of the $L_{1,\infty}$-weight normalization against other regularization methods via a fair comparison. Therefore, we only use some simple architectures for demonstration. In MNIST experiment, the input image is resized to 784×1, and then passed to a 900-10 fully connected neural network with ReLU. The loss function is cross entropy.

By using L_1 regularization with the hyperparameter λ, a penalty term $\lambda \sum \|\mathbf{W}\|_{1,1}$ is added to the original loss function, where $\mathbf{W}$ is the weight matrix. By implementing the L_2 regularization with the hyperparameter λ, a penalty term $\frac{1}{2}\lambda \sum \|\mathbf{W}\|_{2,2}^2$ is added to the original loss function. For each experiment, we compare different regularizers with various hyperparameters on the same baseline model to make a fair comparison. It is shown in Table 3 that our method is competitive with methods with other common regularizers.

6. Concluding Remarks

We have developed a systematic framework for sparse DNNs through $L_{1,\infty}$ weight normalization. We have established the Rademacher complexity of the related sparse DNN space. Based on this result, we have derived generalization error bounds for both regression and classification. The easily implemented gradient projection descent algorithm allows us to obtain a sparse DNN in practice. In experiments we have shown that the proposed $L_{1,\infty}$ minimization process leads to neural network sparsification that is competitive with current approaches while empirically validating our theoretical findings.

We have so far used a single c to control the sparsity of the network. It is interesting to extend the current framework to the network with different cs at different layers.

This poses additional challenges for computation to tune these hyperparameters. We are trying to use Bayesian optimization (Shahriari et al., 2016) to automatically select these hyperparameters. This research is currently under investigation and will be presented in another report.

Acknowledgements The research of Zhouwang Yang is supported by the NSF of China (No. 11871447), and Anhui Initiative in Quantum Information Technologies (AHY150200). The research of Xiao Wang is supported by US National Science Foundation. We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Titan Xp used for this research.

References

Bartlett, P. L. (1998). 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*(2), 525–536.

Duchi, J., S. Shalev-Shwartz, Y. Singer, and T. Chandra (2008). Efficient projections onto the l 1-ball for learning in high dimensions. In *Proceedings of the 25th international conference on Machine learning*, pp. 272–279.

Golowich, N., A. Rakhlin, and O. Shamir (2018). Size-independent sample complexity of neural networks. In *Proceedings of the 31st Conference On Learning Theory.*

Goodfellow, I., Y. Bengio, A. Courville, and Y. Bengio (2016). *Deep learning*, Volume 1. MIT press Cambridge.

Han, S., H. Mao, and W. J. Dally (2016). Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *International Conference on Learning Representations*.

Jaderberg, M., K. Simonyan, A. Zisserman, et al. (2015). Spatial transformer networks. In *Advances in neural information processing systems*, pp. 2017–2025.

Kohavi, R. et al. (1995). A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Ijcai*, Volume 14, pp. 1137–1145. Montreal, Canada.

Krizhevsky, A. (2009). Learning multiple layers of features from tiny images.

Krizhevsky, A., I. Sutskever, and G. E. Hinton (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pp. 1097–1105.

Louizos, C., K. Ullrich, and M. Welling (2017). Bayesian compression for deep learning. In *Advances in Neural Information Processing Systems*, pp. 3288–3298.

Louizos, C., M. Welling, and D. P. Kingma (2018). Learning sparse neural networks through l_0 regularization. In *International Conference on Learning Representations*.

Mohri, M., A. Rostamizadeh, and A. Talwalkar (2012). *Foundations of machine learning*. MIT press.

Molchanov, D., A. Ashukha, and D. P. Vetrov (2017). Variational dropout sparsifies deep neural networks. In *ICML*.

Neyshabur, B., R. Tomioka, and N. Srebro (2015). Norm-based capacity control in neural networks. In *Conference on Learning Theory*, pp. 1376–1401.

Nowlan, S. J. and G. E. Hinton (1992). Simplifying neural networks by soft weight-sharing. *Neural computa-*

tion 4(4), 473–493.

Salimans, T. and D. P. Kingma (2016). Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *Advances in Neural Information Processing Systems*, pp. 901–909.

Shahriari, B., K. Swersky, Z. Wang, R. P. Adams, and N. De Freitas (2016). Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE 104*(1), 148–175.

Simonyan, K. and A. Zisserman (2015). Very deep convolutional networks for large-scale image recognition. *International Conference on Learning Representations*.

Srinivas, S., A. Subramanya, and R. V. Babu (2017). Training sparse neural networks. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2017 IEEE Conference on*, pp. 455–462. IEEE.

Srivastava, N., G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research 15*, 1929–1958.

Sun, S., W. Chen, L. Wang, X. Liu, and T.-Y. Liu (2016). On the depth of deep neural networks: A theoretical view. In *AAAI*, pp. 2066–2072.

Szegedy, C., W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich (2015). Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9.

Tou, J. T. and R. C. Gonzalez (1974). Pattern recognition principles.

School of Mathematical Sciences, University of Science and Technology of China, China

E-mail: mosqwen@mail.ustc.edu.cn

Department of Statistics, Purdue University, West Lafayette, IN 47907, U.S.A.

E-mail: xu573@purdue.edu

School of Gifted Young, University of Science and Technology of China, China

E-mail: zyunling@mail.ustc.edu.cn

School of Mathematical Sciences, University of Science and Technology of China, China

E-mail: yangzw@ustc.edu.cn

Department of Statistics, Purdue University, West Lafayette, IN 47907, U.S.A.

E-mail: wangxiao@purdue.edu