An Attempt to Escape the Deep Saddle Points

Ankit Goyal
Department of Electrical Engineering
Indian Institute of Technology Kanpur
Kanpur, India
ankgoyal@iitk.ac.in

Md Enayat Ullah
Department of Mathematics and Statistics
Indian Institute of Technology Kanpur
Kanpur, India
enayat@iitk.ac.in

Abstract

Gradient Descent style algorithms like Stochastic Gradient Descent (SGD) are popular convex optimization techniques, which have been widely applied on various non-convex optimization problems, including the ones encountered while training Deep Neural Networks (DNN). In spite of their empirical success, there exists limited understanding about the theoretical applicability of SGD in non-convex settings. Recent works have shown that the crucial challenge in high-dimensional non-convex optimization problems is averting convergence at saddle points and spurious local minima. We propose to build upon the recent results, which guarantee a gradient descent style algorithm to escape saddle points, if the optimization problem exhibits the strict-saddle property. In particular, our intention is to extend the analysis for Tensor Decomposition to two-layered Neural Networks (NN), as both these non-convex problems suffer from escalation of saddle points in high-dimensional settings, owing to their symmetry. We particularly build upon the work on training neural networks as a tensor decomposition problem.

1 Introduction

The idea of biologically inspired artificial neural networks existed since the 1980's [1], however their true potential was realized with the advent of computational power and large-scale data. Recently, Deep Neural Networks (DNN) have emerged as a hammer cracking down a plethora of machine learning problems [2, 3]. But despite their empirical success, there have been limited studies concerning their theoretical properties.

In general, the objective function for training a DNN is non-convex. However, initially researchers used off-the-shelf convex optimization techniques like Stochastic Gradient Descent (SGD) and L-BFGS [4], for training DNNs. In particular, SGD has been extensively studied in convex settings [5], but its effectiveness in non-convex settings is still not completely understood. Dauphin et al. in [6] showed that for non-convex optimization in high-dimensional settings, it is the proliferation of saddle-points, and not local minima that pose problem for gradient descent based methods. They also argued that in such high-dimensional settings, it is sufficient to converge to a local minimum, as the value of objective function at a local minimum would be very close to its value at the global minimum. Following these lines, they suggested a gradient descent based method for non-convex optimization, that uses second-order Hessian information, to avoid avoid/escape the saddle point. Furthermore, Ge et al. in [7] identified a strict-saddle property which allowed them to use SGD for escaping saddle points even without using the second-order information, and thereby avoiding the computational and memory overhead. They illustrated the effectiveness of this approach for Tensor Decomposition, which is a widely used non-convex optimization problem. They showed how objective function of Orthogonal Tensor Decomposition can be reformulated so as to obey the strict-saddle property, and thereby proposed a framework for Orthogonal Tensor Decomposition.

There have always been attempts to study non-convex problems, by identifying properties similar to the convex ones. Recent works have analyzed non-convex optimization problems with added constraints like Restricted Strong Convexity (RSC) [8]. However, since these constraints ensure that there is a unique stationary point [7], the analysis of these

problems cannot be transferred to high-dimensional DNN cost function optimization, as here we have a mushrooming of saddle points. Tensor Decomposition problem exhibits a symmetry in its solution, whereby if $(v_1, v_2, ..., v_d)$ is a solution, then for any permutation π and any sign flips $\kappa \in \{\pm 1\}^d$, $(.., \kappa_i v_{\pi(i)}, ...)$ is also a valid solution. Such a symmetry results in escalation of saddle points. Neural Networks (NN) exhibit a similar symmetry in its weight space, and thereby suffers from an excess of saddle points in its objective function. Saxe et al. in [9] show how scaling symmetries in Deep linear MLP give rise to saddle points. At the same time, DNN architecture serve as a natural extension to simple NN. The above factors serve as motivation for analyzing NN using the framework proposed by Ga et al. [7].

We initially intended to analyze the existing popular objectives for training neural networks, or if need to reformulate it, so that it falls in the paradigm of strict-saddle functions. However a much direct approach is to leverage the analysis of strict saddle objectives for tensor decomposition and apply it in the context of neural networks. Tensor methods are being widely studied applied to various latent variable and discriminative models[10]. Long story short, we formulate the neural network training as a tensor decomposition problem. This approach also naturally extends Ge et al's theoretical guarantees on the saddle point aversion to the neural network context. Moreover, Sedghi et al in [11] showed how generalized linear models can be trained using tensor decomposition methods. Janzamin et al [12] proposed algorithm based on tensor decomposition for guaranteed training of two-layer feed forward neural networks. We build upon their work in the context of neural networks, and leverage Ge et al's strict saddle guarantees to obtain good results for this highly non-convex problem.

2 Mathematical Preliminaries

2.1 Stochastic Gradient Descent (SGD)

Stochastic gradient descent is a gradient descent based optimization routine which solves the following problem:

$$w = \arg\min_{w \in \mathbb{R}^d} f(w), \text{ where } f(w) = \mathbb{E}_{x \sim \mathcal{D}}[\phi(w, x)] \tag{1}$$

The data point x is drawn from some unknown distribution $\mathcal{D}$, and ϕ is a loss function that is defined for a pair (x, w). The aim is to minimize the expected loss $\mathbb{E}[\phi(w, x)]$. The parameter updation takes place following a stochastic gradient

$$w_{t+1} = w_t - \eta \nabla_{w_t} \phi(w_t, x_t), \tag{2}$$

where x_t is a random sample drawn from distribution $\mathcal{D}$ and η is the learning rate.

