Different theory techniques for shallow neural networks

Han Guo [*1] **Yikai Liu** [*1]

Abstract

We give an literature review on different theory techniques for shallow neural networks. We specifically focus on limited number of fully connected layer and convolutional layer with standard activation function and pooling. Our literature review will cover three topics: 1) how different optimization techniques affect training performance on shallow neural network, 2) how different architectures of shallow neural network differ in optimizations, 3) what assumptions/conditions are held in papers of interest. While we do not propose new methodology and analysis, our review provide some insights on different neural network settings and convergence analysis by quantitatively and qualitatively cross examining state-of-the-art work in relevant area.

1. Introduction

The stunning performance of deep learning which is backboned by neural network has made itself a promising topic in machine learning area. Deep learning performs well on variety kinds of problems including object detection and natural language processing. Neural network even outperforms human in image classification competition on ImageNet dataset, with human scored 94.9% (Russakovsky et al., 2015) and a well trained deep neural network scored 95.06% (He et al., 2015b). Despite its extraordinary empirical success, however, theoretical reasoning on why deep neural network works so well has remain relatively less well understood(Soltani & Hegde, 2019), though tremendous efforts are made trying to unveil its mystery. The significant difficulty that hinders from developing thorough and generalized theoretical aspect of deep neural network comes from highly non-convex nature of optimization posed by neural networks.(Goel et al., 2018) Empirically, it demonstrated that neural networks with more layers ("deep" learning) are essential for better performance. However, due to its non-convex optimization, it is hard to tackle with deep neural network directly; instead, focusing on shallower neural network might provide some insightful discoveries that serve as stepping stones to understand deeper and more complex models. Nevertheless, (Blum & Rivest, 1989) has proved that, without any constraints, training on shallow neural network can be NP-Complete. Thus, many works have provided convergence analysis with certain constraints to reduce the workload in reasonable sense (Brutzkus & Globerson, 2017; Jagatap & Hegde, 2018; Soltanolkotabi, 2017; Blum & Rivest, 1989; Zhang et al., 2018; Du et al., 2019; 2018a; Hardt & Ma, 2018; Du et al., 2018b; Goel et al., 2018).

The main motivation of our paper is to introduce some state-of-the-art theoretical analysis of different optimization techniques for shallow neural network. In particular, we will cover three aspects:

- How do different optimization techniques (e.g. Gradient Descent and Alternating minimization) affect the performance of shallow neural network.
- How do different neural network architectures (e.g. Conv block and fully connected layer) affect corresponding convergence analysis .
- How do conditions and assumptions (e.g. input distribution and weight initialization) differ from different theoretical works, and how do those conditions affect the convergence analysis.

The rest of this paper is structured as follow: Section 2 will provide necessary textual definitions and mathematical notations and other background information, Section 3 will focus on individual aspect by analytically cross examining variety of SOTA manuscripts, and Section 4 will proceed to discussion on current research works and future directions.

2. Background

In this section, we will briefly introduce key concepts in shallow neural network and optimization. We will then proceed to the problem setup in the context of optimization in next section. In the architecture subsection, we will provide mathematical expressions for fully connected layers, convolutional layers, residual block, and activation layers. In

[*]Equal contribution [1]Department of Computer Science, Rice University. Correspondence to: Han Guo <hg31@rice.edu>, Yikai Liu <yl163@rice.edu>.

optimization technique subsection, we will provide definitions of gradient descent and alternating minimization; more proposed techniques will be introduced in next section.

2.1. Architecture

The concept of neural network feed-forward pass is simple enough; here we provide the definitions for some variations of shallow neural networks. For the sake of simplicity, we do not include structures other than convolutional layer, fully connected layer, activation layer, input layer, and output layer.

2.1.1. Fully connected layer

Definition 2.1. (Zhang et al., 2018) A typical one-hidden-layer neural network (one input layer, one hidden layer, one output layer) has the following form:

$$y_i = \sum_{j=1}^{K} \sigma((\mathbf{w}_j^*)^\top \mathbf{z}_i) + \epsilon_i \tag{1}$$

Here, $\mathbf{w}_j^* \in \mathbb{R}^d$ is the weight parameter with respect to the j-th neuron, $\sigma(x)$ denotes activation function, $\{x_i\}_i^N \subseteq \mathbb{R}^d$ denotes input, $\{y_i\}_i^N \subseteq \mathbb{R}^d$ denotes output, and $\{\epsilon_i\}_i^N \subseteq \mathbb{R}^d$ denotes noise.

The above expression provides a general structure of shallow neural network though, variation exists. One structural variation based on equation 1 that is adopted in (Soltani & Hegde, 2019; Du et al., 2018b;c).

Definition 2.2. (Soltani & Hegde, 2019)

$$\hat{y} = \sum_{j=1}^{r} a_j \sigma(w_j^\top x) = \sum_{j=1}^{r} a_j \langle w_j, x\rangle^2 \tag{2}$$

here, the network comprises p input nodes, a single hidden layer with r neurons with activation function $\sigma(x)$, weights $\{w_j\}_{j=1}^r \subset \mathbb{R}^p$, and the single node output layer with weights $\{a_j\}_{j=1}^r \subset \mathbb{R}$.

An multilayer fully connected neural network can be generalized as a variation from section 3.3 of (Du et al., 2019) without normalization factor.

Definition 2.3. (Du et al., 2019) $x^{(h)} = \sigma\left(\mathbf{W}^{(h)}\mathbf{x}^{(h-1)}\right)$, $1 \le h \le H$

$$f(\mathbf{x}, \theta) = \mathbf{a}^\top \mathbf{x}^{(H)} \tag{3}$$

here, $\mathbf{x} \in \mathbb{R}^d$ denotes input, $\mathbf{W}^{(1)} \in \mathbb{R}^{m\times d}$ denotes the first weight matrix, $\mathbf{W}^{(h)} \in \mathbb{R}^{m\times m}$ denotes the h-th layer's weights, $\mathbf{a} \in \mathbb{R}^m$ denotes the output layer, and $\sigma(x)$ denotes the activation function.

