Training Neural Networks without Backpropagation

Shucong Zhang

Master of Science
School of Informatics
University of Edinburgh
2017

Abstract

Gradient-based optimizers are widely used in training neural networks. However, since the optimization problem is non-convex, gradient-based methods, such as backpropagation, are often stuck in local optima. Current literature argues training a shallow neural network with scalar output via orthogonal tensor decomposition is guaranteed to converge to the global optimum. We analyze this novel approach and prove that it is only suitable for training a particular family of neural networks: the neural networks must meet several assumptions. We implement and test this new training method on the synthetic data set. Our experiment results coincide the statement of guaranteed training. We also purpose initializing backpropagation via the alternative least squares tensor decomposition. Although our method does not guarantee global optimum, it only makes one assumption about the neural network. On the synthetic data, our method beats backpropagation with random initialization significantly.

Acknowledgements

Many thanks to my supervisor Dr. Shay Cohen for his kindly help and guidance. Many thanks for my family and friends who support me during my entire MSc study.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Shucong Zhang*)

Table of Contents

Chapter 1

Introduction

1.1 Motivation

Neural networks are extremely successful models. There are varieties of different neural network models, such as convolutional neural networks [KSH12], recurrent neural networks [HS97] and recursive neural networks [SLMN11]. These models have achieved state-of-the-art performance in machine translation [BCB14], speech recognition [SKRP15], hand-written recognition [CMS12] as well as other tasks. In 2017, Alpha Go, an automatic computer agency implemented via neural network models [SHM$^+$16], beat Ke Jie, who ranks first among current human Go players.

Although neural network models achieve outstanding performance in different domains, training these models is a well-known difficult problem. The standard method to train neural network models is backpropagation, which is a gradient-based optimization procedure. Backpropagation has several problems, e.g., slow to converge and the vanishing gradient problem [Hoc98]. Moreover, training neural networks via backpropagation is a non-convex optimization procedure. Therefore, even for training a shallow neural network, backpropagation often converges to local optima or saddle points. To alleviate this problem, the multi-initialization method is often used. The parameters of a neural network are randomly initialized multiple times. Since the initial values of the parameters are different, each time the training procedure will converge to a different point. The best local optimum is usually picked as the result of the entire training process.

The drawback of multi-initialization is obvious. Firstly, it is an expensive method. Due to the large amount of parameters, training neural networks is slow. Multi-initialization will make the training process extremely time-consuming. Alternatively, multiple training processes can be executed parallel. But it requires additional computational resources. Secondly, if we only random initialize the parameters finite times, due to the randomness, it is not guaranteed that the global optimum is among the local optima we find. Therefore, to find the global optimum, it is necessary to find an alternative training algorithm.

1.2 Objective

[JSA15] propose a novel procedure to train neural networks. They argue that training a neural network with one hidden layer and a scalar output via orthogonal tensor decomposition, the training procedure is guaranteed to converge to the global optimum. However, this algorithm is not implemented or tested yet. The initial objective of this project is to test this novel algorithm and compare it to backpropagation. Since training via orthogonal tensor decomposition is guaranteed to converge to the global optimum, it should always achieve lower training error than backpropagation. Furthermore, even if the tensor method does not beat backpropagation, it does not necessarily mean the tensor method does not converge to the global optimum. In this case, the point found by the tensor method may not be sufficiently close to the global optimum. Therefore, we argue that by further tuning the parameters via backpropagation, we can push the point sufficiently close to the global optimum. In summary, we expect that the tensor method, or backpropagation with tensor method initialization, will beat backpropagation with random initialization in terms of training errors.

1.3 Results Achieved

During the analysis of the tensor method, we prove that this algorithm can only be applied to learn a particular family of neural networks. Moreover, we prove that in order to apply this novel algorithm, the training data should meet certain assumptions. Real-world data rarely meet these assumptions. Besides, given the training set, we do not know if the data meets these assumptions unless we start the tensor training algorithm. Therefore, we use synthetic data, which satisfies all the assumptions, to test the tensor method. On the synthetic data set, the tensor method always beat backpropagation. Furthermore, the tensor method often outperforms backpropagation dramatically. Therefore, we claim that on the synthetic data set, the performance of the tensor method coincides the statement of guaranteed training.

Other than the orthogonal tensor decomposition (OTD) method [JSA15], we also explore other tensor decomposition methods. The advantage of other tensor decomposition methods is they do not require the data or the neural network to meet these assumptions. However, these tensor decomposition procedures are also non-convex optimization procedures. We compare the alternative least squares (ALS) tensor decomposition [CLDA09], which is the current standard method for tensor decomposition, with OTD and backpropagation. We find ALS always beats backpropagation. We also propose possible explanations for this phenomenon.

1.4 Dissertation outline

This dissertation is organized as follow: Chapter 2 introduces the shallow neural networks and backpropagation. It also discusses related work. Chapter 3 introduces orthogonal tensor decomposition, and prove the problems of this training method. It also gives our novel approach to train shallow neural networks. Chapter 4 discusses and evaluates the experiment results. Chapter 5 presents the conclusion and future work.

Chapter 2

Background

This chapter will introduce shallow neural networks and backpropagation. The intrinsic problems caused by backpropagation will also be discussed.

2.1 Shallow Feed-forward Neural Networks

A shallow feed-forward neural network with a scalar output can be defined as

$$y = a_2\sigma(A_1^T x + b_1) + b_2 \tag{1}$$

$$where\ x \in R^d,\ A_1 \in R^{d\times k},\ b_1 \in R^k,\ a_2 \in R^k,\ b_2 \in R,\ y \in R$$

The structure of such a neural network is shown blow.

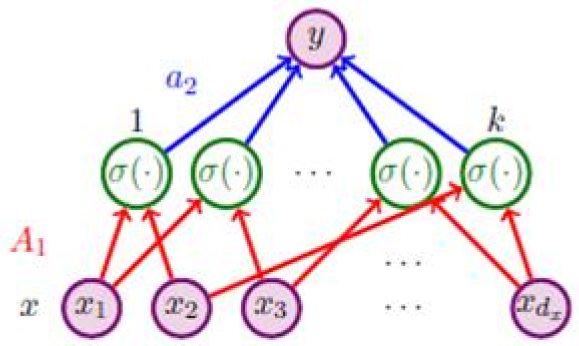

Figure 2.1: The architecture of a two layer neural network with a scaler output. Figure adapted from [JSA15].

The first layer of the neural network is the input layer. The second layer of the neural network is the hidden layer. The function $\sigma(\cdot)$ is the activation function for each unit in the hidden layer. Usually, $\sigma(\cdot)$ is a non-linear function. Thus, other than linear transformation, the neural network can extract features from the data automatically.

For example, one popular activation function is the sigmoid function:

$$\sigma(z) = \frac{1}{1+e^{-z}} \; z = A_1^T x + b_1 \tag{2}$$

For the j-th node in the hidden layer, the weights are $(A_1)_j$ and $(b_1)_j$, where $(A_1)_j$ denotes the j-th column of the matrix A_1 and $(b_1)_j$ denotes the j-th element of the bias vector b_1. Each node has different weights. The output of the sigmoid function shows if the input is near a linear decision boundary as well as the input is on which side of the linear boundary. Since the weights of the hidden units are different, each hidden unit decides the relationship between the input and a unique linear boundary. Thus, by a linear combination of the outputs of the hidden layer, the neural network is able to tell if the input is in a region whose boundary is complicated. Thus, such a structure is suitable for classification.

There are several extensions to shallow feed-forward neural networks, such as deep convolutional neural networks [KSH12] and recursive neural networks [HS97]. However, in the scope of this dissertation, we only discuss the shallow feed-forward neural network with a scalar output, since the tensor decomposition training method can only be applied to this simple model.

2.2 Gradient Descent and Backpropagation

The standard method to train neural networks is backpropagation, which is a gradient-based optimization procedure. The training procedure can be defined as to tune the parameters of the neural network to minimize the differences between the outputs of the model and the desired outputs. We use a loss function to measure the difference. One common loss function is the average square error function, which can be defined as :

$$loss = \frac{1}{N}\sum(f(x_i) - y_i)^2 \tag{3}$$

where $f(\cdot)$ denotes the neural network, (x_i, y_i) denotes the i-th input-output pair in the training set. Different parameters lead to a different loss. The value of the loss function changes most dramatically along the direction of the derivative of the loss function with respect to the parameters. When the loss function achieves its minimum, the derivative is zero. Therefore, to train the model, we could randomly initialize the parameters. Then, we move along the direction of the derivative to minimize the error until we cannot achieve any improvement. This procedure is defined as a gradient-based optimization procedure. The gradient descent algorithm can be described as

1. Initialize the parameters w randomly.
2. Compute the derivative of the loss function with respect to the parameters.
3. Update w by $w \leftarrow w - \eta \cdot \bigtriangledown_w f$, where η is a fixed small number
4. Evaluate $\bigtriangledown_w f$ with the current weight. If the current value of $\bigtriangledown_w f$ is below a threshold, return w. Otherwise, repeat this procedure from step 2.