Algorithm 1 is a slightly modified SGD, presented by Ge et al. It essentially adds a uniform noise to the gradient descent step, so as to drive the updates in the direction to escape saddle points.

Algorithm 1 Noisy Stochastic Gradient

Input: SG oracle $SG(w)$, initial point w_0, desired accuracy κ.
Output: w_t that is close to some local minimum $w^\star$.
1: Choose $\eta = \min\{\tilde{O}(\kappa^2 / \log(1/\kappa)), \eta_{\max}\}$, $T = \tilde{O}(1/\eta^2)$
2: **for** $t = 0$ to $T - 1$ **do**
3: Sample noise n uniformly from unit sphere.
4: $w_{t+1} \leftarrow w_t - \eta(SG(w) + n)$
5: **end for**

2.2 Strict-Saddle Property

Given a twice differentiable function $f(w)$, we can investigate the nature of stationary points using by looking at the hessian $\nabla^2 f(w)$. If $\nabla^2 f(w)$ is positive definite then w is a local minimum; if $\nabla^2 f(w)$ is negative definite then w is a local maximum; if $\nabla^2 f(w)$ has both positive and negative eigenvalues then w is a saddle point. However, there is another case where $\nabla^2 f(w)$ can be positive semidefinite with an eigenvalue equal to 0, in which case the point could be a local minimum or a saddle point. We give the definitions of strict saddle functions based on this intuition.

Definition 1. A twice differentiable function $f(w)$ is strict-saddle, if all its local minima have $\nabla^2 f(w) \succ 0$ and all its other stationary points satisfy $\lambda_{min}(\nabla^2 f(w)) < 0$.

Definition 2. A twice differentiable function $f(w)$ is $(\alpha, \gamma, \epsilon, \delta)$-strict saddle, if for any point w at least one of the following is true

1. $\|\nabla f(w)\| \geq \epsilon$.
2. $\lambda_{min}(\nabla^2 f(w)) \leq -\gamma$.
3. There is a local minimum $w^\star$ such that $\|w - w^\star\| \leq \delta$, and the function $f(w')$ restricted to 2δ neighborhood of $w^\star$ ($\|w' - w^\star\| \leq 2\delta$) is α-strongly convex.

2.3 Tensor Decomposition

A tensor is a natural generalization of scalars and vectors to arbitrary dimensions. Tensors can be constructed from tensor products. $(u \otimes v)$ denotes a second order tensor where $(u \otimes v)_{i,j} = u_i v_j$. This generalizes to higher order and $u^{\otimes 4}$ denotes the 4-th order tensor

$$[u^{\otimes 4}]_{i_1,i_2,i_3,i_4} = u_{i_1} u_{i_2} u_{i_3} u_{i_4}.$$

A 4-th order tensor $T \in \mathbb{R}^{d^4}$ has an orthogonal decomposition if it can be written as

$$T = \sum_{i=1}^{d} a_i^{\otimes 4}, \tag{3}$$

where a_i's are orthonormal vectors that satisfy $\|a_i\| = 1$ and $a_i^T a_j = 0$ for $i \neq j$. Vectors a_i's are called the components of this decomposition. A tensor also defines a multilinear form (just as a matrix defines a bilinear form), for a p-th order tensor $T \in \mathbb{R}^{d^p}$ and matrices $M_i \in \mathbb{R}^{d \times n_i} i \in [p]$, we define

$$[T(M_1, M_2, ..., M_p)]_{i_1,i_2,...,i_p} = \sum_{j_1,j_2,...,j_p \in [d]} T_{j_1,j_2,...,j_p} \prod_{t \in [p]} M_t[i_t, j_t]. \tag{4}$$

The above equation defines a mapping corresponding to a fixed tensor, which takes in a bunch of matrices and outputs a tensor (or a matrix, vector or scalar). The multi-linear form is used heavily to devise objective functions corresponding to tensor method problems,

Orthogonal Tensor decomposition is a non-convex optimization problem where the objective function is generally formulated as follows:

$$\min_{\forall i, \|u_i\|^2 = 1} \quad \|T - \sum_{i=1}^{d} u_i^{\otimes 4}\|_F^2. \tag{5}$$

Ge et al. [7] proposed an alternative objective function because they were unable to prove strict-saddle property for the objective in Eq. 5. The following objective function for Tensor Decomposition was proved to satisfy strict-saddle property.

$$\min_{\forall i, \|u_i\|^2 = 1} \quad \sum_{i \neq j} T(u_i, u_i, u_j, u_j), \tag{6}$$

We, however, will use the following objective, since it is directly related to our problem. We will also give a proof that the objective 12 satisfies the strict-saddle property.

$$\min_{\|u\|^2 = 1} T(u, u, u), \tag{7}$$

2.4 Score Function

Score functions are general-purpose tensor valued features which can be pre-trained using unlabeled samples. The higher order score functions capture local variations in the probability density function of the input.

$$\mathcal{S}_m(x) := (-1)^m \frac{\nabla_x^{(m)} p(x)}{p(x)}, \tag{8}$$

where $p(x)$ is the probability density function of random vector $x \in \mathbb{R}^d$.

