Beating the Perils of Non-Convexity: Guaranteed Training of Neural Networks using Tensor Methods

Majid Janzamin[*] Hanie Sedghi[†] Anima Anandkumar[‡]

Abstract

Training neural networks is a challenging non-convex optimization problem, and backpropagation or gradient descent can get stuck in spurious local optima. We propose a novel algorithm based on tensor decomposition for guaranteed training of two-layer neural networks. We provide risk bounds for our proposed method, with a polynomial sample complexity in the relevant parameters, such as input dimension and number of neurons. While learning arbitrary target functions is NP-hard, we provide transparent conditions on the function and the input for learnability. Our training method is based on tensor decomposition, which provably converges to the global optimum, under a set of mild non-degeneracy conditions. It consists of simple embarrassingly parallel linear and multi-linear operations, and is competitive with standard stochastic gradient descent (SGD), in terms of computational complexity. Thus, we propose a computationally efficient method with guaranteed risk bounds for training neural networks with one hidden layer.

Keywords: neural networks, risk bound, method-of-moments, tensor decomposition

1 Introduction

Neural networks have revolutionized performance across multiple domains such as computer vision and speech recognition. They are flexible models trained to approximate any arbitrary target function, e.g., the label function for classification tasks. They are composed of multiple layers of *neurons* or *activating functions*, which are applied recursively on the input data, in order to predict the output. While neural networks have been extensively employed in practice, a complete theoretical understanding is currently lacking.

Training a neural network can be framed as an optimization problem, where the network parameters are chosen to minimize a given loss function, e.g., the *quadratic loss* function over the error in predicting the output. The performance of training algorithms is typically measured through the notion of *risk*, which is the expected loss function over unseen test data. A natural question to ask is the hardness of training a neural network with a bounded risk. The findings are mostly negative (Rojas, 1996; Šíma, 2002; Blum and Rivest, 1993; Bartlett and Ben-David, 1999; Kuhlmann, 2000). Training even a simple network is NP-hard, e.g., a network with a single neuron (Šíma, 2002).

[*]University of California, Irvine. Email: mjanzami@uci.edu
[†]University of California, Irvine. Email: sedghih@uci.edu
[‡]University of California, Irvine. Email: a.anandkumar@uci.edu

The computational hardness of training is due to the non-convexity of the loss function. In general, the loss function has many *critical points*, which include *spurious local optima* and *saddle points*. In addition, we face *curse of dimensionality*, and the number of critical points grows exponentially with the input dimension for general non-convex problems (Dauphin et al., 2014). Popular local search methods such as gradient descent or *backpropagation* can get stuck in bad local optima and experience arbitrarily slow convergence. Explicit examples of its failure and the presence of bad local optima in even simple separable settings have been documented before (Brady et al., 1989; Gori and Tesi, 1992; Frasconi et al., 1997); see Section 6.1 for a discussion.

Alternative methods for training neural networks have been mostly limited to specific activation functions (e.g., linear or quadratic), specific target functions (e.g., polynomials) (Andoni et al., 2014), or assume strong assumptions on the input (e.g., Gaussian or product distribution) (Andoni et al., 2014), see related work for details. Thus, up until now, there is no unified framework for training networks with general input, output and activation functions, for which we can provide guaranteed risk bound.

In this paper, for the first time, we present a guaranteed framework for learning general target functions using neural networks, and simultaneously overcome computational, statistical, and approximation challenges. In other words, our method has a low computational and sample complexity, even as the dimension of the optimization grows, and in addition, can also handle approximation errors, when the target function may not be generated by a given neural network. We prove a guaranteed risk bound for our proposed method. NP-hardness refers to the computational complexity of training worst-case instances. Instead, we provide transparent conditions on the target functions and the inputs for tractable learning.

Our training method is based on the method of moments, which involves decomposing the empirical cross moment between output and some function of input. While pairwise moments are represented using a matrix, higher order moments require tensors, and the learning problem can be formulated as tensor decomposition. A CP (CanDecomp/Parafac) decomposition of a tensor involves finding a succinct sum of rank-one components that best fit the input tensor. Even though it is a non-convex problem, the global optimum of tensor decomposition can be achieved using computationally efficient techniques, under a set of mild non-degeneracy conditions (Anandkumar et al., 2014a,b,c, 2015; Bhaskara et al., 2013). These methods have been recently employed for learning a wide range of latent variable models (Anandkumar et al., 2014b, 2013).