We can apply this training procedure to train neural networks. Due to the multi-layer structure of neural networks, from the output layer, we propagate the loss layer by layer. The mathematic interpretation of this procedure is the chain rule. For the $l-th$ layer of a feed-forward neural network, let

$$a^{(l)} = A^{(l)} h^{(l-1)} + b^{(l)},\ h^l = \sigma(a^{(l)}),\ \frac{\partial Loss}{\partial a^{(l)}}$$

by the chain rule, we can derive:

1. $\frac{\partial Loss}{\partial b^{(l)}} = \frac{\partial Loss}{\partial a^{(l)}}$
2. $\frac{\partial Loss}{\partial A^{(l)}} = \frac{\partial Loss}{\partial a^{(l)}} h^{(l-1)T}$
3. $\frac{\partial Loss}{\partial h^{(l-1)}} = A^{(l)T} \frac{\partial Loss}{\partial a^{(l)}}$
4. $\frac{\partial Loss}{\partial a^{(l-1)}} = \sigma^{(l-1)'}(a^{(l-1)}) \bigodot \frac{\partial Loss}{\partial h^{(l-1)}}$ where $\bigodot$ denotes elementwise multiply

Through these rules, the derivatives with respect to all parameters can be computed from the top layer of the network to the bottom layer of the network. Once these derivatives are computed, the gradient-descent algorithm can be applied to update the parameters.

2.3 Problems of Gradient-based Optimization

There are several drawbacks of gradient-based optimization procedures. For example, it is well known that training neural networks via backpropagation is time-consuming. Another problem, which we aim to solve in this project, is gradient-descent is not guaranteed to find the global optimum. Instead, it often stuck at a local optimum or a saddle point. Since the derivative of the loss function at a local optimum or a saddle point is zero, if the gradient-descent algorithm initializes near such points, the algorithm will get stuck at these points.

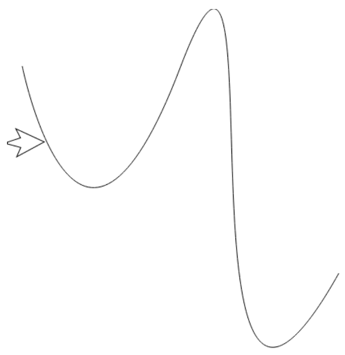

Figure 2.2: An Example of bad initialization. The gradient-based method will stuck at the local optimum and it will never find the global optimum

This problem is particularly severe for training neural networks. Firstly, even for a simple neural network, there may exist exponentially many local minima [Ším02]. Thus, from a probability perspective, it is extremely challenging for the gradient-based training procedure to find the global optimum. Secondly, even for a shallow feed-forward neural network with simple data, local optima may cause the neural network fail trivial tasks. [JSA15] empirically show that local optima may lead a neural network fail to classify two-dimensional data with two parallel lines as the decision boundary. Thirdly, empirical results show that for neural networks, local optima often lead to poor generalization [B+09]. Thus, it is crucial to develop a training algorithm which is guaranteed to converge to the global optimum. [JSA15] argues training shallow neural networks with a scalar output via orthogonal tensor decomposition is guaranteed to converge to the global optimum, if the loss function is the mean squared error function. Next chapter gives detailed description and analysis of this algorithm.

2.4 Related Work

Optimizing a non-convex function is a challenging problem in general. For the training of neural networks, several approaches are proposed to alleviate the problem of stuck at local optima.

[APVZ14] study learning low degree polynomials with a two layer neural network. They show that under two conditions: (1) the number of hidden units is suf-

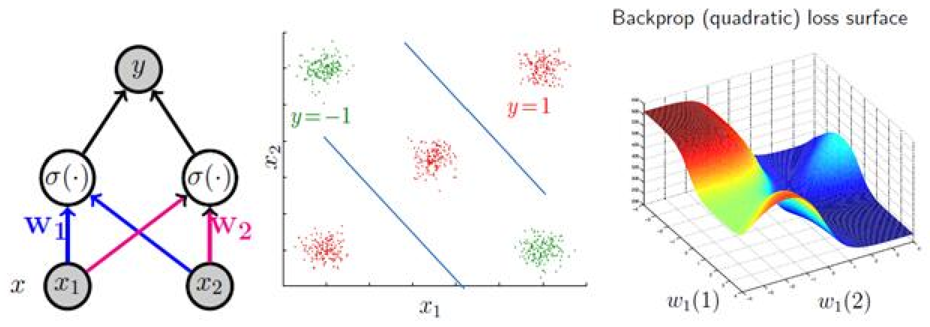

Figure 2.3: A shallow feed-forward neural network fails to classify data correctly. The green lines are the correct decision boundary. However, the neural network classifies the red points as one class. From the loss surface, we see that backpropagation stuck at a local optimum in the dark red region. This local optimum leads the neural network fails this simple task. Figures adapted from [JSA15].

ficiently large; (2) the inputs and the weights are complex numbers, the two layers neural network can learn any low degree polynomial. Moreover, the training process is guaranteed to converge to the global optimum. This training method can be extended to real-value inputs under uniform or Gaussian distribution. However, in real applications, the inputs are rarely complex. Besides, the distribution of the input is often unknown. Moreover, we cannot guarantee the distribution to be uniform or Gaussian.

[HOT06] propose a pre-training procedure to initialize the training of deep belief networks. The pre-training method is an unsupervised algorithm. It processes one layer at a time. The input information is preserved during the pre-training process. For each layer, it learns a non-linear transformation which captures the main variations in its input. Finally, gradient descent is used to fine-tune the parameters. However, this method is only to discrete input.

The technology used for the unsupervised pre-training is the Restricted Boltzmann Machine (RBM). The training of an RBM only requires the unlabeled input data x and it will extract features automatically. The RBM has the form:

$$p(x,h) = e^{-E(x,h)}/Z$$

$$E(x,h) = -h^T W x - c^T x - b^T x.$$

$E(x,h)$ is referred as the energy function. Z is the normalized term and it is often intractable. The factor group representation of an RBM is shown in the figure below.

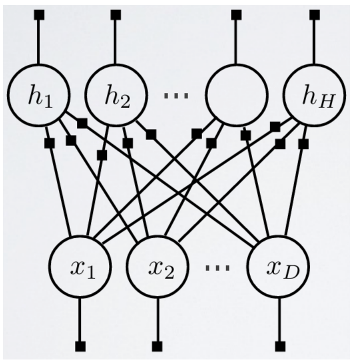

Figure 2.4: The factor group of an RBM [Lar]

For if we restrict the value of each h_i to be $0,1$, then

$$p(x) = \sum_{h\in 0,1^H} p(x) = \sum_{h\in 0,1^H} e^{-h^T Wx - c^T x - b^T x}/Z$$

It can be shown that

$$\sum_{h\in 0,1^H} e^{-h^T Wx - c^T x - b^T x}/Z = exp(c^T x + \sum_{j=1}^{H} log(1+exp(b_j + W_j x)))/Z.$$

To train the RBM, by the principle of the maximum likelihood, we minimize the average negative log-likelihood:

$$\frac{1}{N}\sum -log(p(x_i)).$$

At the minimum, the derivative of the negative log-likelihood with respect to the parameters is zero. To find such a point, the method of contrastive divergence [Hin02] is used. After training, Wx is the extracted feature. We apply the same procedure to the newly built feature layer to make the network go deep.

[B$^+$09] extend the pre-training method to make it be able to process continuous input more properly. Their experiment results also support their hypothesis that the pre-training algorithm will initialize weights in a region near a good local optimum.

[EBC$^+$10] provide a comprehensive study of the pre-training algorithm. Their experiments show that for deep networks with small enough layers, random initialization systematically outperforms pre-training. However, for deep networks with big enough layers, pre-training leads to worse training error but better generalization error. Thus, they argue that pre-training can be viewed as regularization. They also observe that the advantage of pre-training does not vanish with the increasing amount of labeled data, which contradicts to the classical regularizations. One possible explanation is trained by Stochastic Gradient Descent, deep networks are more sensitive to early examples.

However, the training process should also capture the information provided by late examples, especially when the dataset is large. Therefore, pre-training has a consistently good performance regardless of the size of the dataset.

The pre-training algorithm is effective in a number of applications. [DYDA12] propose a pre-trained deep neural network hidden Markov model hybrid architecture for large-vocabulary speech recognition. This model benefits from initializing parameters through the pre-training algorithm. Their results show the model significantly beat the conventional context-dependent Gaussian mixture model on a challenging business search data set.

Chapter 3

Methodology

In this chapter, we introduce the orthogonal tensor decomposition method (OTD) for training neural networks. Second, we analyze this algorithm in detail and point out the problems in the original paper of OTD [JSA15]. Lastly, we describe possible solutions and our new approach to train shallow neural networks with scalar output.

3.1 Training Neural Networks Via Orthogonal Tensor Decomposition

3.1.1 Tensors for Neural Networks

[JSA15] propose using OTD to train a shallow feed-forward neural network. They argue that if the mean squared error is used as the loss function, this training algorithm is guaranteed to converge to the global optimum. Instead of minimizing a loss function, they suggest using the input-output data pairs and the probability density function (PDF) of the input to construct a target tensor. The tensor contains information about the parameters of the neural network. Thus, via tensor decomposition, the parameters can be recovered. This algorithm can only be applied to train shallow feed-forward neural networks with scalar outputs (defined in chapter 2). The reason for this restriction is only in this case we can construct a third order tensor by using the training data and the PDF. If the outputs are vectors, the tensor we construct will be a fourth order tensor. Unfortunately, currently, people do not have the knowledge of guaranteed decomposition methods for tensors whose order are higher than three. Although [JSA15] argues the training method can be extended to shallow neural networks with vector outputs, because of the problems we found about the tensor training algorithm, this extension is infeasible. Later sections will discuss these problems in details.