Theorem 3. *For random vector $x \in \mathbb{R}^d$, let $p(x)$ and $\mathcal{S}_m(x)$ respectively denote the pdf and the corresponding m-th order score function. Consider any continuously differentiable output-function $\mathbb{E}[y|x] = g(x) : \mathbb{R}^d \to \mathbb{R}$ satisfying some mild regularity conditions. Then we have*

$$\mathbb{E}\left[y \cdot \mathcal{S}_m(x)\right] = \mathbb{E}\left[g(x) \cdot \mathcal{S}_m(x)\right] = \mathbb{E}\left[\nabla_x^{(m)} g(x)\right].$$

For proof please refer to [13].
Remark: Various methods for estimating the score functions exists. Assuming the knowledge of data generating probability distribution function, we have used the analytic formulation of the score function. In cases where it is not suitable to make the assumption, score function matching techniques, as described in [14, 15] can be used for finding parameters of probabilistic models

3 Approach

3.1 Neural Network Architecture

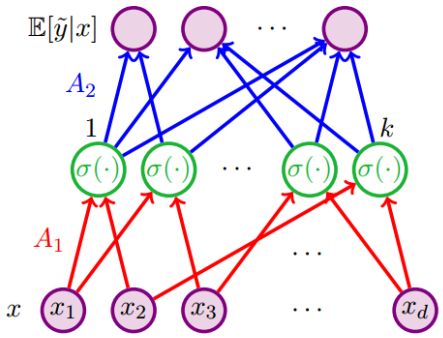

Figure 1: A two-layer feed forward neural network

We consider the simplest neural network setting: a two-layer feed forward NN wherein each layer is fully connected with the adjacent layer as shown in Fig. 1. The input features are fed to the first layer. Each node in a neural net receives a set of inputs from other connected nodes, and outputs a value which is a non-linear(sigmoid, in our case) function applied on the weighted sum of the inputs. The weights are updated so as to minimize the empirical loss on the training data.

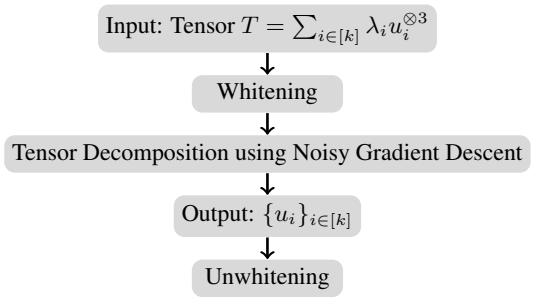

Figure 2: Overview of tensor decomposition algorithm .

Algorithm 2 Whitening

input Tensor $T \in \mathbb{R}^{d \times d \times d}$.

1: Second order moment $M_2 \in \mathbb{R}^{d \times d}$ is constructed such that it has the same decompositon form as target tensor T
2: Compute the rank-k SVD, $M_2 = U \operatorname{Diag}(\gamma) U^\top$, where $U \in \mathbb{R}^{d \times k}$ and $\gamma \in \mathbb{R}^k$.
3: Compute the whitening matrix $W := U \operatorname{Diag}(\gamma^{-1/2}) \in \mathbb{R}^{d \times k}$.
4: **return** $T(W, W, W) \in \mathbb{R}^{k \times k \times k}$.

Procedure 3 Un-whitening

input Orthogonal rank-1 components $v_j \in \mathbb{R}^k, j \in [k]$.

1: Consider matrix M_2 which was exploited for whitening in Procedure 2, and let $\tilde{\lambda}_j, j \in [k]$ denote the corresponding coefficients as $M_2 = A_1 \operatorname{Diag}(\tilde{\lambda}) A_1^\top$;
2: Compute the rank-k SVD, $M_2 = U \operatorname{Diag}(\gamma) U^\top$, where $U \in \mathbb{R}^{d \times k}$ and $\gamma \in \mathbb{R}^k$.
3: Compute

$$(A_1)_j = \frac{1}{\sqrt{\tilde{\lambda}_j}} U \operatorname{Diag}(\gamma^{1/2}) v_j, \quad j \in [k].$$

4: **return** $\{(A_1)_j\}_{j \in [k]}$.