2.1.2. Convolutional layer

The convolutional layer creates "patches", which complicates the mathematical notation.

Definition 2.4. (Goel et al., 2018) Convolutional layer with overlapping patches is computed as follows (we exclude average pooling which appeared in original equation):

$$f_w(x) = \sum_{i=1}^{k} \sigma(w^\top P_i x) \tag{4}$$

here, $x \in \mathbb{R}^n$ denotes the input, the neural network computes k patches of size r where the location of each patch is indicated by matrix $P_1, \cdots, P_k \in 0, 1^{r\times n}$ and each P_i has exactly one 1 in each row and at most one 1 in every column, $\sigma(x)$ denotes the activation function, and $w \in \mathbb{R}^r$ denotes the weight vector of convolution filter.A special case of convolutional layer arises when the patches do not overlap. This results an easier analysis which can be found in Section 3. (Du et al., 2018b) has a slightly different definition of convolutional layer, though the key concepts align with equation 4.

2.1.3. Activation layer

Activation layer provides non-linear features to the model, which increase the expressiveness (Raghu et al., 2017). One of the most widely used activation function is Rectified Linear Unit (ReLU), and it's defined as follows.

Definition 2.5. (Soltanolkotabi, 2017) ReLU preserves the positive values and set all negative values to 0s

$$\sigma(x) = \max(0, \langle \mathbf{w}, \mathbf{x}\rangle) \tag{5}$$

here, $\mathbf{x} \in \mathbb{R}^{d\times n}$ denotes input, and $\mathbf{w} \in \mathbb{R}^d$ denotes weights.
An derivation from ReLU is called Leaky ReLU, and its definition follows.

Definition 2.6. (Goel et al., 2018) Instead turning all negative values to 0s, Leaky ReLU scales those values by some factors α

$$\sigma(x) = \begin{cases} x & \text{if } x \ge 0 \\ \alpha x & \text{otherwise} \end{cases} \tag{6}$$

here, $\alpha \in [0, 1]$.

Another activation function that does not belong to ReLU family is quadratic activation.

Definition 2.7. (Soltani & Hegde, 2019) Quadratic activation is not as commonly used as ReLU,

$$\sigma(x) = x^2 \tag{7}$$

but it has shown competitive expressive power as well(Livni et al., 2014).

2.1.4. Residual connection

ResNet utilizes this structure achieves outstanding image classification scores. Residual connection enables neural networks to go deeper without worrying about vanishing gradient problem (He et al., 2015a). The key observation here is that the neural network learns the identity mapping.

Definition 2.8. (Li & Yuan, 2017) The following function contains identity mapping

$$f(x, \mathbf{W}) = \left\|\sigma((\mathbf{I} + \mathbf{W})^\top x)\right\|_1 \tag{8}$$

here, $x \in \mathbb{R}^d$ denotes input vector, $\mathbf{W} \in \mathbb{R}^{d\times d}$ denotes weights, and $\mathbf{I}$ is the identity matrix. This representation is equivalent to $\mathcal{H}(x) = \mathcal{F}(x) + x$(He et al., 2015a).

A generalized multilayer residual connection takes the following expression

Definition 2.9. (Hardt & Ma, 2018)

$$\hat{y} = (\mathbf{I} + \mathbf{A}_l)\cdots(\mathbf{I} + \mathbf{A}_1)x \tag{9}$$

where $\mathbf{A}_1 \cdots \mathbf{A}_l \in \mathbb{R}^{d\times d}$ denotes weights, and $\mathbf{I}$ is the identity matrix.

2.2. Optimization

2.2.1. Gradient Descent

Gradient descent perhaps is the most commonly used optimization technique in deep learning. The following describes gradient decent optimization.

Definition 2.10. (Soltanolkotabi, 2017) The following expression is a variation from original projected gradient descent

$$\mathbf{w}_{\tau+1} = \mathbf{w}_\tau - \eta\nabla\mathcal{L}(\mathbf{w}_\tau) \tag{10}$$

here, $\mathbf{w}_{\tau+1}$ denotes $\tau + 1$-th weights, $\mathbf{w}_\tau$ denotes τ-th weights, η denotes stepsize, and $\nabla\mathcal{L}(w_\tau)$ denotes loss function which will be introduced in Subsection 2.3. Other problem-specific modification on gradient descent concept will be specified in Section 3.

2.2.2. Alternating minimization

Another less common optimization technique is alternating minimization. The core concept behind this technique is to always treat one unknown as variable and alternating the process of choosing unknown and minimizes it. This technique is problem-specific which will be introduced in detail in Section 3.

2.3. Loss function

In deep learning optimization, a function used to evaluate a candidate solution is referred as loss function or objective solution. Canonically, minimization operation is performed on a loss function, meaning that we are searching for a candidate solution that has the highest score (the score is calculated by comparing the predicted value and ground-truth label). Section 2.3.1 introduces empirical risk minimization, and Section 2.3.2 introduces population loss minimization.

2.3.1. Empirical risk minimization

In practice, we do not have access to the true distribution of data that we are working on; however, we have access to some amount of data, and we are trying to approximate the loss of entire population with the limited access of sampling data. The following defines a typical empirical loss of a simple feedforward function with only weights and activation.

Definition 2.11. (Soltanolkotabi, 2017) The least-squares empirical loss is a variation from the original expression in (Soltanolkotabi, 2017)

$$\min_{\mathbf{w}\in\mathbb{R}^d} \mathcal{L}(\mathbf{w}) := \frac{1}{2}(f(\mathbf{w}, \mathbf{x}) - y)^2 \tag{11}$$

here, $f(\mathbf{w}, \mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n}\sigma(\mathbf{w}^\top\mathbf{x}_i)$ denotes a feedforward function with weights $\mathbf{w}$, input data $\mathbf{x}_i$, activation function $\sigma(x)$, and y is ground truth label.