For any function, [JSA14] show that

$$E[y \cdot S_m(x)] = E[\frac{\partial^m}{\partial x^m} f(x)] \tag{4}$$

where

$$S_m(x) = (-1)^m (\frac{\nabla_x^m p(x)}{p(x)}) \tag{5}$$

and $p(x)$ is the PDF of the input. $S_m(x)$ is also referred as the $m-th$ order score function of x.

For a shallow feed-forward neural network $E[f|x] = f(x) = a_2^T \sigma(A_1^T x + b_1) + b_2$, [JSA15] show that

$$E[\frac{\partial^3}{\partial x^3} f(x)] = \sum_{j \in [k]} \lambda_j \cdot (A_1)_j \bigotimes (A_1)_j \bigotimes (A_1)_j \tag{6}$$

where $\bigotimes$ denotes the tensor product, i.e., $(x \bigotimes y)_{(i,j)} = x_i \cdot y_j$, and $(A_1)_j$ denotes the $j-th$ column of the matrix A_1 . Thus, from (4) and (6),

$$T = E[y \cdot S_3(x)] = E[\frac{\partial^3}{\partial x^3} f(x)] = \sum_{j \in [k]} \lambda_j \cdot (A_1)_j \bigotimes (A_1)_j \bigotimes (A_1)_j \tag{7}$$

Therefore, by using the knowledge of the training data and the PDF of the input, we can construct a tensor which contains the information of the weight matrix for the hidden layer. The reason for we can only apply this algorithm to shallow neural networks is only when there is one hidden layer, the third order derivative of $f(x)$ with respect to x is in the form of (6). This argument can be verified easily via computing the third order derivative of $f(x)$ directly. By decomposing the tensor T, we can recover each column $(A_1)_j$ for the hidden matrix respectively.

3.1.2 Tensor Construction

From (7), we know that to construct the tensor T, we need the knowledge of the desired outputs as well as the PDF of the input. However, finding the PDF of the input is not a trivial problem. In real application, the PDF is rarely known. Although we can estimate the PDF via autoencoders [SFG$^+$17] or via Gaussian Mixture Models (GMMs), training autoencoders via backpropagation or training GMMs via EM algorithm are non-convex optimization procedures, which means we can only have a local optimum estimation of the PDF. Therefore, for our purpose of guaranteed neural network training, both approaches are problematic. For example, if we use autoencoders to estimate the PDF, then it means by using a local optimum of a neural network, we can find the global optimum of another neural network. Intuitively, this argument does not make much sense. Therefore, in our experiments, we decided to generate data from a known PDF as well as to estimate the PDF via GMMs. The performance of the tensor training algorithm will be tested for both cases.

Once we have the PDF of the input, the construction of the tensor is trivial. We can estimate the tensor by the empirical cross-moment $T = \frac{1}{n}\sum_{i\in[j]} y_i \cdot s_3(x_i)$.

3.1.3 Orthogonal Tensor Decomposition Method

For a tensor $D \in R^{n_1\times\cdots\times n_d}$, it may be represented as $D = \sum\lambda_i a_i^1 \otimes \cdots \otimes a_i^d$ where $a_i^d \in R^{n_d}$. The process of recovering such a set of vectors is Can-decomp/Parafac (CP) tensor decomposition. By applying CP tensor decomposition method to the empirical cross-moment $T = \frac{1}{n}\sum_{i\in[j]} y_i \bigotimes s_3(x_i)$, we may recover the columns for the hidden weight matrix A_1. The standard method of CP decomposition is alternative least squares (ALS) [CLDA09]. However, ALS is also a non-convex optimization procedure. Therefore, in general, the decomposition of tensors is not guaranteed to converge to the global optimum.

[AGH$^+$14] propose a guaranteed CP tensor decomposition method for orthogonal third-order tensors. They show that for a tensor $D = \sum_{j\in[k]} \lambda_j \cdot (a)_j \bigotimes (a)_j \bigotimes (a)_j$, if all $(a)_j's$ are orthogonal to each other, then their CP decomposition is guaranteed to converge to the global optimum. The core step of this algorithm is tensor power iteration:

$$u \leftarrow \frac{D(I,u,u)}{|D(I,u,u)|},$$

where $u \in R^d$. $D(M_1, M_2, M_3)$ is the multilinear form of a tensor D, which is defined as

$$D(M_1,M_2,M_3) = \sum\sum\sum T_{j_1,j_2,j_3} \cdot M_1(j_1 :) \bigotimes M_2(j_2 :) \bigotimes M_3(j_3 :) \qquad (8)$$

where $D \in R^{q_1\times q_2\times q_3}$, $D(M_1,M_2,M_3) \in R^{p_1\times p_2\times p_3}$ and $M_i \in R^{q_i\times p_i}$. The OTD algorithm is described in next page.

We should notice that in general, the empirical cross-moment $T = \frac{1}{n}\sum_{i\in[j]} y_i \bigotimes s_3(x_i)$ is not an orthogonal tensor. Our experiments show the performance of OTD on non-orthogonal tensor is extremely unsatisfactory. Therefore, to recover the columns of the hidden matrix, we cannot apply the orthogonal tensor decomposition method to the empirical cross-moment T directly.

Algorithm 1 Orthogonal tensor decomposition method [AGH+14]

```
Input:
    symmetric tensor $T \in R^{k\times k\times k}$ , number of iterations N, number of initializations R.
Output:
    the estimated eigenvector/eigenvalue pair; the deflated tensor.
1: for $i = 1$ to $R$ do
2:     Initialize $v_0^i$ with SVD-based initialization.
3:     for $t = 1$ to $N$ do
4:         Compute Power Iteration Update
               $$v_t^i = \frac{T(I, v_{t-1}^i, v_{t-1}^i)}{|T(I, v_{t-1}^i, v_{t-1}^i)|}$$
5:     end for
6: end for
7: $j = argmax_{i\in[R]} T(v_N^i, v_N^i, v_N^i)$
8: Do $N$ power iteration updates starting from $v_N^j$ to obtain $v$, and set $u = T(v, v, v)$
9: return Return the estimated eigenvector/eigenvalue v/u and the deflated tensor $T' = T - u \cdot v^{\otimes 3}$
```

Algorithm 2 SVD-based Initialization [AGJ14]

```
Input:
    tensor $T \in R^{k\times k\times k}$
1: for $i = 1$ to $\delta$ do
2:     Draw a random standard Gaussian vector $\theta^i$
3:     Compute $u_1^i$ as the top left singular vector of $T(I, I, \theta^i) \in R^{k\times k}$
4: end for
5: $v_0 = max_{i\in[\delta]}(u_1^i)_{min}$
6: return Return $v_0$
```

To solve this problem, [JSA15] propose a whitening algorithm to convert the empirical cross-moment T to an orthogonal tensor. Then, they apply OTD to get intermediate results. Then, they apply a un-whitening process to recover the columns of the hidden matrix A_1 from the intermediate results. However, their whitening and un-whitening processes are problematic. They cannot be applied to an arbitrary data set or an arbitrary neural network. To apply the whitening and un-whitening process, the data set and the neural network should meet certain assumptions. We will discuss this problem in details in later sections. The whitening and un-whitening algorithm are described below.

Algorithm 3 Whitening [JSA15]

```
Input:
    tensor $T \in R^{d\times d\times d}$
1: Compute the second order tensor
       $$M_2 = E[y \cdot S_2(x)]$$
2: Compute the rank-k SVD, $M_2 = UDiag(\gamma)U^T$ , where $U \in R^{d\times k} and \gamma \in R^k$
3: Compute the whitening matrix $W = UDiag(\gamma^{-1/2}) \in R^{d\times k}$
4: return $T(W,W,W) \in R^{k\times k\times k}$
```

Algorithm 4 Un-whitening [JSA15]

```
Input:
    Orthogonal rank-1 components $v_j \in R^k, j \in [k]$
1: Consider matrix $M_2$ which was exploited for whitening, and let $\lambda'_j, j \in [k]$ denotes
   the corresponding coefficients as $M_2 = A_1 Diag(\lambda')A_1^T$.
2: Compute the rank-k SVD, $M_2 = UDiag(\gamma)U^T$ , where $U \in R^{d\times k} and \gamma \in R^k$
3: Compute the
       $$(A_1)_j = \frac{1}{\sqrt{\lambda'_j}} UDiag(\gamma^{1/2})v_j, j \in [k]$$
4: return $T(W,W,W) \in R^{k\times k\times k}$
```

They also suggest using a Fourier method to find the bias term for each hidden unit. The Fourier method is shown in the next page.

After we computing the hidden matrix A_1 and the bias vector b_1, since the output layer is a linear layer, we reduce the problem to a linear regression problem. If the output layer is non-linear, then finding the weight matrix and the bias for the output layer may lead to a non-convex optimization problem. [JSA15] argue that each procedure in their method is guaranteed to converge to the global optimum and hence the entire method is guaranteed.