3.2 Tensor Decomposition Formulation

Essentially, the broader aim to tweak the neural network training procedure so as to provide saddle point escaping guarantees. We commence with our goal of formulating a neural network training problem as a tensor decomposition optimization problem. We draw inspiration from the work of Janzamin et al[12], where they examined the neural network shown in Fig. 1, and proposed an algorithm to learn the weights of both the layers using tensor methods. The corresponding label function of the shown neural network is given by

$$\tilde{f}(x) := \mathbb{E}[\tilde{y}|x] = \langle a_2, \sigma(A_1^\top x + b_1) \rangle + b_2, \tag{9}$$

Janzamin et al showed that Tensor Decomposition of the third order expected cross moment gives the weight of the first layer (upto a scale) as in Eq. (10), and once A_1 is estimated, A_2 is determined by ridge-regression. The full algorithm to recover the weights of both the layers is presented in algorithm 4.

$$M_3 = \mathbb{E}\left[\tilde{y} \cdot \mathcal{S}_3(x)\right] = \sum_{j \in [k]} \lambda_j \cdot (A_1)_j \otimes (A_1)_j \otimes (A_1)_j \in \mathbb{R}^{d \times d \times d}, \tag{10}$$

For proof, see [12]. An important point to note is that the formulation works for any m^{th} order score function. For example:

$$M_2 = \mathbb{E}\left[\tilde{y} \cdot \mathcal{S}_2(x)\right] = \sum_{j \in [k]} \lambda_j \cdot (A_1)_j \otimes (A_1)_j \in \mathbb{R}^{d \times d}, \tag{11}$$

However, we do not consider a second-order tensor (matrix) decomposition as being a matrix, it can only identify the weights upto a subspace. Higher order tensors alleviate this problem, but any increase in the order of the tensor significantly increases the computational complexity. We therefore stick with the third order tensor M_3.

Procedure 4 NN-LIFT (Neural Network LearnIng using Feature Tensors)

Input: Labeled samples $\{(x_i, y_i) : i \in [n]\}$, parameter $\tilde{\epsilon}_1$, parameter λ.
Input: Third order score function $\mathcal{S}_3(x)$ of the input x

1: Compute $\widehat{T} := \frac{1}{n} \sum_{i \in [n]} y_i \cdot \mathcal{S}_3(x_i)$.
2: $\{(\hat{A}_1)_j\}_{j \in [k]} =$ tensor decomposition$(\widehat{T})$
3: $\hat{b}_1 =$ Fourier method$(\{(x_i, y_i) : i \in [n]\}, \hat{A}_1, \tilde{\epsilon}_1)$
4: $(\hat{a}_2, \hat{b}_2) =$ Ridge regression$(\{(x_i, y_i) : i \in [n]\}, \hat{A}_1, \hat{b}_1, \lambda)$

3.3 Pre and Post-Processing

We started with the intention of solving the Eq. 10 tensor decomposition problem to recover the weights via the objective Eq.12. However a crucial problem we face here is that Eq. 10 is not an orthogonal tensor decomposition, and a direct application of the strict-saddle satisfying objectives is infeasible. To tackle the problem, we first orthogonalize the tensor via a whitening procedure and then apply decomposition algorithm. The whitening procedure is described in algorithm 2. We also have an unwhitening procedure, following the orthogonal tensor decomposition of the whitened tensor. The Unwhitening procedure algorithm 3 finally give us the components of the original tensor. Figure 3 presents an low-level schematic of the pre and post-processing involved around the tensor decomposition.

There is however, a practical loophole to the unwhitening procedure. Line 1 of algorithm 3 tells how to obtain λ_j's corresponding to each component of the tensor decomposition. The procedure Janzamin et al provide involves plugging in weights A_1, which is a parameter we are essentially trying to estimate. Janzamin et al nowhere discuss an alternative ad-hoc strategy to estimate these λ_j's. For the sake of completeness, we leveraged on the knowledge of A_1 to appropriately set values of λ_j's, but that is completely infeasible in realistic settings.

3.4 Tensor Decomposition

Once the whitened tensor is obtained, we now focus on its orthogonal decomposition. In the original NN-LIFT Algorithm 4, Janzamin et al used the Tensor Power method to decompose the whitened tensor. This is the part where we make a detour. We formulate the orthogonal tensor decomposition as an optimization problem using a strict saddle objective, and then use a gradient descent solver to recover the weights. The details of our tensor decomposition algorithm is presented in algorithm 5. We also provide here the original robust tensor power method in algorithm 6 to clearly highlight the differences we make with respect to it.

We restate the objective function here:

$$\min_{\|u\|^2=1} T(u,u,u), \tag{12}$$

The objective expanded via the multilinear map gives:

$$T(u,u,u) := \sum_{j_1\in[q_1]} \sum_{j_2\in[q_2]} \sum_{j_3\in[q_3]} T_{j_1,j_2,j_3} \cdot u(j_1) \otimes u(j_2) \otimes u(j_3). \tag{13}$$

Procedure 5 Tensor Decomposition Using Noisy Gradient Descent

Input: Symmetric tensor $\tilde{T} \in \mathbb{R}^{d'\times d'\times d'}$, number of iterations N
Output: the estimated eigenvector/eigenvalue pair; the deflated tensor.

1: Compute $\widehat{v}$ using by using Noisy Gradient Descent on the following objective function

$$\max_{\|u\|^2=1} \quad \tilde{T}(u,u,u), \tag{14}$$

2: Set $\hat{\mu} := \tilde{T}(\widehat{v},\widehat{v},\widehat{v})$.
3: **return** the estimated eigenvector/eigenvalue pair $(\widehat{v},\hat{\mu})$; the deflated tensor $\tilde{T} - \hat{\mu}\cdot\hat{v}^{\otimes 3}$.