2.3.2. Population loss

In an ideal case, if we have access to true distribution of data that we are working on, we can assume a ground-truth global minimum weight $\mathbf{w}^*$. This assumption simplifies some of the convergence analysis. The following defines a typical population loss of a simple feedforward function with only weights and activation.

Definition 2.12. (Du et al., 2018a) A least-squares population loss is a variation from the original expression in (Du et al., 2018a)

$$\min_{\mathbf{w}\in\mathbb{R}^d} \mathcal{L}(\mathbf{w}, \mathbf{x}) := \frac{1}{2}(f(\mathbf{w}, \mathbf{x}) - f(\mathbf{w}^*, \mathbf{x}))^2 \tag{12}$$

here, $f(\mathbf{w}, \mathbf{x})$ follows the same definition as in Definition 2.11, and $f(\mathbf{w}^*, \mathbf{x})$ in equation 12 has ground-truth weights w^*.

Section 2.3.1 and 2.3.2 denotes empirical loss and population loss in general form; however, depending on the problem setup, some adaptations might be necessary. Detailed modification will be noted in Section 3 as needed.

So far, we have provided some basic notations that will be utilized in next section. We will then proceed to our analysis on three main aspects.

3. Methodology

In this section, we are trying to answer the questions that we have brought up in Section 1 by quantitatively and qualitatively reviewing relevant literature works on optimization with different settings. In Section 3.1, we will discuss how Gradient Descent and other optimization techniques. In Section 3.2, we will analyze how different neural network architectures result in different convergence analysis. In Section 3.3, we will discuss on some common assumptions and conditions that either loose or constrain the applicability and generalizability of convergence analysis, and how does those conditions help to develop theoretical bounds/constraints.

3.1. Different optimization techniques

3.1.1. Gradient-based optimization

In Section 2.2.1, we have briefly introduced the basic concept of gradient descent in which weights are updated in the direction where the loss is maximally minimized. A line of research focusing on the behavior of gradient-based algorithm has shown its power. (Tian, 2017b) use a variant of gradient descent algorithm – population gradient descent where instead of empirical loss, population loss is considered. The population gradient takes the following form.

Definition 3.1. (Tian, 2017b)If we assume population loss, then population gradient $\mathbb{E}_X[\nabla J_w(\mathbf{w})]$ with population loss $J(\mathbf{w}) = \frac{1}{2}\|g(X;\mathbf{w}^* - g(X;\mathbf{w}))\|^2$ where $g(X;\mathbf{w}) = \sum_{j=1}^{K}\sigma(\mathbf{w}_j^\top x)$ with respect to weight $\mathbf{w}_j$ has the expression

$$\mathbb{E}_X[\nabla_{\mathbf{w}_j} J] = \sum_{j'=1}^{K}\mathbb{E}[F(e_j,\mathbf{w}_{j'})] - \sum_{j'=1}^{K}\mathbb{E}[F(e_j,\mathbf{w}_{j'}^*)] \tag{13}$$

here, $e_j = \mathbf{w}_j/\|\mathbf{w}_j\|$. Then simply substituting $\nabla\mathcal{L}(\mathbf{w}_\tau)$ in equation 10 with equation 13, we will have the final population gradient descent

$$\mathbf{w}_{\tau+1} = \mathbf{w}_\tau - \eta\nabla\mathbb{E}_X[\nabla_{\mathbf{w}} J] \tag{14}$$

Under the conditions of spherical Gaussian input and randomized weight initialization, (Tian, 2017b) has proved that one-layer one neuron model is able to recover true weight vector.

(Soltanolkotabi, 2017) improves this result by using empirical loss that looses the constraint. Particularly,

$$n_0 = \mathcal{M}((R),\mathbf{w}^*) = \omega^2(\mathcal{C}_R(\mathbf{w}^*)\cap\mathcal{B}^d) \tag{15}$$

defines n_0 to be the exact minimum number of samples required, where $\mathcal{C}_R(\mathbf{w}^*)$ is a cone descent of a regularizer function $\mathcal{R}$ at $\mathbf{w}^*$ and $\mathcal{B}^d$ denotes unit ball of $\mathbb{R}^d$, then the empirical projected gradient descent can be defined as follow.

Definition 3.2. (Soltanolkotabi, 2017) Let $\nabla\mathcal{L}(\mathbf{w}_\tau)$ be empirical loss function, then empirical projected gradient descent is

$$\mathbf{w}_{\tau+1} = \mathcal{P}_k(\mathbf{w}_\tau - \eta\nabla\mathcal{L}(\mathbf{w}_\tau)) \tag{16}$$

here, η denotes the step size and $\mathcal{K} = \{\mathbf{w}\in\mathbb{R}^d : \mathcal{R}(\mathbf{w}\le\mathcal{R})\}$ is the constraint set with $\mathcal{P}_\mathcal{K}$ denoting the Euclidean projection onto this set. The key theorem here is that if equation 15 is satisfied, then equation 16 obey $\|\mathbf{w}_\tau - \mathbf{w}^*\|_F \le \left(\frac{1}{2}\right)^\tau\|\mathbf{w}^*\|_F$. This theorem shows that with near minimal number of data sample n_0, projected gradient descent learns ground-truth weight with linear convergence rate. This result also applies to both convex and nonconvex regularization functions, and it shows that with near minimal number of data samples, project gradient descent converges without getting trapped in bad local optima.

3.1.2. Alternating minimization optimization

Alternating minimization is another optimization technique proposed by (Jagatap & Hegde, 2018). On the high level, the idea of alternating minimization is to estimate the activation patterns of each ReLU for all given samples and interleave with weight updates via a least-squares loss.