In summary, the tensor training method constructs a third order tensor $T = \frac{1}{n}\sum_{i\in[j]} y_i \cdot s_3(x_i)$ and a second order tensor $M_2 = \frac{1}{n}\sum_{i\in[j]} y_i \cdot s_2(x_i)$. Then, it applies the whitening procedure to T to construct an orthogonal tensor T'. T' is fed to the orthogonal tensor decomposition method to get intermediate results. The columns of the hidden matrix A_1 are recovered from applying the un-whitening procedure to the intermediate results. The bias for the hidden layer are estimated by the Fourier method. Once we have the hidden matrix and the bias for the hidden layer, since the output layer is a linear layer, we reduce the training to a regression problem.

Algorithm 5 Fourier method for estimating b_1 [JSA15]

```
Input:
    Labeled samples $(x_i, y_i) : i \in [n]$, estimate $A_1$, parameter $\varepsilon$, the probability density function $p(x)$ of the input $x$.
1: for $l = 1$ to $k$ do
2:     Let $\Omega_l = \{\omega \in R^d : \|\omega\| = \frac{1}{2}, |< \omega, (A_1)_l >| \geq \frac{1-\varepsilon^2/2}{2}\}$
3:     Draw $n$ i.i.d. frequencies $\omega_i$, $i \in [n]$, uniformly from set $\Omega_l$
4:     Let $v = \frac{1}{n}\sum \frac{y_i}{p(x_i)} e^{-j<x_i,\omega_i>}$, which is a complex number. Let $\angle\cdot$ denotes the phrase operator.
5:     Let $b_1(l) = \frac{1}{\pi}(\langle V - \langle \Sigma(1/2)$, where $\sigma(x)\ Fourier \rightarrow \Sigma(x)$, $Fourier \rightarrow$ denotes the Fourier transform
           $$\Sigma(\omega) = \int_{-\infty}^{\infty} \sigma(x) e^{-j<x,\omega>} dx$$
6: end for
7: return $b$
```

3.2 Analysis of the Tensor Methods

In this section, we analyze some key steps in the tensor methods and show there exists several problems.

3.2.1 The SVD of M_2

The whitening/un-whitening procedure involves the SVD of M_2. In this procedure, M_2 should be decomposed into the form of $M_2 = UDiag(\gamma)U^T, Diag(\gamma)_{(i,i)} > 0$ via SVD. However, in general, unless M_2 is positive-definite, this decomposition is infeasible. Suppose $M_2 \in R^{n\times n}$ is not positive-definite, and $M_2 = UDiag(\gamma)U^T, Diag(\gamma)_{(i,i)} > 0$.

Then, for any vector $X \in R^n$, consider $X^T AX$.

$$X^T AX = (X^T U)Diag(\gamma)(U^T X)$$

$$X^T AX = M^T Diag(\gamma)M$$

since $Diag(\gamma)_{(i,i)} > 0$,

$$M^T Diag(\gamma)M > 0$$

$$M^T Diag(\gamma)M = X^T AX > 0$$

which contradicts the assumption that A is not positive-definite.

Next, we argue that M_2 is not guaranteed to be positive-definite. M_2 is constructed via $M_2 = \frac{1}{n}\sum_{i\in[j]} y_i \bigotimes s_2(x_i)$, which is an estimation of $M_2 = E[y \cdot s_2(x)]$. Since $E[f''(x)] = E[y \cdot s_2(x)]$ [JSA14], for $f(x) = a_2^T \sigma(A_1^T X + b_1) + b2$, by the rule of derivatives,

$$M_2 = E[f''(x)] = \Sigma\, \lambda'_j \cdot (A_1)_j \bigotimes (A_1)_j$$

where

$$\lambda'_j = E[\sigma''((A_1^T X + b_1)_j)] \cdot (a_2)_j.$$

Thus,

$$X^T M_2 X = X^T \ (\Sigma \lambda'_j \cdot (A_1)_j \bigotimes (A_1)_j)\ X$$

$$X^T M_2 X = \Sigma \lambda'_j (X^T (A_1)_j \bigotimes (A_1)_j X).$$

Since

$$X^T (A_1)_j \bigotimes (A_1)_j X > 0$$

for any X and $(A_1)_j$, the sign of $X^T MX$ is only decided by λ'_j. However, before training, we do not have any information about the neural network, so λ'_j is unknown. Therefore, there exists a fault in the tensor training algorithm. To find the parameters, we need to construct a positive-definite second order tensor M_2. However, to ensure the matrix M_2 to be positive-definite, the parameters should be set explicitly, but our goal is to find the parameters. Moreover, even if we already have a set of parameters which is the global optimum, it is possible that these parameters make all λ' negative. Thus, even if we already know the global optimum, and each y_i and each $f(x_i)$ is exactly same, $M_2 = \frac{1}{n}\sum_{i\in[j]} y_i \bigotimes s_2(x_i)$ is not guaranteed to be positive-definite. Our experiments also confirm our arguments.

For M_2, it is always possible to decompose it into

$$M_2 = UDiag(\gamma)V^T, Diag(\gamma)_{(i,i)} > 0.$$

Since in the whitening process, only U and $Diag(\gamma)$ are exploited, it seems that it is not required to decompose M_2 into $M_2 = UDiag(\gamma)U^T$, so M_2 does not have to be positive-definite. However, in next section, we will show that to convert the tensor T to an orthogonal tensor, M_2 has to be positive-definite.

3.2.2 Analysis of the Whitening/Un-Whitening Process

For

$$T = \sum_{j\in[k]} \lambda_j \cdot (A_1)_j \bigotimes (A_1)_j \bigotimes (A_1)_j$$

$$M_2 = \sum_{j\in[k]} \lambda'_j \cdot (A_1)_j \bigotimes (A_1)_j = UDiag(\gamma)U^T, Diag(\gamma)_{(i,i)} > 0$$

$$W = UDiag(\gamma^{-1/2})$$

in the whitening process [JSA15], we aim to construct an orthogonal tensor T' by apply the multi-linear form (equation 8) to T and W:

$$T' = T(W,W,W) = \sum_{j\in[k]} \lambda_j (W^T(A_1)_j)^{\otimes 3}$$

$$\sum_{j\in[k]} \lambda_j (W^T(A_1)_j)^{\otimes 3} = \sum_{j\in[k]} \frac{\lambda_j}{\lambda_j'^{3/2}} (W^T(A_1)_j \sqrt{\lambda'_j})^{\otimes 3}.$$

Let $v_j = (W^T(A_1)_j\sqrt{\lambda'_j})$, $V = [v_1, v_2, ..., v_k] \in R^{k\times k}$. Now, consider VV^T.

$$VV^T = W^T A_1 Diag(\lambda') A_1^T W = W^T M_2 W$$

Since $UU^T = I$,

$$W^T M_2 W = Diag(\gamma^{-1/2}) U^T U Diag(\gamma) U^T U Diag(\gamma^{-1/2}) = I.$$

Thus, V is an orthogonal basis and T' is an orthogonal tensor. If M_2 is not positive definite, then $M_2 = UDiag(\gamma)V^T$. In this case,

$$W^T M_2 W = Diag(\gamma^{-1/2}) U^T U Diag(\gamma) V^T U Diag(\gamma^{-1/2}) \neq I.$$

Thus, to ensure T' is orthogonal, M_2 must be positive-definite.

In summary, to ensure that the whitening procedure works properly, we need a method to construct the second order tensor M_2, such that M_2 has the same decomposition as T and M_2 is positive-definite. Moreover, even if we have a method to guarantee M_2 is positive-definite, we still cannot find the exact value of the columns of the hidden matrix.

For $V = W^T A_1 Diag(\lambda'^{1/2})$, substituting the whitening matrix $W = UDiag(\gamma^{-1/2})$ in V,

$$V = Diag(\gamma^{-1/2}) U^T A_1 Diag(\lambda'^{1/2}).$$

Multiple $UDiag(\gamma^{1/2})$ to the left,

$$UDiag(\gamma^{1/2})V = UDiag(\gamma^{1/2})Diag(\gamma^{-1/2})U^T A_1 Diag(\lambda'^{1/2})$$

$$UDiag(\gamma^{1/2})V = A_1 Diag(\lambda'^{1/2}).$$

Thus,

$$(A_1)_j = \frac{1}{\sqrt{\lambda'_j}} UDiag(\gamma^{1/2}) v_j.$$

However,

$$M_2 = \Sigma \lambda'_j \cdot (A_1)_j \bigotimes (A_1)_j,\ \lambda'_j = E[\sigma''((A_1^T X + b_1)_j)] \cdot (a_2)_j.$$

Before training, we have no information about the value of λ'_j. To find A_1, we need the value of λ'. To find λ', we need A_1 as well as other parameters. Therefore, we cannot find the exact value of A_1 via the un-whitening process. For each column $(A_1)_j$, there is an unknown constant $\frac{1}{\sqrt{\lambda'_j}}$.