The tensor power methods uses a set of initializations and solve the decomposition problem using power iteration updates with respect to each of them, to finally obtain a good solution to the non-convex problem. We however solve the decomposition directly using a gradient descent style algorithm, with Eq. 12 as the objective. The theoretical guarantees provided by the strict saddle property prevents arbitrary convergence at saddle points. It thus converges to a local minima and we fairly assume that it a good solution to this highly non-convex problem [6].

3.5 Strict-Saddle Proof

We now give the proof below that the formulated objective 12 is indeed $(\alpha,\gamma,,\delta)$ strict-saddle.

Algorithm 6 Robust tensor power method [10]

input symmetric tensor $\tilde{T} \in \mathbb{R}^{d' \times d' \times d'}$, number of iterations N, number of initializations R.
output the estimated eigenvector/eigenvalue pair; the deflated tensor.

1: **for** $\tau = 1$ to R **do**
2: Initialize $\widehat{v}_0^{(\tau)}$ with SVD-based method.
3: **for** $t = 1$ to N **do**
4: Compute power iteration update

$$\widehat{v}_t^{(\tau)} := \frac{\tilde{T}(I, \widehat{v}_{t-1}^{(\tau)}, \widehat{v}_{t-1}^{(\tau)})}{\|\tilde{T}(I, \widehat{v}_{t-1}^{(\tau)}, \widehat{v}_{t-1}^{(\tau)})\|} \tag{15}$$

5: **end for**
6: **end for**
7: Let $\tau^* := \arg\max_{\tau \in [R]} \{\tilde{T}(\widehat{v}_N^{(\tau)}, \widehat{v}_N^{(\tau)}, \widehat{v}_N^{(\tau)})\}$.
8: Do N power iteration updates (15) starting from $\widehat{v}_N^{(\tau^*)}$ to obtain $\widehat{v}$, and set $\hat{\mu} := \tilde{T}(\widehat{v}, \widehat{v}, \widehat{v})$.
9: **return** the estimated eigenvector/eigenvalue pair $(\widehat{v}, \hat{\mu})$; the deflated tensor $\tilde{T} - \hat{\mu} \cdot \hat{v}^{\otimes 3}$.

Note: The proof draws heavily from [7]. We have used results obtained by Ge et al. to drive the proof. The reader is encouraged to refer to [7] to thoroughly comprehend it.

We restate the optimization problem

$$\begin{aligned} \max \quad & T(u, u, u), \\ & \|u\|^2 = 1. \end{aligned} \tag{16}$$

Here the tensor T has orthogonal decomposition $T = \sum_{i=1}^{d} a_i^{\otimes 3}$. We change the coordinate system to what is specified by (a_i)'s . In particular, let $u = \sum_{i=1}^{d} x_i a_i$ (where $x \in \mathbb{R}^d$), then we can see $T(u, u, u, u) = \sum_{i=1}^{d} x_i^3$. Therefore let $f(x) = -\|x\|_3^3$. This gives us the following equivalent optimization problem

$$\begin{aligned} \min \quad & f(x) \\ \text{s.t.} \quad & \|x\|_2^2 = 1 \end{aligned} \tag{17}$$

This is a constrained optimization. Let $c(x) = \|x\|_2^2 - 1$. We first compute the Lagrangian

$$\mathcal{L}(x, \lambda) = f(x) - \lambda c(x) = -\|x\|_3^3 - \lambda(\|x\|_2^2 - 1). \tag{18}$$

We have only one constraint and the gradient when $\|x\| = 1$ always have norm 2, so the set of constraints satisfy 2-RLICQ (refer [7]). In particular, we can compute the correct value of Lagrangian multiplier λ,

$$\lambda^*(x) = \arg\min_{\lambda} \|\nabla_x \mathcal{L}(x, \lambda)\| = \arg\min_{\lambda} \sum_{i=1}^{d} (3x_i^2 + 2\lambda x_i)^2 = -\frac{3}{2}\|x\|_3^3 \tag{19}$$

Therefore, the gradient in the tangent space is equal to

$$\begin{aligned} \chi(x) &= \nabla_x \mathcal{L}(x, \lambda)|_{(x, \lambda^*(x))} = \nabla f(x) - \lambda^*(x) \nabla c(x) \\ &= -3(x_1^2, \cdots, x_d^2)^T - 2\lambda^*(x)(x_1, \cdots, x_d)^T \\ &= 3\left((x_1 - \|x\|_3^3)x_1, \cdots, (x_d - \|x\|_3^3)x_d\right) \end{aligned} \tag{20}$$

The second-order partial derivative of Lagrangian is equal to

$$\begin{aligned} \mathfrak{M}(x) &= \nabla_{xx}^2 \mathcal{L}(x, \lambda)|_{(x, \lambda^*(x))} = \nabla^2 f(x) - \lambda^*(x) \nabla^2 c(x) \\ &= -6\text{diag}(x_1, \cdots, x_d) - 2\lambda^*(x) I_d \\ &= -6\text{diag}(x_1, \cdots, x_d) + 3\|x\|_3^3 I_d \end{aligned} \tag{21}$$

Since the variable x has bounded norm, and the function is a polynomial, it's clear that the function itself is bounded and all its derivatives are bounded. Moreover, all the derivatives of the constraint are bounded. The following lemma summarizes it.