Incorporating tensor methods for training neural networks requires addressing a number of nontrivial questions: What form of moments are informative about network parameters? Earlier works using tensor methods for learning assume a linear relationship between the hidden and observed variables. However, neural networks possess non-linear activation functions. How do we adapt tensor methods for this setting? How do these methods behave in the presence of approximation and sample perturbations? How can we establish risk bounds? We address these questions shortly.

1.1 Summary of Results

The main contributions are: (a) we propose an efficient algorithm for training neural networks, termed as Neural Network-LearnIng using Feature Tensors (NN-LIFT), (b) we demonstrate that the method is embarrassingly parallel and is competitive with standard SGD in terms of computational complexity, and as a main result, (c) we establish that it has bounded risk, when the number of training samples scales polynomially in relevant parameters such as input dimension and number of neurons.

We analyze training of a two-layer feedforward neural network, where the second layer has a linear activation function. This is the classical neural network considered in a number of works (Cybenko, 1989b; Hornik, 1991; Barron, 1994), and a natural starting point for the analysis of any learning algorithm. Note that training even this two-layer network is non-convex, and finding a computationally efficient method with guaranteed risk bound has been an open problem up until now.

At a high level, NN-LIFT estimates the weights of the first layer using tensor CP decomposition. It then uses these estimates to learn the bias parameter of first layer using a simple Fourier technique, and finally estimates the parameters of last layer using linear regresion. NN-LIFT consists of simple linear and multi-linear operations (Anandkumar et al., 2014b,c, 2015), Fourier analysis and ridge regression analysis, which are parallelizable to large-scale data sets. The computational complexity is comparable to that of the standard SGD; in fact, the parallel time complexity for both the methods is in the same order, and our method requires more processors than SGD by a multiplicative factor that scales linearly in the input dimension.

Generative vs. discriminative models: Generative models incorporate a joint distribution $p(x,y)$ over both the input x and label y. On the other hand, discriminative models such as neural networks only incorporate the conditional distribution $p(y|x)$. While training neural networks for general input x is NP-hard, **does knowledge about the input distribution $p(x)$ make learning tractable?**

In this work, we assume knowledge of the input density $p(x)$, which can be any continuous differentiable function. Unlike many theoretical works, e.g., (Andoni et al., 2014), we do not limit ourselves to distributions such as product or Gaussian distributions for the input. While unsupervised learning, i.e., estimation of density $p(x)$, is itself a hard problem for general models, in this work, we investigate how $p(x)$ can be exploited to make training of neural networks tractable. The knowledge of $p(x)$ is naturally available in the *experimental design* framework, where the person designing the experiments has the ability to choose the input distribution. Examples include conducting polling, carrying out drug trials, collecting survey information, and so on.

Utilizing generative models on the input via score functions: We utilize the knowledge about the input density $p(x)$ (up to normalization)[1] to obtain certain (non-linear) transformations of the input, given by the class of score functions. *Score functions* are normalized derivatives of the input pdf; see (8). If the input is a vector (the typical case), the first order score function (i.e., the first derivative) is a vector, the second order score is a matrix, and the higher order scores are tensors. In our NN-LIFT method, we first estimate the cross-moments between the output and the input score functions, and then decompose it to rank-1 components.

Risk bounds: Risk bound includes both approximation and estimation errors. The approximation error is the error in fitting the target function to a neural network of given architecture, and the estimation error is the error in estimating the weights of that neural network using the given samples.

We first consider the *realizable setting* where the target function is generated by a two-layer neural network (with hidden layer of neurons consisting of any general sigmoidal activations), and

[1]We do not require the knowledge of the normalizing constant or the partition function, which is $\#P$ hard to compute (Wainwright and Jordan, 2008).

a linear output layer. Note that the approximation error is zero in this setting. Let $A_1 \in \mathbb{R}^{d\times k}$ be the weight matrix of first layer (connecting the input to the neurons) with k denoting the number of neurons and d denoting the input dimension. Suppose these weight vectors are non-degenerate, i.e., the weight matrix A_1 (or its tensorization) is full column rank. We assume continuous input distribution with access to score functions, which are bounded on any set of non-zero measure. We allow for any general sigmoidal activation functions with non-zero third derivatives in expectation, and satisfying Lipschitz property. Let $s_{\min}(\cdot)$ be the minimum singular value operator, and $M_3(x) \in \mathbb{R}^{d\times d^2}$ denote the matricization of input score function tensor $\mathcal{S}_3(x) \in \mathbb{R}^{d\times d\times d}$; see (1) and (8) for the definitions. For the Gaussian input $x \sim \mathcal{N}(0, I_d)$, we have $\mathbb{E}\left[\left\|M_3(x)M_3^\top(x)\right\|\right] = \tilde{O}\left(d^3\right)$. We have the following learning result in the realizable setting where the target function is generated by a two layer neural network (with one hidden layer).