In fact, M_2 does not have to be constructed via $E[y \cdot s_2(x)]$ so λ'_j does not have to be $E[\sigma''((A_1^T X + b_1)_j)] \cdot (a_2)_j$. As long as $\lambda'_j > 0$ for all j, M_2 is positive-definite. In summary, in order to make the whitening/un-whitening procedure works properly, we need a method to construct M_2, such that:

1. M_2 is positive-definite.

2. M_2 has the same decomposition as T, i.e., $M_2 = \Sigma \lambda'_j \cdot (A_1)_j \bigotimes (A_1)_j$.

3. The value of λ'_j is known for all j.

Suppose we have such a method, for $A_1 \in R^{d \times k}$, if $d \times k \leq (1 + 2 + \cdots + d)$, we do not need the tensor training method or backpropagation to recover A_1. A_1 can be found via solving quadratic equations. $M_2 \in R^{d \times d}$ and M_2 is symmetric, so $M_2 = \Sigma \lambda'_j \cdot (A_1)_j \bigotimes (A_1)_j$ has $1 + 2 + \cdots + d$ (i.e., number of elements in the upper/lower triangle of the matrix) equations. Since $d \times k \leq (1 + 2 + \cdots + d)$, in $M_2 = \Sigma \lambda'_j \cdot (A_1)_j \bigotimes (A_1)_j$, the unknown variables (i.e. $d \times k$ elements in A_1) are less than the equations. By the method M_2 is constructed, it is known M_2 must have such a decomposition. Therefore, the quadratic equations $M_2 = \Sigma \lambda'_j \cdot (A_1)_j \bigotimes (A_1)_j$ must be solvable. In summary, if the number of hidden units k and the number of input units d satisfy $d \times k \leq (1+2+\cdots+d)$, A_1 can be found via solving quadratic equations. Therefore, if there exists a method to construct M_2 such that M_2 has the last two properties, the problem of training a shallow neural network becomes trivial. Thus, intuitively, finding such a method is extremely challenging, if not impossible.

3.3 Possible Solutions to the Problems in the Tensor Methods

In this section, we state several assumptions on the data set and the neural network so OTD can be applied properly. We also consider alternative training approaches.

3.3.1 Assumptions about Data and the Neural Network

Firstly, we assume that in the parameter set which is the global optimum, each column of A_1 is normalized. For $(A_1)_j = \frac{1}{\sqrt{\lambda'_j}} UDiag(\gamma^{1/2})v_j$, λ'_j is unknown. Because of the assumption, $(A_1)_j$ can be found by normalizing the vector $UDiag(\gamma^{1/2})v_j$. Thus, the knowledge of λ'_j is not required.

Since $M_2 = E[f''(x)] = \Sigma\, \lambda'_j \cdot (A_1)_j \bigotimes (A_1)_j$ where $\lambda'_j = E[\sigma''((A_1^T X + b_1)_j)] \cdot (a_2)_j$, if $\sigma''((A_1^T X + b_1)_j) > 0$ and $(a_2)_j > 0$, M_2 is guaranteed to be positive-definite. For all widely-used activation functions, we only find the softplus function:

$$\sigma(z) = In(1+e^z),$$

satisfies $\sigma''(z) > 0$ for all z. Therefore, we restrict the activation function to the softplus function. We also assume for the global optimum, all elements in the vector a_2 are positive.

T is constructed via $\frac{1}{n}\Sigma y_i \cdot s_3(x_i)$. Therefore, M_2 and T has the same decomposition.

Since M_2 is estimated by $\frac{1}{n}\Sigma y_i \cdot s_2(x_i)$, if the different between y and $f(x)$ is significant, even if the neural network meets all assumptions, $\frac{1}{n}\Sigma y_i \cdot s_2(x_i)$ may not be positive-definite. Thus, we assume for the global optimum, $f(x) = y$ and the data is noiseless.

In summary, we assume that the activation function is the softplus function, at the global optimum, $(a_2)_j > 0$ for all j, $f(x) = y$ and the data is noiseless to guarantee that M_2 is positive-definite. Since both M_2 and T are constructed via the second order and the third order score function respectively, they have the same decomposition. Since we assume the norm of the columns in A_1 is one, we do not need the value of λ'_j to find $(A_1)_j$.

In really-world application, the data rarely meets all these assumptions. Therefore, we also consider alternative tensor decomposition method.

3.3.2 Alternative Tensor Decomposition Method

Currently, the alternative least squares (ALS) is the standard method of tensor decomposition [CLDA09]. It does not require the tensor to be orthogonal. Therefore, for ALS, the whitening/unwhitening process is not necessary. We can apply ALS directly to T. However, as an inherit problem/property of any tensor decomposition method, ALS can only find a set of normalized vectors. Therefore, we still need to assume the norm of $(A_1)_j$ is one. We also should notice ALS is a non-convex optimization procedure. Thus, the global optimum is not guaranteed.

3.4 Bias Estimation

[JSA15] propose using a Fourier method (Algorithm 5) to find the bias for the hidden layer. This algorithm involves the Fourier transform of the activation function σ:

$$\Sigma(\omega) = \int_{-\infty}^{\infty} \sigma(x) e^{-j<x,\omega>} dx.$$

However, the Fourier transform of the activation function may not exist. If the Fourier transform of a function $f(x)$ exists, then $f(x)$ is absolutely integrable [Cha87]:

$$\int_{-\infty}^{\infty} |f(x)|\, dx < \infty.$$

However, since the softplus function $f(x) = ln(1+e^x)$ is unbounded, this function is not absolutely integrable. Therefore, the Fourier transform of the softplus function does not exist.

We may consider using other activation functions. However, for other activation functions, such as the sigmoid and the tanh, the second order derivatives are not positive for arbitrary input. Therefore, it is difficult to ensure the second order tensor M_2 is positive-definite.

Moreover, even if the tensor is not restricted to be orthogonal so the activation function can be chosen arbitrarily, the Fourier method is still problematic. If $f(x)$ is absolutely integrable, then

$$\lim_{x\to\infty} f(x) = 0, \lim_{x\to-\infty} f(x) = 0.$$

For most activation functions, such as the ReLu function, the sigmoid function and the tanh function, this condition does not hold, since they are either unbounded or converge to a non-zero number. So, they are not absolutely integrable. Since they are not absolutely integrable, the Fourier transform of these functions does not exist. In fact, among popular activation functions, we only found the Gaussian function ($f(x) = e^{-x^2}$) and the Sinc function ($f(x) = \frac{sin(x)}{x}$) are absolutely integrable and have Fourier transform.

We could add an extra entry whose value is 1 to the input vector x to make the training algorithm learn the hidden matrix and the bias in the same training procedure. However, to apply the tensor methods, the PDF of the input is required. Thus, to have a valid PDF, we add an additional entry whose mean is 1 and whose variance is a significantly small number to the input vector x. Intuitively, this manipulation is almost equivalent to move the data points parallel to the plane $x_{d+1} = 1$. Thus, the new PDF gains no additional information about the data. In our experiments, after the manipulation, the elements in the tensor which correspond to the original input vector remain almost exactly same, and all the elements which correspond to the additional entry are zeros. Our experiment results confirm our intuition that we cannot find the bias by adding an extra entry to the input vector.

Thus, in general, the Fourier method cannot be used to find the bias for the hidden layer. In our experiments, we assume the bias is known. Both the tensor method and backpropagation do not need to learn the bias.

3.5 Initialize Backpropagation with ALS

Since the guaranteed tensor training method cannot be applied to the real data in general, we purpose a new approach to initialize backpropagation. Firstly, since in real application, the PDF of the input is often unknown, we use GMMs to estimate the PDF. We construct the tensor T via y and the estimated PDF. Since in this case, we cannot guarantee the whitening/unwhitening process works properly, we apply ALS to decompose T directly. Then, we have an estimation of the hidden matrix A_1. We use the estimated matrix to initialize backpropagation. Other parameters of the neural network are initialized randomly.

Although our method is not guaranteed to converge to the global optimum, we argue that our initialization method can avoid bad initialization points. In the process of constructing the tensor T, the PDF is exploited. Thus, compared to random initialization, our initialization method considers additional information. Thus, it can avoid bad initialization points.

It should be noticed that our new approach also makes one assumption about the shallow neural network: each column in the hidden matrix is a unit vector. Intuitively, if the optimum hidden matrix does not meet this assumption, our initialization method is still better. In this case, each estimated column only needs to time a constant to get close to the global optimum/good local optimum. Since we fine-tune the hidden matrix via backpropagation, we still argue our initialization method outperforms random initialization.

Chapter 4

Results and Evaluation

4.1 Experimental Setup

To apply the tensor training method, the data and the neural network need to meet several assumptions. Moreover, to construct the tensor T and the matrix M_2, the PDF of input x is required. In real application, the data rarely meet these assumptions and the PDF of input is seldom known. Although the PDF can be estimated via autoencoders [SFG+17] or Gaussian mixture models (GMMs), both approaches are non-convex yet our goal is to find a method of guaranteed training. Thus, we decided to test the tensor training algorithm on the synthetic data.