Specifically, they linearize all samples by defining *state* of the neural network as the collection of binary variables that indicates whether ReLU is active or not and fixing that state. This idea is inspired by the feature of ReLU that positive values of ReLU will remain their weights and negative values will be clipped to 0, so we can separate the value of weights to an indicator matrix and values of weights. This process can be linearized. Let

$$B = [\text{diag}(p_1)X\ldots\text{diag}(p_k)X]_{n\times dk} \tag{17}$$

be linearized state of ReLU where $p_i = \mathbb{1}_{\{Xw_i>0\}}$ denotes the indicator function for sign of weights. Then

Definition 3.3. (Jagatap & Hegde, 2018)Feedforward function can be expressed as

$$f(X) = \sum_{i=1}^{k}\text{ReLU}(Xw_i) = B\cdot\text{vec}(W) \tag{18}$$

here, $\text{vec}(W)$ vectorize weight W. And the minimization update can be thus described as

$$\text{vec}(W)^{t+1} = \underset{\text{vec}(W)}{\arg\min}\left\|B^t\cdot\text{vec}(W) - y\right\|_2^2 \tag{19}$$

Thus, by alternating equation 17 and equation 19 , alternating minimization converges to global minimum with linear convergence rate if the initial weight W^0 satisfying $\text{dist}(W^0,W^*)\le\delta_1\|W^*\|_F$ for $0<\delta_1<1$, where $\text{dist}(W,W')$ is defined as

$$\text{dist}(W,W') = \min_{\text{all possible of column perturbations}}\|W - W'\|_F$$

Algorithm 1 Alternating Minimization

```
Require: X, y, T, k
  Initialize $W^0$ s.t. $\text{dist}(W^0, W^*) \le \delta_1 \|W^*\|_F$
  for $t = 0, \cdots, T-1$ do
    $p_q^t = \mathbb{1}_{\{Xw_q^t > 0\}}, \forall q \in \{1...k\}$
    $B^t = [\text{diag}(p_1^t)X ... \text{diag}(p_k^t)X]_{n \times dk}$
    $\text{vec}(W)^{t+1} = \underset{\text{vec}(W)}{\arg\min} \left\|B^t \cdot \text{vec}(W) - y\right\|_2^2$
    $W^{t+1} \leftarrow \text{reshape}(\text{vec}(W)^{t+1}, [d, k])$
  end for
  Return $W^T \leftarrow W^t$
```

. This weight initialization can be obtained by set $W^0 \leftarrow \mathbf{I}$. The detailed implementation is shown in Algorithm 1.

3.1.3. Low rank matrix estimation

Though theoretical aspect of neural network is not well understood, there are areas that we do have enough theoretical findings. One natural idea is to bridge the problem of learning (shallow) neural network with a well understood problem of low-rank matrix estimation. Specifically, the problem of learning a shallow neural network can be treated as a low-rank matrix estimation problem where the rank of the resulting matrix equals to the number of hidden neurons. The following definition provides a problem setup the network of our interest.

Definition 3.4. (Soltani & Hegde, 2019) The network of our interest consists p input nodes, a single hidden layer with r neurons with quadratic activation function $\sigma(z) = z^2$, first layer weights $\{w_j\}_{j=1}^r \subset \mathbb{R}^p$, and an output layer comprising of a single node and weights $\{a_j\}_{j=1}^r \subset \mathbb{R}$, then the input-output relation can be expressed as the following

$$\hat{y} = \sum_{j=1}^{r} a_j \sigma(w_j^T x) = \sum_{j=1}^{r} a_j \langle w_j^T x \rangle^2 \tag{20}$$

Consider set of training input-output pairs $\{(x_i, y_i)\}_{i=1}^m$ and set of weights $\{(a_j, w_j)\}_{j=1}^r$. We define matrix variable $L_* = \sum_{j=1}^r a_j w_j w^T$, then input-output relation becomes

$$\hat{y}_i = x_i^T L_* x_i = \langle x_i x_i^T, L_* \rangle \tag{21}$$

here $x_i \in \mathbb{R}^p$ denotes the i_{th} training sample, L_* is a rank-r matrix of size $p \times p$, so empirical loss function

$$\min_{W \in \mathbb{R}^{r \times p}, a \in \mathbb{R}^r} F(W, a) = \frac{1}{2m} \sum_{i=1}^{m} (y_i - \hat{y}_i)^2 \tag{22}$$

can be viewed as an instance of learning a fixed rank-r symmetric matrix $L_* \in \mathbb{R}^{p \times p}$ where $r \ll p$ from small number of *rank-one* linear observation given by $A_i = x_i x_i^T$.

A few algorithms are proposed to estimate L_*,given $\{x_i, y_i\}_{i=1}^m$. The first method, called Exact Projections for Rank-One Matrix, or EP-ROM, which solves the non-convex, constrained risk minimization problem:

$$\min_{L \in \mathbb{R}^{p \times p}} F(L) = \frac{1}{2m} \sum_{i=1}^{m} (y_i - x_i^T L x_i)^2 \tag{23}$$

is demonstrated in Algorithm 2.

Algorithm 2 EP-ROM

```
  Inputs: y, number of iterations K, independent data samples $\{x_1^T ..., x_m^T\}$ for t = 1,....,K,rank r
  Outputs: Estimates $\hat{L}$
  Initialization: $L_0 \leftarrow 0, t \leftarrow 0$
  Calculate: $\hat{y} = \frac{1}{m} \sum_{i=1}^m y_i$
  while $t \le K$ do
    $L_{t+1} = P_r(L_t - \frac{1}{2m} \sum_{i=1}^m ((x_i^t)^T L_t x_i^t - y_i) x_i^t (x_i^t)^T - (\frac{1}{2m} 1^T A(L_t) - \frac{1}{2}\hat{y}) I)$,
    $t \leftarrow t + 1$
  end while
  Return $\hat{L} = L_k$
```

While EP-ROM exhibits linear convergence, the per-iteration complexity is still high since it requires projection onto the space of rank-r matrices, which necessitates the application of SVD. The total running time of EP-ROM is $O(mp^2 log(\frac{1}{\epsilon}))$Thus, a second algorithm, called Approximate Projection for Rank One Matrix estimation, or AP-ROM, is proposed, as shown in Algorithm 3.