Lemma 4. *The objective function (12) is bounded by* 1*, its* p*-th order derivative is bounded by* $O(\sqrt{d})$ *for* $p = 1, 2, 3$*. The constraint's* p*-th order derivative is bounded by* 2*, for* $p = 1, 2, 3$*.*

Therefore the function satisfies all the smoothness condition we need. Finally we show the gradient and Hessian of Lagrangian satisfy the $(\alpha, \gamma, \epsilon, \delta)$-strict saddle property. Note that we did not try to optimize the dependency with respect to d.

Theorem 5. *The only local minima of optimization problem (12) are* $\pm a_i$ $(i \in [d])$*. Further it satisfy* $(\alpha, \gamma, \epsilon, \delta)$*-strict saddle for* $\gamma = 7/d$*,* $\alpha = 3$ *and* $\epsilon, \delta = 1/poly(d)$*.*

In order to prove this theorem, we consider the transformed version Eq.17. We first need following two lemma for points around saddle point and local minimum respectively. We choose

$$\epsilon_0 = (10d)^{-4}, \;\; \epsilon = 4\epsilon_0^2, \;\; \delta = 2d\epsilon_0, \;\; \mathfrak{S}(x) = \{i \mid |x_i| > \epsilon_0\} \tag{22}$$

Where by intuition, $\mathfrak{S}(x)$ is the set of coordinates whose value is relative large.

Lemma 6. *Under the choice of parameters in Eq.(22), suppose* $\|\chi(x)\| \le \epsilon$*, and* $|\mathfrak{S}(x)| \ge 2$*. Then, there exists* $\hat{v} \in \mathcal{T}(x)$ *and* $\|\hat{v}\| = 1$*, so that* $\hat{v}^T \mathfrak{M}(x) \hat{v} \le -7/d$*.*

Proof. Suppose $|\mathfrak{S}(x)| = p$, and $2 \le p \le d$. Since $\|\chi(x)\| \le \epsilon = 4\epsilon_0^2$, by Eq.(20), we have for each $i \in [d]$, $|[\chi(x)]_i| = 4|(x_i^2 - \|x\|_4^4)x_i| \le 4\epsilon_0^2$. Therefore, we have:

$$\forall i \in \mathfrak{S}(x), \qquad |x_i - \|x\|_3^3| \le \epsilon_0 \tag{23}$$

and thus:

$$\begin{aligned} &|\|x\|_3^3 - \frac{1}{p}| = |\|x\|_3^3 - \frac{1}{p}\sum_i x_i^2| \\ \le &|\|x\|_3^- \frac{1}{p}\sum_{i \in \mathfrak{S}(x)} x_i^2| + |\frac{1}{p}\sum_{i \in [d] - \mathfrak{S}(x)} x_i^2| \le \epsilon_0 + \frac{d-p}{p}\epsilon_0^2 \le 2\epsilon_0 \end{aligned} \tag{24}$$

Combined with Eq.23, this means:

$$\forall i \in \mathfrak{S}(x), \qquad |x_i - \frac{1}{p}| \le 3\epsilon_0 \tag{25}$$

Because of symmetry, WLOG we assume $\mathfrak{S}(x) = \{1, \cdots, p\}$. Since $|\mathfrak{S}(x)| \ge 2$, we can pick $\hat{v} = (a, b, 0, \cdots, 0)$. Here $a > 0, b < 0$, and $a^2 + b^2 = 1$. We pick a such that $ax_1 + bx_2 = 0$. The solution is the intersection of a radius 1 circle and a line which passes $(0, 0)$, which always exists. For this $\hat{v}$, we know $\|\hat{v}\| = 1$, and $\hat{v}^T x = 0$ thus $\hat{v} \in \mathcal{T}(x)$. We have:

$$\begin{aligned} &\hat{v}^T \mathfrak{M}(x) \hat{v} = -(6x_1^2 + 3\|x\|_3^3)a^2 - (6x_2^2 + 3\|x\|_3^3)b^2 \\ = &- 3x_1^2 a^2 - 3x_2^2 b^2 - 3(x_1 - \|x\|_3^3))a^2 - 3(x_2 - \|x\|_3^3))b^2 \\ \le &- \frac{3}{p} + 3\epsilon_0 \le -7/d \end{aligned} \tag{26}$$

Which finishes the proof. □

Lemma 7. *Under the choice of parameters in Eq.(22), suppose* $\|\chi(x)\| \le \epsilon$*, and* $|\mathfrak{S}(x)| = 1$*. Then, there is a local minimum* $x^\star$ *such that* $\|x - x^\star\| \le \delta$*, and for all* x' *in the* 2δ *neighborhood of* $x^\star$*, we have* $\hat{v}^T \mathfrak{M}(x') \hat{v} \ge 3$ *for all* $\hat{v} \in \mathcal{T}(x')$*,* $\|\hat{v}\| = 1$