Theorem 1 (Informal result for realizable setting)**.** *Our method NN-LIFT learns a realizable target function up to error ϵ when the number of samples is lower bounded as*[2]*,*

$$n \geq \tilde{O}\left(\frac{k}{\epsilon^2} \cdot \mathbb{E}\left[\left\|M_3(x)M_3^\top(x)\right\|\right] \cdot \frac{s_{\max}^2(A_1)}{s_{\min}^6(A_1)}\right).$$

Thus, we can efficiently learn the neural network parameters with polynomial sample complexity using NN-LIFT algorithm. In addition, the method has polynomial computational complexity, and in fact, its parallel time complexity is the same as stochastic gradient descent (SGD) or backpropagation. See Theorem 3 for the formal result.

We then extend our results to the *non-realizable setting* where the target function *need not be* generated by a neural network. For our method NN-LIFT to succeed, we require the approximation error to be sufficiently small under the given network architecture. Note that it is not of practical interest to consider functions with large approximation errors, since classification performance in that case is poor (Bartlett, 1998). We state the informal version of the result as follows.

We assume the following: the target function $f(x)$ has a continuous Fourier spectrum and is sufficiently smooth, i.e., the parameter C_f (see (13) for the definition) is sufficiently small as specified in (18). This implies that the approximation error of the target function can be controlled, i.e., there exists a neural network of given size that can fit the target function with bounded approximation error. Let the input x be bounded as $\|x\| \leq r$. Our informal result is as follows. See Theorem 5 for the formal result.

Theorem 2 (Informal result for non-realizable setting)**.** *The arbitrary target function $f(x)$ is approximated by the neural network $\hat{f}(x)$ which is learnt using NN-LIFT algorithm such that the risk bound satisfies w.h.p.*

$$\mathbb{E}_x[|f(x) - \hat{f}(x)|^2] \leq O(r^2 C_f^2) \cdot \left(\frac{1}{\sqrt{k}} + \delta_1\right)^2 + O(\epsilon^2),$$

where k is the number of neurons in the neural network, and δ_τ is defined in (16)*.*

In the above bound, we require for the target function $f(x)$ to have bounded first order moment in the Fourier spectrum; see (18). As an example, we show that this bound is satisfied for the class of scale and location mixtures of the Gaussian kernel function.

[2]Here, only the dominant terms in the sample complexity are noted; see (12) for the full details.

Corollary 1 (Learning mixtures of Gaussian kernels)**.** *Let $f(x) := \int K(\alpha(x+\beta))G(d\alpha, d\beta)$, $\alpha > 0$, $\beta \in \mathbb{R}^d$, be a location and scale mixture of the Gaussian kernel function $K(x) = \exp\left(-\frac{\|x\|^2}{2}\right)$, the input be Gaussian as $x \sim \mathcal{N}(0, \sigma_x^2 I_d)$, and the activations be step functions, then, our algorithm trains a neural network with risk bounds as in Theorem 2, when*

$$\int |\alpha| \cdot |G|(d\alpha, d\beta) \leq \text{poly}\left(\frac{1}{d}, \frac{1}{k}, \epsilon, \frac{1}{\sigma_x}, \exp\left(-1/\sigma_x^2\right)\right).$$

We observe that when the kernel mixtures correspond to smoother functions (smaller α), the above bound is more likely to be satisfied. This is intuitive since smoother functions have lower amount of high frequency content. Also, notice that the above bound has a dependence on the variance of the Gaussian input σ_x. We obtain the most relaxed bound (r.h.s. of above bound) for middle values of σ_x, i.e., when σ_x is neither too large nor too small. See Appendix D.1 for more discussion and the proof of the corollary.