Algorithm 3 AP-ROM

```
  Inputs: y, number of iterations K, independent data samples $\{x_1^T ..., x_m^T\}$ for t = 1,....,K,rank r
  Outputs: Estimates $\hat{L}$
  Initialization: $L_0 \leftarrow 0, t \leftarrow 0$
  Calculate: $\hat{y} = \frac{1}{m} \sum_{i=1}^m y_i$
  while $t \le K$ do
    $L_{t+1} = \tau(L_t - H(\frac{1}{2m} \sum_{i=1}^m (((x_i^t)^T l_t x_i^t - y_i) x_i^t (x_i^t)^T - (\frac{1}{2n} 1^T A(L_t) - \frac{1}{2}\hat{y}) I))$,
    $t \leftarrow t + 1$
  end while
  Return $\hat{L} = L_k$
```

The specific choice of approximate SVD algorithm that simulates the operators τ(.) and H(.) is flexible. AP-ROM also demonstrates linear convergence as EP-ROM. However, AP-ROM demonstrates a better running time of $O(mprlog(p)log(\frac{1}{\epsilon}))$

Note that without any assumptions on spectral norm, estimating L_* takes $\mathcal{O}(p^3 r^2)$ running time complexity, due to the calculation of SVD. However, this can be improved by replacing standard SVD with approximate heuristics such as randomized Block Krylov SVD to $\mathcal{O}(p^2 r^4 \log^2(\frac{1}{\epsilon}) \text{polylog}(p))$.

3.2. Different neural network architectures

Mathematical formulation of the problem setup depends strictly on the architectural design of neural network. For shallow neural network, most common design patterns are input layer, hidden layer, and output layer. While input and output layer are canonically invariant across different network designs, hidden layers take much versatile forms. Modern neural network architecture design mainly focus on better hidden layer construction. Here we convey three common architecture practices.

3.2.1. Fully Connected Architecture

Fully connected layer (FC) forms fundamental connection between neurons in adjacent layers. While we have briefly provide some definitions in Section 2.1.1, we have yet provide a interpretation of FC in general. The following definition defines fully connected layer in linear algebraic aspect.

Definition 3.5. Assuming $\sigma(\cdot)$ is activation operator (if assume $\sigma(x)$ to be ReLU, then $\sigma(x) = \max(0, x)$). Let $x \in \mathbb{R}^m$ and $y_i \in \mathbb{R}$ be the i-th output, then

$$y_i = \sigma(w_1 x_1 + ... w_m x_m) \tag{24}$$

holds. Full output y is then

$$y = \begin{bmatrix} \sigma(w_{1,1}x_1 + ...w_{1,m}x_m) \\ \vdots \\ \sigma(w_{n,1}x_1 + ...w_{n,m}x_m) \end{bmatrix} \tag{25}$$

Note that since the concept of FC involves summation over all weights, in the convergence analysis, we can easily rearrange summation operator in our favor. In (Zhang et al., 2018), where the original population loss function

$$\mathcal{L}(\mathbf{W}) = \frac{1}{2}\mathbb{E}_{\mathbf{X}\sim\mathcal{D}_{\mathbf{X}}}\left(\sum_{j=1}^{K}\sigma(\mathbf{w}_j^\top\mathbf{X}) - \sum_{j=1}^{K}\sigma(\mathbf{w}_j^{*\top}\mathbf{W})\right)^2$$

its partial derivative takes form

$$\left[\nabla\hat{\mathcal{L}}_N(\mathbf{W})\right] = \sum_{j=1}^{K}(\hat{\mathbf{\Sigma}}(\mathbf{w}_j, \mathbf{w}_k)\mathbf{w}_j - \hat{\mathbf{\Sigma}}(\mathbf{w}_j^*, \mathbf{w}_k)\mathbf{w}_j^*) - \frac{1}{N}\sum_{i=1}^{N}\epsilon_i\mathbf{x}_i\cdot\mathbb{1}\{\mathbf{w}_k^\top\mathbf{x}_i \geq 0\}$$

Where we can see that the summation is taken out as a stand alone factor. (Zhang et al., 2018) also shows that with this partial derivative as gradient update and initial weight $\mathbf{W}^0$ satisfies $\left\|\mathbf{W}^0 - \mathbf{W}^*\right\|_F \leq c\sigma_K/(\lambda\kappa^3K^2)$, gradient descent converges to ground truth $\mathbf{W}^*$ in linear time. The weight initialization requirement can be achieved by tensor initialization that is discussed in Section 3.3.2.

3.2.2. Convolutional Filter Architecture

The defining characteristic of Convolutional Neural Network (CNN) is its convolutional layer. Unlike fully connected layer, convolutional layer relies on convolution filter, or kernal, to extract features from original input. The mathematical expression for convolutional shallow neural network is described in Section 2.4. Based on the dimension of input, we consider two general types of convolution – 1D Convolution and 2D Convolution.

Definition 3.6. (Goel et al., 2018) Consider a 1D image of dimension n. Let the patch size be r and stride be d. Let the patches be indexed from 1 and let patch i start at position $(i-1)d+1$ and be contiguous through position $(i-1)d+r$. The matrix P_i of dimension $r \times n$ of patch i looks as follows,

$$P_i = (0_{r\times((i-1)d+1)}I_r 0_{r\times(n-r-(i-1)d)}) \tag{26}$$

here $0_{a\times b}$ indicates all zero matrix of size $a \times b$, and I_r indicates identity matrix of size r. Let $k = \lfloor\frac{n-r}{d}\rfloor + 1$ The structure of P is summarized as below.

$$P_{i,j} = \begin{cases} k-a & \text{if } |i-j| = ad \\ 0 & \text{otherwise} \end{cases} \tag{27}$$

We bound extremal eigenvalue $P = \sum_{i,j=1}^{k} P_i P_j^\top$.