We use multivariate normal distributions with zero means and identity covariance matrices to generate input x. Then, we feed x into a neural network whose activation function is the softplus function and all elements in the vector a_2 are positive. Other parameters of the neural network are randomly generated. The output of this neural network is the output y in the data set. After x and y are generated, we apply the EM algorithm to find a GMM with five components to estimate the PDF. Both the true PDF and the estimated PDF are used in computing the score function. The results of using the true PDF for training and using the estimated PDF for training are compared. We also apply backpropagation to the synthetic data to compare it with the tensor training methods.

We use the ALS method in the MATLAB package ”Tensor Toolbox” [BK+15] as the non-orthogonal tensor decomposition method. The orthogonal tensor decomposition method is implemented via MATLAB [Inc15] on a DICE machine.

The methods to find the parameters of the neural network is summarized below.

Table 4.1: Methods Summarization

Methods to find A_1
Backpropagation
OTD with whitening/un-whitening procedure
ALS on T, without whitening/un-whitening procedure
Methods to find b_1
Given as known condition
Backpropagation with random initialization
Backpropagation with A_1 initialized by tensor methods

4.2 Dataset Description

Through the procedure in the previous section, We generate training sets of different sizes. The sizes of training sets are 5000,10000,15000, 20000 and 25000 respectively. For each size, we generate three training sets.

We mainly generate two dimensional input data since backpropagation may fail trivial tasks on two dimensional input data[JSA15] and the limitation of computing resources. For 10000 two dimensional input data, it takes about 3 hours for the DICE machine to finish all training procedures via the tensor methods. For the two dimensional input data, we use neural networks with two hidden units to generate the output data y.

We also run experiments on high dimension data to test if the tensor methods can be applied to more complicated tasks. We also generate ten dimensional input data. However, for 10000 ten dimensional input data, it takes about 40 hours for the DICE machine to finish all training procedures via the tensor methods. The most expensive part is using the estimated PDF to construct the tensor T and the matrix M_2. Thus, for the ten dimensional input data, due to the limitation of computing resources, we only use the true PDF, i.e., the multivariate normal distribution, to construct T and M_2. We use neural networks with five hidden units to generate the output data y.

We should notice that the tensor decomposition is not expensive. The complexity is $O(ndk)$, where n is the number of data points, d is the dimension of the input, k is the number of the hidden units [JSA15]. The construction of T and M_2 are extremely time consuming. Since they are constructed via the average over the product of the output y_i and the score function $s_m(x_i)$, the construction is parallelable. Thus by parallel computing, the construction of T and M_2 can speed up significantly.

4.3 Evaluation Metrics

The assumption of the tensor training algorithm with orthogonal tensor decomposition is it converges to the global optimum. Thus, it can only be evaluated on the training set. Although overfitting is possible, for this task, we should not consider this problem: the goal is to test if the tensor methods converge to the global optimum, regardless if the global optimum leads to overfitting or not. Thus, the development set and the test set are not needed. We only evaluate different training methods on the test set. We use the mean squared error (MSE) as the measure of loss, since [JSA15] argue their method is guaranteed to converge to the global optimum with respect to this loss function.

4.4 Experiment Results and Evaluations

We use "BP" denotes backpropagation and "OTD" denotes "orthogonal tensor decomposition with the whitening/un-whitening procedure".

When the value of the bias and the PDF of input x are given as known conditions, the results for two dimensional input data are shown in the figure below.

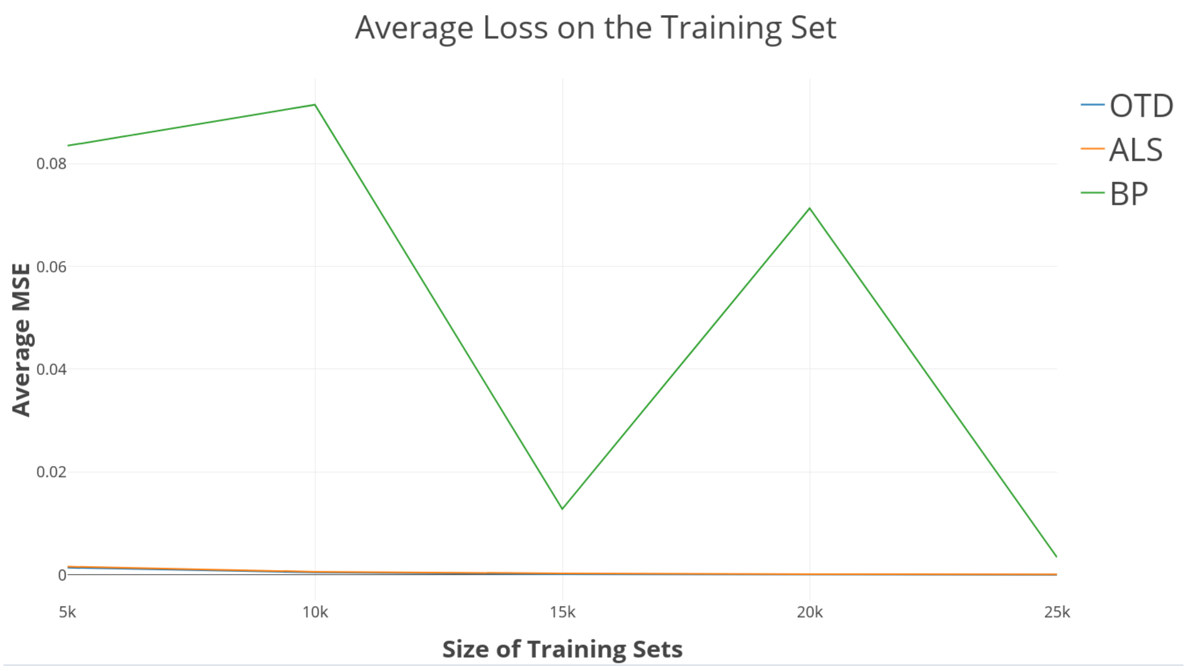

Figure 4.1: The average MSE for the three training algorithms. Each point is the average MSE on three randomly generated dataset.

From Figure 4.1, it is clearly that both ALS and OTD outperform BP significantly.

The next figure displays the performances of ALS and OTD in a small scale.

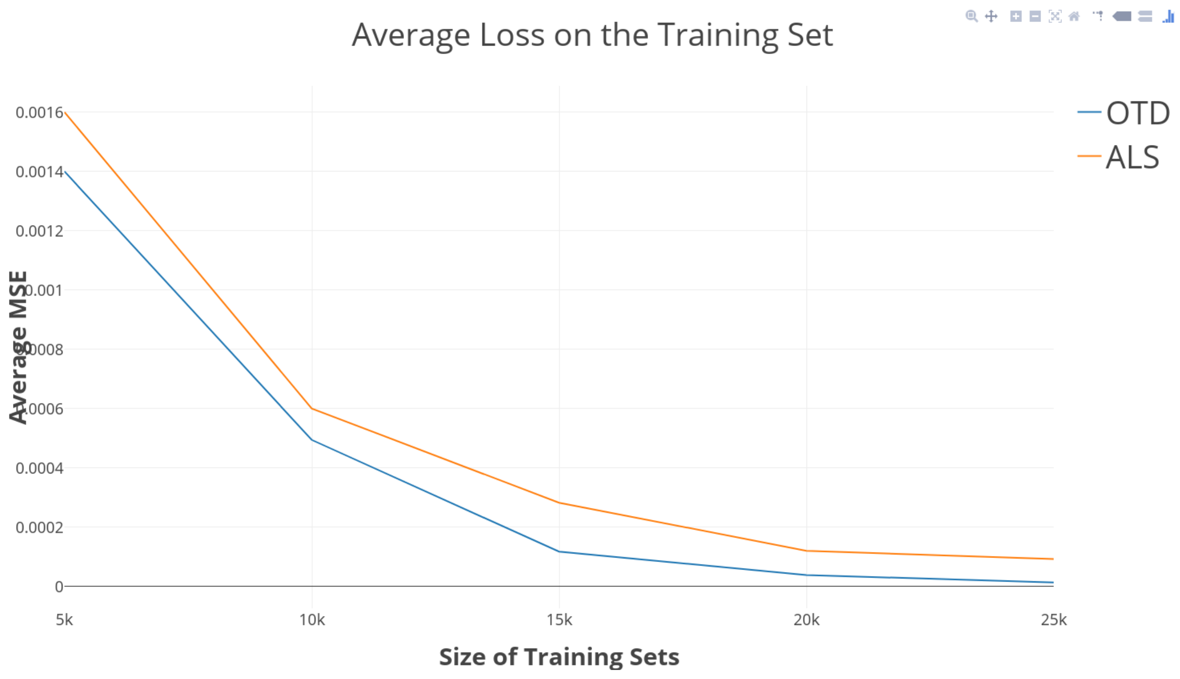

Figure 4.2: The average MSE for OTD and ALS

Although the differences between the loss of ALS and OTD is not significant, OTD always outperforms ALS. Also, we notice that when the size of the data set is 25000, the loss of OTD is almost zero. The exact value of this loss is 1.31×10^{-5}. Thus, for the two dimensional input, the tensor methods outperform backpropagation significantly. Also, as the increasing size of the data set, the loss goes to zero monotonically. These observations satisfy our assumption that on the synthesis data, the OTD is guaranteed to converge to the global optimum. One example of the original hidden matrix A_1 and the hidden matrix learned by OTD is shown in the table below.