Intuitions behind the conditions for the risk bound: Since there exist worst-case instances where learning is hard, it is natural to expect that NN-LIFT has guarantees only when certain conditions are met. We assume that the input has a regular continuous probability density function (pdf); see (11) for the details. This is a reasonable assumption, since under Boolean inputs (a special case of discrete input), it reduces to learning parity with noise which is a hard problem (Kearns and Vazirani, 1994). We assume that the activating functions are *sufficiently non-linear*, since if they are linear, then the network can be collapsed into a single layer (Baldi and Hornik, 1989), which is non-identifiable. We precisely characterize how the estimation error depends on the non-linearity of the activating function through its third order derivative.

Another condition for providing the risk bound is non-redundancy of the neurons. If the neurons are redundant, it is an over-specified network. In the realizable setting, where the target function is generated by a neural network with the given number of neurons k, we require (tensorizations of) the weights of first layer to be linearly independent. In the non-realizable setting, we require this to be satisfied by k vectors randomly drawn from the Fourier magnitude distribution (weighted by the norm of frequency vector) of the target function $f(x)$. More precisely, the random frequencies are drawn from probability distribution $\Lambda(\omega) := \|\omega\| \cdot |F(\omega)|/C_f$ where $F(\omega)$ is the Fourier transform of arbitrary function $f(x)$, and C_f is the normalization factor; see (26) and corresponding discussions for more details. This is a mild condition which holds when the distribution is continuous in some domain. Thus, our conditions for achieving bounded risk are mild and encompass a large class of target functions and input distributions.

Why tensors are required? We employ the cross-moment tensor which encodes the correlation between the third order score function and the output. We then decompose the moment tensor as a sum of rank-1 components to yield the weight vectors of the first layer. We require at least a third order tensor to learn the neural network weights for the following reasons: while a matrix decomposition is only identifiable up to orthogonal components, tensors can have identifiable non-orthogonal components. In general, it is not realistic to assume that the weight vectors are orthogonal, and hence, we require tensors to learn the weight vectors. Moreover, through tensors, we can learn overcomplete networks, where the number of hidden neurons can exceed the input/output dimensions. Note that matrix factorization methods are unable to learn overcomplete models, since the rank of the matrix cannot exceed its dimensions. Thus, it is critical to incorporate tensors for training neural networks. A recent set of papers have analyzed the tensor methods in detail, and established convergence and perturbation guarantees (Anandkumar et al., 2014a,b,c, 2015; Bhaskara et al., 2013), despite non-convexity of the decomposition problem. Such strong theoretical guarantees are essential for deriving provable risk bounds for NN-LIFT.

Extensions: Our algorithm NN-LIFT can be extended to more layers, by recursively estimating the weights layer by layer. In principle, our analysis can be extended by controlling the perturbation introduced due to layer-by-layer estimation. Establishing precise guarantees is an exciting open problem.

In this work, we assume knowledge of the generative model for the input. As argued before, in many settings such as experimental design or polling, the design of the input pdf $p(x)$ is under the control of the learner. Even if $p(x)$ is not known, a recent flurry of research activity has shown that a wide class of probabilistic models can be trained consistently using a suite of different efficient algorithms: convex relaxation methods (Chandrasekaran et al., 2010), spectral and tensor methods (Anandkumar et al., 2014b), alternating minimization (Agarwal et al., 2014), and they require only polynomial sample and computational complexity, with respect to the input and hidden dimensions. These methods can learn a rich class of models which also includes latent or hidden variable models.

Another aspect not addressed in this work is the issue of regularization for our NN-LIFT algorithm. In this work, we assume that the number of neurons is chosen appropriately to balance bias and variance through cross validation. Designing implicit regularization methods such as *dropout* (Hinton et al., 2012) or *early stopping* (Morgan and Bourlard, 1989) for tensor factorization and analyzing them rigorously is another exciting open research problem.

1.2 Related works

We first review some works regarding the analysis of backpropagation, and then provide some theoretical results on training neural networks.

Analysis of backpropagation and loss surface of optimization: Baldi and Hornik (1989) show that if the activations are linear, then backpropagation has a unique local optimum, and it corresponds to the principal components of the covariance matrix of the training examples. However, it is known that there exist networks with non-linear activations where backpropagation fails; for instance, Brady et al. (1989) construct simple cases of linearly separable classes that backpropagation fails. Note that the simple perceptron algorithm will succeed here due to linear separability. Gori and Tesi (1992) argue that such examples are artificial and that backpropagation succeeds in reaching the global optimum for linearly separable classes in practical settings. However, they show that under non-linear separability, backpropagation can get stuck in local optima. For a detailed survey, see Frasconi et al. (1997).