Definition 3.7. (Goel et al., 2018) Consider a 2D image of dimension $n_1 \times n_2$. Let the patch size be $r_1 \times r_2$ and the stride in both directions be d_1, d_2 respectively. Enumerate patches such that patch (i, j) starts at position $((i-1)d_1 + 1, (i-1)d_2 + 1)$ and is a rectangle with diagonally opposite point $((i-1)d_2+r_1, (j-1)d_2+r_2)$. Let $k_1 = \lfloor\frac{n_1-r_1}{d_1}\rfloor+1$ and $k_2 = \lfloor\frac{n_2-r_2}{d_2}\rfloor + 1$. Let $Q_{(i,j)}$ be indicator matrix of dimension $r_1r_2 \times n_1n_2$ with 1 at (a, b) if ath location of patch (i, j) is b. Formally,

$$(Q_{(i,j)})_{a,b} = 1 \tag{28}$$

for all $a = pr_2 + q + 1$ for $0 \leq p < r_1$, $0 \leq q < r_2$, and $b = ((i-1)d_1 + p)n_2 + jd_2 + q + 1$ else 0. The extremal eigenvalue is bounded by $Q = \sum_{i,p=1}^{k_1}\sum_{j,q=1}^{k_2} Q_{(i,j)}Q_{(p,q)}^\top$

Notice that the above 1D Convolution and 2D Convolution make no assumption on whether patches overlap or not. In fact, if there is one patch P_1 that does not overlap with any other patches, the convergence analysis simplifies significantly because the term $P_qP_j^\top = P_j^\top P_q = 0$ for all $P_j \neq 1$; in particular, the resulting expectation of loss in non-overlapping case eliminates the bounding eigenvalue terms comparing to overlapping case due to the orthogonality exhibited by $P_qP_j^\top = P_j^\top P_q = 0$.

Another work by (Du et al., 2018a) considers patches with ”close relations”. Specifically, they show if the input patches are highly correlated $\theta(Z_i, Z_j) \leq \rho$ for some small $\rho > 0$, then gradient descent with random initialization recovers the filter in polynomial time, and the stronger the correlation, the faster the convergence rate. The high level approach is to first divide input patches into 4 events, find average patch in each event, and find max and min eigenvalues respectively. Assume

$$\begin{aligned}\max_{w:\theta(w,w_*)\leq\phi} &\lambda_{max}(\mathbb{E}[Z_{S_{w,w_*}} Z_{S_{w,-w_*}}^\top]) \\ &+ \lambda_{max}(\mathbb{E}[Z_{S_{w,w_*}} Z_{S_{-w,w_*}}^\top]) \\ &+ \lambda_{max}(\mathbb{E}[Z_{S_{w,-w_*}} Z_{S_{-w,w_*}}^\top]) \leq L_{cross}\end{aligned}$$

here $Z_{S_{w,w_*}}, Z_{S_{w,-w_*}}, Z_{S_{-w,w_*}}$ are patch average of four events mentioned previously. The, if patch Z_i and Z_j are very similar, then joint probability density of $Z_i \in S(w, w_*)_i$ and $Z_j \in S(w, -w_*)_j$ is small and implies L_{cross} is small. If L_{cross} is small, then by

$$\|w_{t+1} - w_*\|_2^2 \leq \left(1 - \frac{\eta(\gamma(\phi_t) - 6L_{cross})}{2}\right) \|w_t - w_*\|_2^2 \tag{29}$$

we have faster convergence rate in polynomial time.

3.2.3. Residual Connection Architecture

In Section 2.1.4, we have briefly introduce the mathematical notations for residual connection network. In this section we will convey some relevant analysis that leverages this structure.

Definition 3.8. (Li & Yuan, 2017)If the loss function takes form

$$\begin{aligned}\mathcal{L}(\mathbf{W}) = \mathbb{E}_x[(&\sum_i \text{ReLU}(\langle w_i + w_i, x\rangle) \\ &- \sum_i \text{ReLU}(\langle e_i + w_i^*, x\rangle))^2]\end{aligned} \tag{30}$$

there exists $\gamma > \gamma_0 > 0$ such that if $x \sim \mathcal{N}(0, \text{I})$, $\|\mathbf{W}_0\|_2, \|\mathbf{W}^*\|_2 \leq \gamma_0, d \geq 100, \epsilon \leq \gamma^2$, then stochastic gradient descent on $\mathcal{L}(\text{W})$ will find W^* by two phases:

- Phase I, setting step size $\eta \leq \frac{\gamma^2}{G_2^2}$, potential function $g = \sum_{i=1}^d (\|e_i + w_i^*\|_2 - \|e_i + w_i\|_2)$ takes at most $\frac{1}{16\eta}$ steps to decrease to smaller than $197\gamma^2$
- Phase II, for $a > 0$ and $\forall T$ s.t. $T^a \log T \geq \frac{36d}{100^4(1+a)G_F^2}$, if $\eta = \frac{(1+a)\log T}{\delta T}$, then $\mathbb{E}\|\mathbf{W}_T - \mathbf{W}^*\|_F^2 \leq \frac{(1+a)\log TG^2}{\delta^2 T}$

The key observation here is that the residual network converges to global minimum in two phases. In Phase I, the potential function g is decreasing to a small value, and in Phase II, g remains small, so $\mathcal{L}$ is one point convex and $\mathbf{W}$ starts to converge to $\mathbf{W}^*$. The proof of Phase I is fairly simple, by introducing an auxiliary variable $s = (\mathbf{W}^* - \mathbf{W})u$; the proof of Phase II leverages Taylor expansion an controls higher order terms.