Table 4.2: Example of A_1 Learned by OTD

The Original A_1	
0.3769	-0.2511
0.9263	0.9680
The Estimated A_1	
0.4352	-0.3388
0.9003	0.9408

On the same training set, we also use a mixture of Gaussians with five components to estimate the PDF. Then, we use the estimated PDF to construct the tensor T and the matrix M_2. We observe that for OTD, using the estimated PDF outperforms using the original PDF once. The results of using OTD for training with the estimated PDF and true PDF are shown in the figure below.

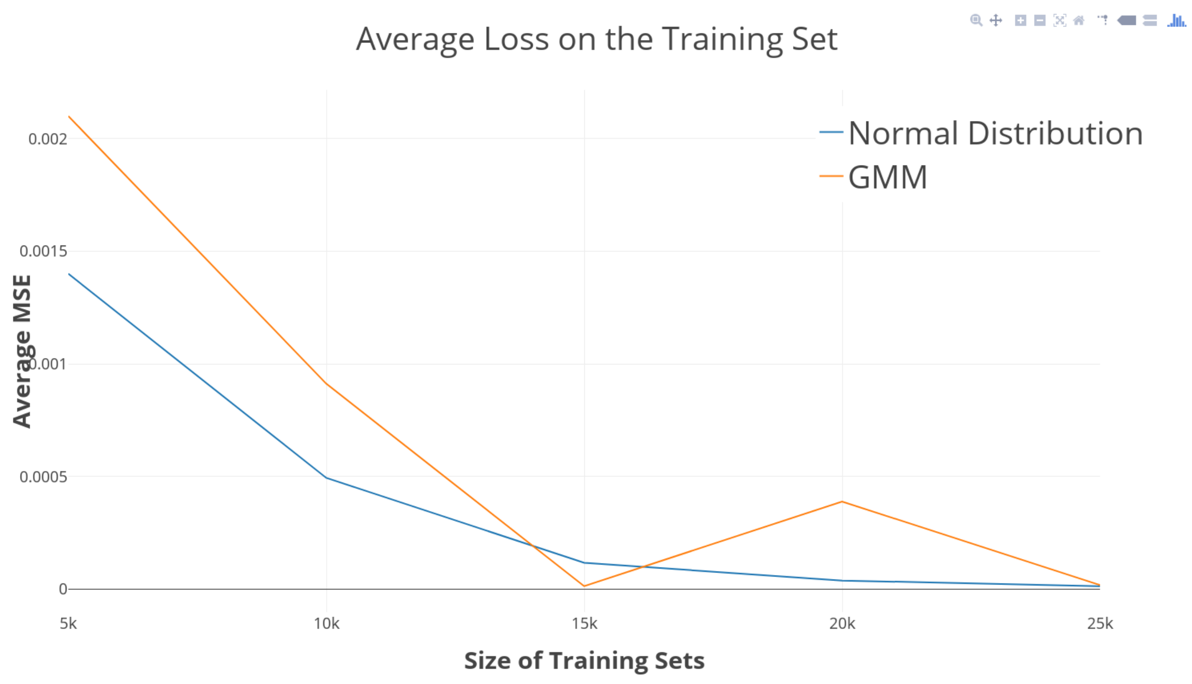

Figure 4.3: The average MSE for constructing the tensor T via the original normal distribution and GMMs.

We notice that when the size of the data set is 15000, using the estimated PDF outperforms using the original PDF. One possible explanation is in this experiment, the GMM explains the distribution of the generated data points more properly, i.e., the likelihood of the data under the GMM is higher than the likelihood under the normal distribution. Therefore, the tensor constructed by exploiting the GMM reflects the structure of the data set more properly. However, as the increasing of the size of the training set, the likelihood of the data under the original normal distribution is also increasing. Thus, using the original PDF to construct the tensor should outperform using the GMM to construct the tensor eventually.

We also test our new approach of parameter initialization. We use GMMs to estimate the PDF. Then, we use ALS to initialize the hidden matrix A_1. We compare our method to backpropagation with random initialization.

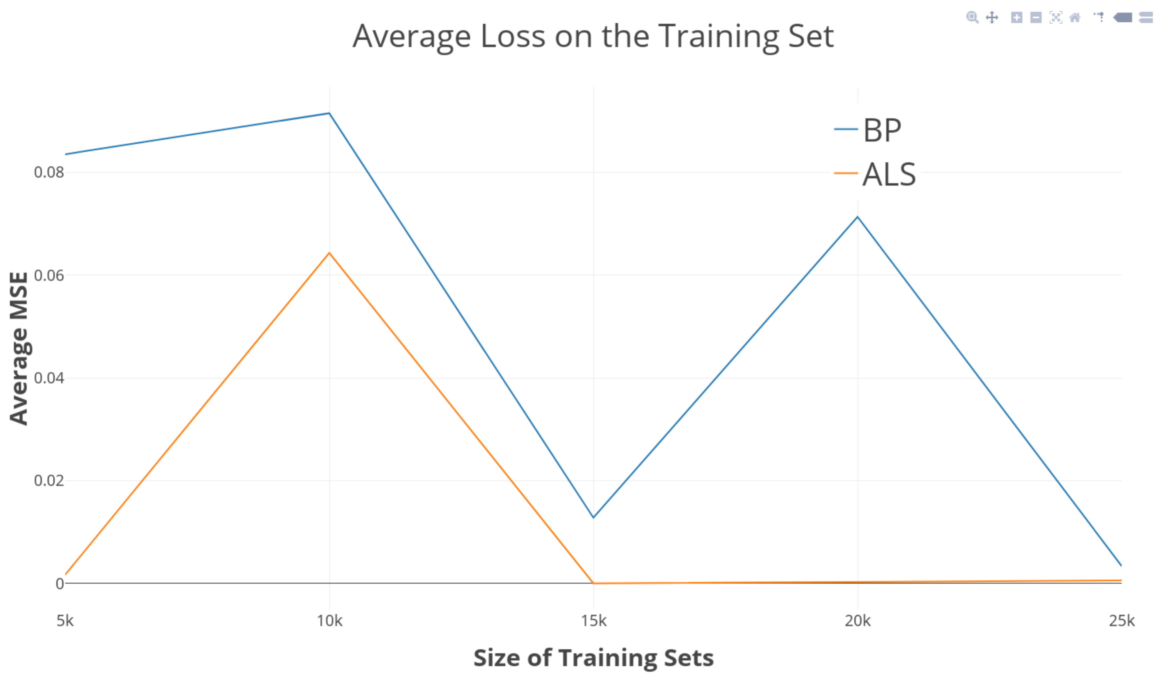

Figure 4.4: The comparison between backpropagation and ALS with GMMS

From Figure 4.4, ALS with GMMs always outperforms backpropagation with random initialization. Therefore, it may be suitable to initialize A_1 via ALS with GMMs. Also, the unstable performance of ALS with GMMs is expected since both of these two methods are non-convex.

We also run experiments on ten dimensional data with known bias and the original PDF. We find when the size of the data set is small (i.e. 5000 in our experiments), the performance of OTD is similar to backpropagation. Because the data points in high dimension space are usually sparse, it requires numerous points to reflect the original PDF. The distribution of a small number of data points may not be well explained by the true PDF. Thus, the tensor constructed via the original PDF does not reflect the structure of the small training set properly.

Moreover, we notice that ALS outperforms OTD. We suppose the reason is compared to ALS, OTD relies more on the PDF. OTD uses the third order T and the second order M_2. Both T and M_2 are constructed by exploiting the PDF. However, M_2 is not required for ALS. Thus, ALS is less sensitive to the PDF. Based on our observation on two dimensional input data, we expect with the increasing size of the training set, the OTD will beat two other methods eventually.

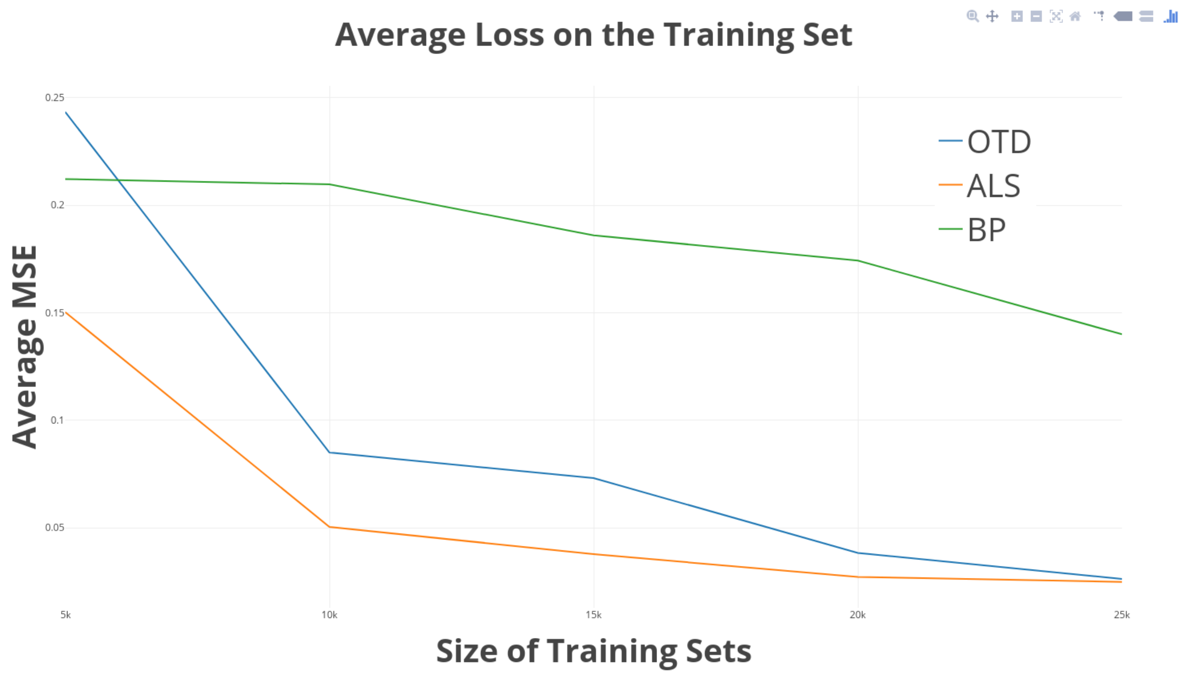

Figure 4.5: The comparison between different training methods on ten dimensional input.