Recently, Choromanska et al. (2015) analyze the loss surface of a multi-layer ReLU network by relating it to a spin glass system. They make several assumptions such as variable independence for the input, equally likely paths from input to output, redundancy in network parameterization and uniform distribution for unique weights, which are far from realistic. Under these assumptions, the network reduces to a random *spin glass model*, where it is known that the lowest critical values of the random loss function form a layered structure, and the number of local minima outside that band diminishes exponentially with the network size (Auffinger et al., 2013). However, this does not imply computational efficiency: there is no guarantee that we can find such a good local optimal point using computationally cheap algorithms, since there are still exponential number of such points.

Haeffele and Vidal (2015) provide a general framework for characterizing when local optima become global in deep learning and other scenarios. The idea is that if the network is sufficiently overspecified (i.e., has enough hidden neurons) such that there exist local optima where some of the neurons have zero contribution, then such local optima are in fact, global. This provides a simple and a unified characterization of local optima which are global. However, in general, it is not clear how to design algorithms that can reach these efficient optimal points.

Previous theoretical works for training neural networks: Analysis of risk for neural networks is a classical problem. Approximation error of two layer neural network has been analyzed in a number of works (Cybenko, 1989b; Hornik, 1991; Barron, 1994). Barron (1994) provides a bound on the approximation error and combines it with the estimation error to obtain a risk bound, but for a computationally inefficient method. The sample complexity for neural networks have been extensively analyzed in Barron (1994); Bartlett (1998), assuming convergence to the globally optimal solution, which in general is intractable. See Anthony and Bartlett (2009); Shalev-Shwartz and Ben-David (2014) for an exposition of classical results on neural networks.

Andoni et al. (2014) learn polynomial target functions using a two-layer neural network under Gaussian/uniform input distribution. They argue that the weights for the first layer can be selected randomly, and only the second layer weights, which are linear, need to be fitted optimally. However, in practice, Gaussian/uniform distributions are never encountered in classification problems. For general distributions, random weights in the first layer is not sufficient. Under our framework, we impose only mild non-degeneracy conditions on the weights. Livni et al. (2014) make the observation that networks with quadratic activation functions can be trained in a computationally efficient manner in an incremental manner. This is because with quadratic activations, greedily adding one neuron at a time can be solved efficiently through eigen decomposition. However, the standard sigmoidal networks require a large depth polynomial network, which is not practical. After we posted the initial version of this paper, Zhang et al. (2015) extended this framework to improper learning scenario, where the output predictor need not be a neural network. They show that if the ℓ_1 norm of the incoming weights in each layer is bounded, then learning is efficient. However, for the usual neural networks with sigmoidal activations, the ℓ_1 norm of the weights scales with dimension and in this case, the algorithm is no longer polynomial time. Arora et al. (2013) provide bounds for leaning a class of deep representations. They use layer-wise learning where the neural network is learned layer-by-layer in an unsupervised manner. They assume sparse edges with random bounded weights, and 0/1 threshold functions in hidden nodes. The difference is here, we are considering the supervised setting where there is both input and output, and we allow for general sigmoidal functions at the hidden neurons.

Recently, after posting the initial version of this paper, Hardt et al. (2015) provided an analysis of stochastic gradient descent and its generalization error in convex and non-convex problems such as training neural networks. They show that the generalization error can be controlled under mild conditions. However, their work does not address about reaching a solution with small risk bound using SGD, and the SGD in general can get stuck in a spurious local optima. On the other hand, we show that in addition to having a small generalization error, our method yields a neural network with a small risk bound. Note that our method is moment-based estimation, and these methods come with stability bounds that guarantee good generalization error.

Closely related to this paper, Sedghi and Anandkumar (2014) consider learning neural networks with sparse connectivity. They employ the cross-moment between the (multi-class) label and (first order) score function of the input. They show that they can provably learn the weights of the first layer, as long as the weights are sparse enough, and there are enough number of input dimensions and output classes (at least linear up to log factor in the number of neurons in any layer). In this paper, we remove these restrictions and allow for the output to be just binary class (and indeed, our framework applies for multi-class setting as well, since the amount of information increases with more label classes from the algorithmic perspective), and for the number of neurons to exceed the input/output dimensions (overcomplete setting). Moreover, we extend beyond the realizable setting, and do not require the target functions to be generated from the class of neural networks under consideration.

2 Preliminaries and Problem Formulation

We first introduce some notations and then propose the problem formulation.