Another work done by (Hardt & Ma, 2018) gives simple proof that arbitrarily deep linear residual networks have no spurious local optima. Specifically, they suggest that it is sufficient for the optimizer to converge to critical points of the population risk since all critical points are global minima. If $\mathcal{B}_\tau = \{A \in \mathbb{R}^{l\times d\times d} : |||A||| \leq \tau\}$ where $|||A||| := \max_{1\leq i\leq j} \|A_i\|$, then

$$\|\nabla f(A)\|_F^2 \geq 4l(1-\tau)^{2l-2}\sigma_{min}(\Sigma)(f(A) - C_{opt}) \tag{31}$$

equation 31 says the gradient has fairly large norm compared to the error, which guarantees convergence if the gradient descent to a global minimum if the iterates stay inside $\mathcal{B}_\tau$. Here C_{opt} is global minimum of f, and $\|\nabla f(A)\|_F^2$ denotes Euclidean norm of $\nabla f(A)$

3.3. Conditions and assumptions

It is known that without any assumptions on the problem, the learning in neural network is NP-hard. To reduce the difficulty of convergence analysis to a more reasonable workload, most of the works we focus on have made certain assumptions on the problem setup. There are two particular conditions that prevail in current works – input distribution and weight initialization. We will discuss about different input distribution in Section 3.3.1 and different weight initialization techniques in Section 3.3.2.

3.3.1. Input distribution

Neural network deals with enormous amount of data. Since most optimization algorithms are not shift invariant, a good input data distribution is critical to obtain good results and to reduce significantly calculation time(Sola & Sevilla, 1997). Through an extensive survey, though not exhaustive, we discover that one of the most commonly used input assumptions is i.i.d Gaussian distribution.

Definition 3.9. (Li & Yuan, 2017) We say input data is in standard i.i.d Gaussian distribution, if the input vector $x \in \mathbb{R}^d$ is sampled from normal distribution $\mathcal{N}(0, I)$.

In (Li & Yuan, 2017), here the original derivative of loss function is defined as

$$\begin{aligned}\nabla L(\mathbf{W})_j = 2\mathbb{E}_x[(&\sum_i \text{ReLU}(\langle e_i + w_i, x\rangle) \\ &- \sum_i \text{ReLU}(\langle e_i + w_i^*, x\rangle))x\mathbb{1}_{\langle e_i+w_j,x\rangle\geq 0}]\end{aligned}$$

This indicates that the original original loss function is not well defined everywhere, and analysis is only valid for each case. However, with the assumption of input is from Gaussian distribution and some modifications of equation 13 from (Tian, 2017a), the derivative of loss function can be rewritten as

$$-\nabla L(\mathbf{W})_j = \sum_{i=1}^{d}\Big(\frac{\pi}{2}(w_i^* - w_i) + (\frac{\pi}{2} - \theta_{i^*,j})(e_i + w_i^*) - (\frac{\pi}{2} - \theta_{i,j})(e_i + w_i) + (\|e_i + w_i^*\|_2 \sin\theta_{i^*,j} - \|e_i + w_i\|_2 \sin\theta_{i,j})\overline{e_j + w_j}\Big)$$

One key observation here is that, if we assume the condition that the input x is from the standard Gaussian distribution, the loss function is smooth and the gradient is well defined every where. This idea conforms with (Tian, 2017a) in a sense that the gradient is treated as a random variable that can be expressed in terms of the expectation.

The idea that having input data distribution simplifies and improves convergence analysis is further supported by (Brutzkus & Globerson, 2017) in *No-Overlap Networks*. In particular, they derive the problem *No-Overlap Networks* from set splitting problem and argues that its complexity is NP-complete; however, if input $x \sim \mathcal{N}(0, 1)$, *No-Overlap Networks* is upper bounded by polynomial factors. This claim comes from the observation that the if input condition is satisfied, gradient descent will stay away from the degenerate saddle point. This claim highlights the importance of input distribution being Gaussian distribution in a sense of asymptotic bounds in convergence analysis. Empirical experiments conducted by (Brutzkus & Globerson, 2017) confirms that Gaussian input trial converges to global minimum while non-Gaussian input trial gets trapped in bad local minimum.

3.3.2. Weight initialization

Weight initialization is crucial in practice, yet we have primitive understanding on this subject(Goodfellow et al., 2016). A natural idea is to have all weights initialized to be 0s. However, this is very unfavorable because zero weights initialization causes **symmetry problem** – all hidden units are symmetric and different layers don't learn different features. Therefore, to break this symmetry, weight initialization technique is necessary. There are several popular choices of weight initialization. One common practice is random weight initialization. This technique is fairly simple – randomly initialize weight parameters with standard Gaussian distribution. (Du et al., 2018b) utilizes random weight initialization and proves that gradient descent learns one-hidden-layer convolutional neural network with non-overlapping patches in polynomial time.

Another weight initialization technique is to randomly initialize weights with $O(1/\sqrt{d})$. This technique is commonly known as "Xavier initialization"(Glorot & Bengio, 2010) or "He initialization"(He et al., 2015a). The difference between "He initialization" and "Xavier initialization" is trivial, and we will not expand more on this topic due to the scope of our paper. (Li & Yuan, 2017) utilizes this initialization technique and leverages the fact that spectral norm of random matrix is $O(1)$. This result justifies $\|W^*\|_2 = O(1)$.