Proof. WLOG, we assume $\mathfrak{S}(x) = \{1\}$. Then, we immediately have for all $i > 1$, $|x_i| \le \epsilon_0$, and thus:

$$1 \ge x_1^2 = 1 - \sum_{i>1} x_i^2 \ge 1 - d\epsilon_0^2 \tag{27}$$

Therefore $x_1 \geq \sqrt{1-d\epsilon_0^2}$ or $x_1 \leq -\sqrt{1-d\epsilon_0^2}$. Which means x_1 is either close to 1 or close to -1. By symmetry, we know WLOG, we can assume the case $x_1 \geq \sqrt{1-d\epsilon_0^2}$. Let $e_1 = (1, 0, \cdots, 0)$, then we know:

$$\|x - e_1\|^2 \leq (x_1 - 1)^2 + \sum_{i>1} x_i^2 \leq 2d\epsilon_0^2 \leq \delta^2 \tag{28}$$

Next, we show e_1 is a local minimum. According to Eq.21, we know $\mathfrak{M}(e_1)$ is a diagonal matrix with 4 on the diagonals except for the first diagonal entry (which is equal to -8), since $\mathcal{T}(e_1) = \text{span}\{e_2, \cdots, e_d\}$, we have:

$$v^T \mathfrak{M}(e_1) v \geq 4\|v\|^2 > 0 \qquad \text{for all } v \in \mathcal{T}(e_1), v \neq 0 \tag{29}$$

Which by (Theorem 24 of [7]) means e_1 is a local minimum.

Finally, denote $\mathcal{T}_1 = \mathcal{T}(e_1)$ be the tangent space of constraint manifold at e_1. We know for all x' in the 2δ neighborhood of e_1, and for all $\hat{v} \in \mathcal{T}(x')$, $\|\hat{v}\| = 1$:

$$\begin{aligned}
\hat{v}^T \mathfrak{M}(x') \hat{v} \geq & \hat{v}^T \mathfrak{M}(e_1) \hat{v} - |\hat{v}^T \mathfrak{M}(e_1) \hat{v} - \hat{v}^T \mathfrak{M}(x') \hat{v}| \\
= & 4\|P_{\mathcal{T}_1} \hat{v}\|^2 - 8\|P_{\mathcal{T}_1^c} \hat{v}\|^2 - \|\mathfrak{M}(e_1) - \mathfrak{M}(x')\| \|\hat{v}\|^2 \\
= & 4 - 12\|P_{\mathcal{T}_1^c} \hat{v}\|^2 - \|\mathfrak{M}(e_1) - \mathfrak{M}(x')\|
\end{aligned} \tag{30}$$

By lemma 26 [7], we know $\|P_{\mathcal{T}_1^c} \hat{v}\|^2 \leq \|x' - e_1\|^2 \leq 4\delta^2$. By Eq.(21), we have:

$$\begin{aligned}
& \|\mathfrak{M}(e_1) - \mathfrak{M}(x')\| \leq \|\mathfrak{M}(e_1) - \mathfrak{M}(x')\| \leq \sum_{(i,j)} |[\mathfrak{M}(e_1)]_{ij} - [\mathfrak{M}(x')]_{ij}| \\
\leq & \sum_i \left| -12[e_1]_i^2 + 4\|e_1\|_4^4 - 12x_i^2 + 4\|x\|_4^4 \right| \leq 64d\delta
\end{aligned} \tag{31}$$

In conclusion, we have $\hat{v}^T \mathfrak{M}(x') \hat{v} \geq 4 - 48\delta^2 - 64d\delta \geq 3$ which finishes the proof. □

Finally, we are ready to prove Theorem 5.

Proof of Theorem 5. According to Lemma 6 and Lemma 7, we immediately know the optimization problem satisfies $(\alpha, \gamma, \epsilon, \delta)$-strict saddle.

The only thing remains to show is that the only local minima of optimization problem (12) are $\pm a_i$ $(i \in [d])$. Which is equivalent to show that the only local minima of the transformed problem is $\pm e_i$ $(i \in [d])$, where $e_i = (0, \cdots, 0, 1, 0, \cdots, 0)$, where 1 is on i-th coordinate.

By investigating the proof of Lemma 6 and Lemma 7, we know these two lemmas actually hold for any small enough choice of ϵ_0 satisfying $\epsilon_0 \leq (10d)^{-4}$, by pushing $\epsilon_0 \to 0$, we know for any point satisfying $|\chi(x)| \leq \epsilon \to 0$, if it is close to some local minimum, it must satisfy $1 = |\mathfrak{S}(x)| \to \text{supp}(x)$. Therefore, we know the only possible local minima are $\pm e_i$ $(i \in [d])$. In Lemma 7, we proved e_1 is local minima, by symmetry, we finishes the proof. □

3.6 Estimating the other parameters

Once the weights A_1 are obtained, we adopt the fourier method presented by Janzamin et al to estimate the bias b_1. The weights and bias of the next layer (which does not involve a non-linear activation) are obtained by simple ridge regression. As there are hardly any differences between us and them in the procedure of estimating b_1, A_2, b_2 ,the details have been omitted. The reader is encouraged to refer to [12] to comprehend the intricacies .

4 Implementation

We empirically validate the hypothesis that the tensor decomposition framework for training neural networks is indeed able to recover the weights, and how it fairs against standard off-the-shelf backpropagation based neural network implementations.

4.1 Data Generation

We generate the data from a mixture of three Gaussians. For simplicity, we consider diagonal covariance matrices. This simplification considerably speeds up the calculation of score tensors. We further use pre-determined matrices for A_1 and A_2, and bias b_1 and b_2 to generate the corresponding labels.