2.1 Notation

Let $[n] := \{1, 2, \ldots, n\}$, and $\|u\|$ denote the ℓ_2 or Euclidean norm of vector u, and $\langle u, v \rangle$ denote the inner product of vectors u and v. For matrix $C \in \mathbb{R}^{d \times k}$, the j-th column is referred by C_j or c_j, $j \in [k]$. Throughout this paper, $\nabla_x^{(m)}$ denotes the m-th order derivative operator w.r.t. variable x. For matrices $A, B \in \mathbb{R}^{d \times k}$, the *Khatri-Rao product* $C := A \odot B \in \mathbb{R}^{d^2 \times k}$ is defined such that $C(l + (i-1)d, j) = A(i,j) \cdot B(l,j)$, for $i, l \in [d], j \in [k]$.

Tensor: A real *m-th order tensor* $T \in \bigotimes^m \mathbb{R}^d$ is a member of the outer product of Euclidean spaces $\mathbb{R}^d$. The different dimensions of the tensor are referred to as *modes*. For instance, for a matrix, the first mode refers to columns and the second mode refers to rows.

Tensor matricization: For a third order tensor $T \in \mathbb{R}^{d \times d \times d}$, the matricized version along first mode denoted by $M \in \mathbb{R}^{d \times d^2}$ is defined such that

$$T(i,j,l) = M(i, l + (j-1)d), \quad i, j, l \in [d]. \tag{1}$$

Tensor rank: A 3rd order tensor $T \in \mathbb{R}^{d \times d \times d}$ is said to be rank-1 if it can be written in the form

$$T = w \cdot a \otimes b \otimes c \Leftrightarrow T(i,j,l) = w \cdot a(i) \cdot b(j) \cdot c(l), \tag{2}$$

where $\otimes$ represents the *outer product*, and $a, b, c \in \mathbb{R}^d$ are unit vectors. A tensor $T \in \mathbb{R}^{d \times d \times d}$ is said to have a CP (Candecomp/Parafac) *rank* k if it can be (minimally) written as the sum of k rank-1 tensors

$$T = \sum_{i \in [k]} w_i a_i \otimes b_i \otimes c_i, \quad w_i \in \mathbb{R}, \ a_i, b_i, c_i \in \mathbb{R}^d. \tag{3}$$

Derivative: For function $g(x) : \mathbb{R}^d \to \mathbb{R}$ with vector input $x \in \mathbb{R}^d$, the m-th order derivative w.r.t. variable x is denoted by $\nabla_x^{(m)} g(x) \in \bigotimes^m \mathbb{R}^d$ (which is a m-th order tensor) such that

$$\left[\nabla_x^{(m)} g(x)\right]_{i_1,\dots,i_m} := \frac{\partial g(x)}{\partial x_{i_1} \partial x_{i_2} \cdots \partial x_{i_m}}, \quad i_1, \dots, i_m \in [d]. \tag{4}$$

When it is clear from the context, we drop the subscript x and write the derivative as $\nabla^{(m)} g(x)$.

Fourier transform: For a function $f(x) : \mathbb{R}^d \to \mathbb{R}$, the multivariate Fourier transform $F(\omega) : \mathbb{R}^d \to \mathbb{R}$ is defined as

$$F(\omega) := \int_{\mathbb{R}^d} f(x) e^{-j\langle \omega, x\rangle} dx, \tag{5}$$

where variable $\omega \in \mathbb{R}^d$ is called the frequency variable, and j denotes the imaginary unit. We also denote the Fourier pair $(f(x), F(\omega))$ as $f(x) \xleftrightarrow{\text{Fourier}} F(\omega)$.

Function notations: Throughout the paper, we use the following convention to distinguish different types of functions. We use $f(x)$ (or y) to denote an arbitrary function and exploit $\tilde{f}(x)$ (or $\tilde{y}$) to denote the output of a realizable neural network. This helps us to differentiate between them. We also use notation $\hat{f}(x)$ (or $\hat{y}$) to denote the estimated (trained) neural networks using finite number of samples.

2.2 Problem formulation

We now introduce the problem of training a neural network in realizable and non-realizable settings, and elaborate on the notion of risk bound on how the trained neural network approximates an arbitrary function. It is known that continuous functions with compact domain can be arbitrarily well approximated by feedforward neural networks with one hidden layer and sigmoidal nonlinear functions (Cybenko, 1989a; Hornik et al., 1989; Barron, 1993).