Algorithm 4 Tensor Initialization

```
procedure INITIALIZATION(set $S$)
    $S_2, S_3, S_4 \leftarrow$ PARTITION($S, 3$)
    $\hat{P}_2 \leftarrow \mathbb{E}_{S_2}[P_2]$
    $V \leftarrow$ POWERMETHOD($\hat{P}_2, k$)
    $\hat{R}_3 \leftarrow \mathbb{E}_{S_3}[P_3(V, V, V)]$
    $\{\hat{u}\}_i i \in [k] \leftarrow$ KCL($\hat{R}_3$)
    $\{\hat{u}_i\}_{i\in[k]} \leftarrow$ RECMAGSIGN($V, \{\hat{u}_i\}_{i\in[k]}, S_4$)
    Return $\{w_i^{(T)}, v_i^{(0)}\}_{i\in[k]}$
end procedure
```

Tensor Initialization proposed by (Zhong et al., 2017) is fairly uncommon yet intriguing. Tensor initialization is a derivation from tensor problem. Although in general tensor problems are NP-hard(Hillar & Lim, 2013), by assuming noiseless and Gaussian input conditions, the authors are able to develop an efficient tensor method of weight initialization algorithm described in Algorithm 4. The core idea of tensor initialization is to leverage tensor decomposition and tensor estimation after dimension reduction. By applying tensor initialization, (Zhang et al., 2018) are able to prove that initial weight W^0 falls into small neighborhood of ground-truth weight W^*, thus leading to the proof of linear convergence rate.

4. Discussion

We have provided rather brief introduction of different theory techniques for shallow neural networks. We have touched upon several commonly used architecture, e.g. residual connection and covolutional filters, and optimization techniques including gradient-based algorithm and alternating minimization. Though not with too much proof detail, we have deliver problem setup and analysis conceptually. Since our purpose is to inform and convey a bigger scope of optimization in shallow neural networks in general, our review does cover essential topic that are crucial to understand.

However, we do notice that our literature review does not cover all topics in shallow neural networks. For example, we only consider simplified network where only input, hidden layer, and output present – we do not consider BN and pooling. We also noticed that most papers assume this simplified version of network architecture as we do. So one possible future direction would be a systematic and holistic review on how BN/pooling/Dropout affect optimization and convergence analysis in shallow neural network. Also, we do not go into detail on theoretical definition of $O/\sqrt{d}$ weight initialization which is largely embraced by the community. Specifically how "Xavier initialization" and "He initialization" differ in different network setting (for example, empirically, He initialization works better on residual connection). This input assumption based analysis is also interesting for further investigation.

References

Blum, A. and Rivest, R. L. Training a 3-node neural network is np-complete, 1989. URL https://papers.nips.cc/paper/125-training-a-3-node-neural-network-is-np-complete.pdf.

Brutzkus, A. and Globerson, A. Globally optimal gradient descent for a convnet with gaussian inputs, 2017.

Du, S. S., Lee, J. D., and Tian, Y. When is a convolutional filter easy to learn?, 2018a.

Du, S. S., Lee, J. D., Tian, Y., Poczos, B., and Singh, A. Gradient descent learns one-hidden-layer cnn: Don't be afraid of spurious local minima, 2018b.

Du, S. S., Zhai, X., Póczos, B., and Singh, A. Gradient descent provably optimizes over-parameterized neural networks. *CoRR*, abs/1810.02054, 2018c. URL http://arxiv.org/abs/1810.02054.

Du, S. S., Lee, J. D., Li, H., Wang, L., and Zhai, X. Gradient descent finds global minima of deep neural networks, 2019.

Glorot, X. and Bengio, Y. Understanding the difficulty of training deep feedforward neural networks. volume 9 of *Proceedings of Machine Learning Research*, pp. 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. JMLR Workshop and Conference Proceedings. URL http://proceedings.mlr.press/v9/glorot10a.html.

Goel, S., Klivans, A., and Meka, R. Learning one convolutional layer with overlapping patches, 2018.

Goodfellow, I., Bengio, Y., and Courville, A. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

Hardt, M. and Ma, T. Identity matters in deep learning, 2018.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition, 2015a.

He, K., Zhang, X., Ren, S., and Sun, J. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, 2015b.

Hillar, C. J. and Lim, L.-H. Most tensor problems are np-hard. 60(6), 2013. ISSN 0004-5411. doi: 10.1145/2512329. URL https://doi.org/10.1145/2512329.

Jagatap, G. and Hegde, C. Learning relu networks via alternating minimization, 2018.

Li, Y. and Yuan, Y. Convergence analysis of two-layer neural networks with relu activation, 2017.

Livni, R., Shalev-Shwartz, S., and Shamir, O. On the computational efficiency of training neural networks, 2014.

Raghu, M., Poole, B., Kleinberg, J., Ganguli, S., and Sohl-Dickstein, J. On the expressive power of deep neural networks, 2017.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. Imagenet large scale visual recognition challenge, 2015.

Sola, J. and Sevilla, J. Importance of input data normalization for the application of neural networks to complex industrial problems. *IEEE Transactions on Nuclear Science*, 44(3):1464–1468, 1997. doi: 10.1109/23.589532.

Soltani, M. and Hegde, C. Fast and provable algorithms for learning two-layer polynomial neural networks. *IEEE Transactions on Signal Processing*, 67(13):3361–3371, 2019. doi: 10.1109/TSP.2019.2916743.

Soltanolkotabi, M. Learning relus via gradient descent, 2017.

Tian, Y. Symmetry-breaking convergence analysis of certain two-layered neural networks with relu nonlinearity. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Workshop Track Proceedings*. OpenReview.net, 2017a. URL https://openreview.net/forum?id=r1lVgRNtx.

Tian, Y. An analytical formula of population gradient for two-layered relu network and its applications in convergence and critical point analysis, 2017b.

Zhang, X., Yu, Y., Wang, L., and Gu, Q. Learning one-hidden-layer relu networks via gradient descent, 2018.

Zhong, K., Song, Z., Jain, P., Bartlett, P. L., and Dhillon, I. S. Recovery guarantees for one-hidden-layer neural networks, 2017.