From our results, it is obvious that as the increasing size of the training set, the loss of OTD drops most rapidly. It coincides our intuition that it is most sensitive to the PDF. The loss of ALS also decreases quickly. Thus, it is possible that one important reason for the tensor methods beating backpropagation is they exploit the PDF during the training process.

Chapter 5

Conclusion and Discussion

5.1 Conclusion

Training neural network via backpropagation is a non-convex optimization procedure and the training process tends to stuck at local optima [Ším02]. [JSA15] propose a novel approach to train shallow neural networks with a scalar output. They argue that this approach is guaranteed to converge to the global optimum. We analyze their algorithm and show to apply the algorithm, the data set and the shallow neural network must satisfy several assumptions. Firstly, the noise should be sufficiently small. Second, the activation function for the shallow neural network must be the softplus function. Thirdly, all elements in the weight vector for the output layer must be strictly positive. Fourthly, each column vector of the hidden matrix must be a unit vector. Lastly, the bias for the hidden units must be known before training.

Since it is difficult, if not impossible, for real data to satisfy all these assumptions, we implement this novel algorithm and test is on the synthetic data. On the synthetic data, our experiments coincide the statement that this algorithm is guaranteed converge to the global optimum.

We also propose using tensor decomposition methods to initialize the hidden matrix for backpropagation. Our method only requires one assumption: the column vectors of the hidden matrix are unit vectors. Our experiments show our method beats backpropagation with random initialization. Moreover, intuitively, we argue that with no restriction on the neural network, our new initialization method can still beat backpropagation with random initialization.

5.2 Discussion and Future Work

Due to the limitation of the computing resource, we mainly run experiments on two dimensional input data. More experiments on high dimensional input data may be necessary.

We only test our method to initialize backpropagation on the synthetic data. Since our method only has one weak assumption on the neural network, we can test our method in real data set. Also, we intuitively argue that the assumption on the neural network may not be necessary for our initialization method. Thus, if our initialization method outperforms random initialization consistently in real data, we may claim we find a new approach to train neural networks. A rigorous proof of why our initialization method outperforms the random initialization is also needed.

We only use the tensor methods to train two layer neural networks with a scalar output since the tensor T must be a third order tensor. For a vector output, the tensor

$$T = E[y \bigotimes S_3(x)] \approx \frac{1}{n} \sum_{i \in [j]} y_i \bigotimes s_3(x_i)$$

is a fourth order tensor. [JSA15] purpose reduce T via the multilinear form

$$T' = T(\theta, I, I, I)$$

where θ is a random vector. However, to apply the OTD method, the third order tensor $M_2 = E[y \otimes S_2(x)]$ must be reduced to a matrix. But it is difficult to guarantee the result is a positive-definite matrix. Nevertheless, since our initialization method does not need M_2, we may extend our method to train a shallow neural network with a vector output.

Moreover, our initialization method can be extended to train neural networks with a non-linear output layer. Such a neural network can be described as $f(x) = g(A_2^T \sigma(A_1^T x + b_1) + b_2)$, where $g(\cdot)$ is a non-linear activation function. If the activation function is injective, then the reverse function $g^{-1}(\cdot)$ is well defined. Inverting the activation function will cast the network to a shallow neural network with linear output layer. Therefore, we reduce the problem to $g^{-1}(f(x)) = A_2^T \sigma(A_1^T x + b_1) + b_2$. Intuitively, if the activation function $g(z)$ is a differentiable well-behaved function, the parameters minimize $L(g^{-1}(y), z)$ will lead to a small value of $L(g(z), y)$, where $L(\cdot)$ denotes a loss function: suppose z is sufficient close to $g^{-1}(y)$, because g is differentiable, $g(z)$ will be sufficient close to $g(g^{-1}(y)) = y$ and $L(g(z), y)$ will be small. Therefore, we may initialize A_1 via applying our initialization method to $g^{-1}(f(x)) = A_2^T \sigma(A_1^T x + b_1) + b_2$. Then we use backpropagation to fine-tune the parameters.

In our experiments, since the goal is to test if the tensor methods converge to the global optimum of the mean squared error loss function on the training set, we do not consider evaluate these methods on a test set. However, in the real-world application, we should consider the generalization error. Since our new approach to initialize backpropagation can be applied to real data, we could evaluate if our method generalizes well in general. Intuitively, when the training set is small, since the estimated PDF may not be similar to the true PDF, our method may be prone to overfitting. When the training set is large, compared to backpropagation with random initialization, our method may have lower training error and lower generalization error. It is also interesting to see our method's performance after applying regularization methods.

Bibliography

[AGH$^+$14] Animashree Anandkumar, Rong Ge, Daniel J Hsu, Sham M Kakade, and Matus Telgarsky. Tensor decompositions for learning latent variable models. *Journal of Machine Learning Research*, 15(1):2773–2832, 2014.

[AGJ14] Animashree Anandkumar, Rong Ge, and Majid Janzamin. Guaranteed non-orthogonal tensor decomposition via alternating rank-1 updates. *arXiv preprint arXiv:1402.5180*, 2014.

[APVZ14] Alexandr Andoni, Rina Panigrahy, Gregory Valiant, and Li Zhang. Learning polynomials with neural networks. 2014.

[B$^+$09] Yoshua Bengio et al. Learning deep architectures for ai. *Foundations and trends® in Machine Learning*, 2(1):1–127, 2009.

[BCB14] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[BK$^+$15] Brett W. Bader, Tamara G. Kolda, et al. Matlab tensor toolbox version 2.6. Available online, February 2015.

[Cha87] David C Champeney. *A handbook of Fourier theorems*. Cambridge University Press, 1987.

[CLDA09] Pierre Comon, Xavier Luciani, and André LF De Almeida. Tensor decompositions, alternating least squares and other tales. *Journal of chemometrics*, 23(7-8):393–405, 2009.

[CMS12] Dan Ciregan, Ueli Meier, and Jürgen Schmidhuber. Multi-column deep neural networks for image classification. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 3642–3649. IEEE, 2012.

[DYDA12] George E Dahl, Dong Yu, Li Deng, and Alex Acero. Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. *IEEE Transactions on Audio, Speech, and Language Processing*, 20(1):30–42, 2012.

[EBC$^+$10] Dumitru Erhan, Yoshua Bengio, Aaron Courville, Pierre-Antoine Manzagol, Pascal Vincent, and Samy Bengio. Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research*, 11(Feb):625–660, 2010.

[Hin02] Geoffrey E Hinton. Training products of experts by minimizing contrastive divergence. *Neural computation*, 14(8):1771–1800, 2002.

[Hoc98] Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02):107–116, 1998.

[HOT06] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[Inc15] The MathWorks Inc. R2015a. Available online, 2015.

[JSA14] Majid Janzamin, Hanie Sedghi, and Anima Anandkumar. Score function features for discriminative learning: Matrix and tensor framework. *arXiv preprint arXiv:1412.2863*, 2014.

[JSA15] Majid Janzamin, Hanie Sedghi, and Anima Anandkumar. Beating the perils of non-convexity: Guaranteed training of neural networks using tensor methods. *arXiv preprint arXiv:1506.08473*, 2015.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[Lar] Hugo Larochelle. Restricted boltzmann machine.

[SFG$^+$17] Bharath Sriperumbudur, Kenji Fukumizu, Arthur Gretton, Aapo Hyvärinen, and Revant Kumar. Density estimation in infinite dimensional exponential families. *Journal of Machine Learning Research*, 18(57):1–59, 2017.

[SHM$^+$16] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[Ším02] Jiří Šíma. Training a single sigmoidal neuron is hard. *Neural Computation*, 14(11):2709–2728, 2002.

[SKRP15] George Saon, Hong-Kwang J Kuo, Steven Rennie, and Michael Picheny. The ibm 2015 english conversational telephone speech recognition system. *arXiv preprint arXiv:1505.05899*, 2015.

[SLMN11] Richard Socher, Cliff C Lin, Chris Manning, and Andrew Y Ng. Parsing natural scenes and natural language with recursive neural networks. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 129–136, 2011.