The input (feature) is denoted by variable x, and output (label) is denoted by variable y. We assume the input and output are generated according to some joint density function $p(x, y)$ such that $(x_i, y_i) \overset{\text{i.i.d.}}{\sim} p(x, y)$, where (x_i, y_i) denotes the i-th sample. We assume knowledge of the input density $p(x)$, and demonstrate how it can be used to train a neural network to approximate the conditional density $p(y|x)$ in a computationally efficient manner. We discuss in Section 3.1 how the input density $p(x)$ can be estimated through numerous methods such as score matching or spectral methods. In settings such as experimental design, the input density $p(x)$ is known to the learner since she designs the density function, and our framework is directly applicable there. In addition, we do not need to know the normalization factor or the partition function of the input distribution $p(x)$, and the estimation up to normalization factor suffices.

Risk bound: In this paper, we propose a new algorithm for training neural networks and provide risk bounds with respect to an arbitrary target function. Risk is the expected loss over the joint probability density function of input x and output y. Here, we consider the squared ℓ_2 loss and bound the *risk error*

$$\mathbb{E}_x[|f(x) - \hat{f}(x)|^2], \tag{6}$$

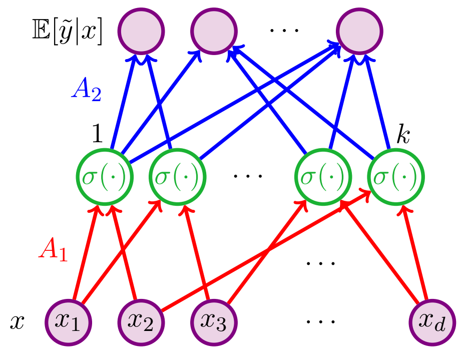


Figure 1: Graphical representation of a neural network, $\mathbb{E}[\tilde{y}|x] = A_2^\top \sigma(A_1^\top x + b_1) + b_2$. Note that this representation is for general vector output $\tilde{y}$ which can be also written as $\mathbb{E}[\tilde{y}|x] = \langle a_2, \sigma(A_1^\top x + b_1)\rangle + b_2$ in the case of scalar output $\tilde{y}$.

where $f(x)$ is an arbitrary function which we want to approximate by $\hat{f}(x)$ denoting the estimated (trained) neural network. This notion of risk is also called mean integrated squared error. The proposed risk error for a neural network consists of two parts: approximation error and estimation error. *Approximation error* is the error in fitting the target function $f(x)$ to a neural network with the given architecture $\tilde{f}(x)$, and *estimation error* is the error in training that network with finite number of samples denoted by $\hat{f}(x)$. Thus, the risk error measures the ability of the trained neural network to generalize to new data generated by function $f(x)$. We now introduce the realizable and non-realizable settings, which elaborates more these sources of error.

2.2.1 Realizable setting

In the realizable setting, the output is generated by a neural network. We consider a neural network with one hidden layer of dimension k. Let the output $\tilde{y} \in \{0,1\}$ be the binary label, and $x \in \mathbb{R}^d$ be the feature vector; see Section 6.2 for generalization to higher dimensional output (multi-label and multi-class), and also the continuous output case. We consider the label generating model

$$\tilde{f}(x) := \mathbb{E}[\tilde{y}|x] = \langle a_2, \sigma(A_1^\top x + b_1)\rangle + b_2, \tag{7}$$

where $\sigma(\cdot)$ is (linear/nonlinear) elementwise function. See Figure 1 for a schematic representation of label-function in (7) in the general case of vector output $\tilde{y}$.

In the realizable setting, the goal is to train the neural network in (7), i.e., to learn the weight matrices (vectors) $A_1 \in \mathbb{R}^{d\times k}$, $a_2 \in \mathbb{R}^k$ and bias vectors $b_1 \in \mathbb{R}^k$, $b_2 \in \mathbb{R}$. This only involves the estimation analysis where we have a label-function $\tilde{f}(x)$ specified in (7) with fixed unknown parameters A_1, b_1, a_2, b_2, and the goal is to learn these parameters and finally bound the overall function estimation error $\mathbb{E}_x[|\tilde{f}(x) - \hat{f}(x)|^2]$, where $\hat{f}(x)$ is the estimation of fixed neural network $\tilde{f}(x)$ given finite samples. For this task, we propose a computationally efficient algorithm which requires only polynomial number of samples for bounded estimation error. This is the first time that such a result has been established for any neural network.