The calculated second and third score function tensors are given below. The notation governing the following equations is that $p_m(x)$ is the pdf of the m^{th} mixture, μ_{mi} is the i^{th} element of the m^{th} mixture mean vector and σ^2_{mi} is the i^{th} diagonal element of the m^{th} mixture covariance matrix.

$$[\mathcal{S}_2(x)]_{ij} = \begin{cases} \sum_{m=1}^{n_m} \frac{\pi_m p_m(x)((x_i - \mu_{mi})^2 - \sigma_{mi})}{(\sigma^2_{mi})^2} & \text{for } i = j \\ \sum_{k=1}^{n_m} \frac{\pi_m p_m(x)((x_j - \mu_{kj})(x_i - \mu_{mi})}{\sigma^2_{mi}\sigma^2_{mj}} & \text{for } i \neq j \end{cases} \tag{32}$$

$$[\mathcal{S}_3(x)]_{ijk} = \begin{cases} \sum_{m=1}^{n_m} \frac{\pi_m p_m(x)(x_i - \mu_{mi})^2(3\sigma^2_{mi} - (x_i - \mu_{mi})^2}{(\sigma^2_{mi})^3} & \text{for } i = j = k \\ \sum_{m=1}^{n_m} \frac{\pi_m p_m(x)(x_k - \mu_{mk})^2(\sigma^2_{mi} - (x_i - \mu_{mi})^2}{(\sigma^2_{mi})^2\sigma^2_{mk}} & \text{for } i \neq j = k \\ \sum_{m=1}^{n_m} \frac{\pi_m p_m(x)(x_i - \mu_{mi})^2(\sigma^2_{mj} - (x_j - \mu_{mj})^2}{(\sigma^2_{mj})^2\sigma^2_{mi}} & \text{for } i = k \neq j \\ \sum_{m=1}^{n_m} -\frac{\pi_m p_m(x)(x_i - \mu_{mi})(x_j - \mu_{mj})(x_k - \mu_{mk})}{(\sigma^2_{mi})^2\sigma^2_{mj}\sigma^2_{mk}} & \text{for } i \neq j \neq k \end{cases} \tag{33}$$

4.2 Results

The comparative average error obtained for different dimensional feature vectors are tabulated in table 1.

Dimension	Average Error (Tensor Decomposition)	Average Error (Standard NN)
d=3	1.82	0.564
d=5	2.03	0.663
d=10	2.79	0.682
d=20	5.67	1.237

Table 1: Results

5 Conclusion

We investigated a completely non-conventional paradigm of training discriminative models. The work essentially amalgamates two key directions in large-scale machine learning: saddle-point escaping and tensor decomposition methods. We further theoretically and empirically, present a viable tensor decomposition based method for training neural networks. As reported earlier, we critique a fallacy presented in the tensor decomposition framework, which is the inability to estimate the λ_j's without using the NN parameters. The paper does not present experiments with the framework, thus have completely missed out on estimation of λ_j's from data. For our part, we used the knowledge of the weights to set the values of λ_j's appropriately.

There are a couple of directions to what we have achieved. One obvious extension is go from classical two-layered NN to encompass the deep learning paradigm in general. The tensor decomposition method is prone to good initializations, and so we can investigate if the results improve if the problem is initialized better.

References

[1] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, DTIC Document, 1985.

[2] 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.

[3] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *Signal Processing Magazine, IEEE*, 29(6):82–97, 2012.

[4] Jiquan Ngiam, Adam Coates, Ahbik Lahiri, Bobby Prochnow, Quoc V Le, and Andrew Y Ng. On optimization methods for deep learning. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 265–272, 2011.

[5] Alexander Rakhlin, Ohad Shamir, and Karthik Sridharan. Making gradient descent optimal for strongly convex stochastic optimization. *arXiv preprint arXiv:1109.5647*, 2011.

[6] Yann N Dauphin, Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, Surya Ganguli, and Yoshua Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In *Advances in Neural Information Processing Systems*, pages 2933–2941, 2014.

[7] Rong Ge, Furong Huang, Chi Jin, and Yang Yuan. Escaping from saddle points—online stochastic gradient for tensor decomposition. *arXiv preprint arXiv:1503.02101*, 2015.

[8] Alekh Agarwal, Sahand Negahban, and Martin J Wainwright. Fast global convergence rates of gradient methods for high-dimensional statistical recovery. In *Advances in Neural Information Processing Systems*, pages 37–45, 2010.

[9] Andrew M Saxe, James L McClelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv preprint arXiv:1312.6120*, 2013.

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

[11] Hanie Sedghi and Anima Anandkumar. Provable tensor methods for learning mixtures of generalized linear models. *arXiv preprint arXiv:1412.3046*, 2014.

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

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

[14] Aapo Hyvärinen. Estimation of non-normalized statistical models by score matching. In *Journal of Machine Learning Research*, pages 695–709, 2005.

[15] Kevin Swersky, David Buchman, Nando D Freitas, Benjamin M Marlin, et al. On autoencoders and score matching for energy based models. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 1201–1208, 2011.