Algorithm 1 NN-LIFT (Neural Network LearnIng using Feature Tensors)

```
input Labeled samples $\{(x_i, y_i) : i \in [n]\}$, parameter $\tilde{\epsilon}_1$, parameter $\lambda$.
input Third order score function $\mathcal{S}_3(x)$ of the input $x$; see Equation (8) for the definition.
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($\widehat{T}$); see Section 3.2 and Appendix B for details.
3: $\hat{b}_1$ = Fourier method($\{(x_i, y_i) : i \in [n]\}, \hat{A}_1, \tilde{\epsilon}_1$); see Procedure 2.
4: $(\hat{a}_2, \hat{b}_2)$ = Ridge regression($\{(x_i, y_i) : i \in [n]\}, \hat{A}_1, \hat{b}_1, \lambda$); see Procedure 3.
5: return $\hat{A}_1, \hat{a}_2, \hat{b}_1, \hat{b}_2$.
```

2.2.2 Non-realizable setting

In the non-realizable setting, the output is an arbitrary function $f(x)$ which is not necessarily a neural network. We want to approximate $f(x)$ by $\hat{f}(x)$ denoting the estimated (trained) neural network. In this setting, the additional approximation analysis is also required. In this paper, we combine the estimation result in realizable setting with the approximation bounds in Barron (1993) leading to risk bounds with respect to the target function $f(x)$; see (6) for the definition of risk. The detailed results are provided in Section 5.

3 NN-LIFT Algorithm

In this section, we introduce our proposed method for learning neural networks using tensor, Fourier and regression techniques. Our method is shown in Algorithm 1 named NN-LIFT (Neural Network LearnIng using Feature Tensors). The algorithm has three main components. The first component involves estimating the weight matrix of the first layer denoted by $A_1 \in \mathbb{R}^{d \times k}$ by a tensor decomposition method. The second component involves estimating the bias vector of the first layer $b_1 \in \mathbb{R}^k$ by a Fourier method. We finally estimate the parameters of last layer $a_2 \in \mathbb{R}^k$ and $b_2 \in \mathbb{R}$ by linear regression.

Note that most of the unknown parameters (compare the dimensions of matrix A_1, vectors a_2, b_1, and scalar b_2) are estimated in the first part, and thus, this is the main part of the algorithm. Given this fact, we also provide an alternative method for the estimation of other parameters of the model, given the estimate of A_1 from the tensor method. This is based on incrementally adding neurons, one by one, whose first layer weights are given by A_1 and the remaining parameters are updated using brute force search on a grid. Since each update involves just updating the corresponding bias term b_1, and its contribution to the final output, this is low dimensional, and can be done efficiently; details are in Section 6.3.

We now explain the steps of Algorithm 1 in more details.

3.1 Score function

The m-th order score function $\mathcal{S}_m(x) \in \bigotimes^m \mathbb{R}^d$ is defined as (Janzamin et al., 2014)

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

where $p(x)$ is the probability density function of random vector $x \in \mathbb{R}^d$. In addition, $\nabla_x^{(m)}$ denotes the m-th order derivative operator; see (4) for the precise definition. The main property of score functions as yielding differential operators that enables us to estimate the weight matrix A_1 via tensor decomposition is discussed in the next subsection; see Equation (9).

In this paper, we assume access to a sufficiently good approximation of the input pdf $p(x)$ and the corresponding score functions $\mathcal{S}_2(x)$, $\mathcal{S}_3(x)$. Indeed, estimating these quantities in general is a hard problem, but there exist numerous instances where this becomes tractable. Examples include spectral methods for learning latent variable models such as Gaussian mixtures, topic or admixture models, independent component analysis (ICA) and so on (Anandkumar et al., 2014b). Moreover, there have been recent advances in non-parametric score matching methods (Sriperumbudur et al., 2013) for density estimation in infinite dimensional exponential families with guaranteed convergence rates. These methods can be used to estimate the input pdf in an unsupervised manner. Below, we discuss in detail about score function estimation methods. In this paper, we focus on how we can use the input generative information to make training of neural networks tractable. For simplicity, in the subsequent analysis, we assume that these quantities are perfectly known; it is possible to extend the perturbation analysis to take into account the errors in estimating the input pdf; see Remark 4.

Estimation of score function: There are various efficient methods for estimating the score function. The framework of score matching is popular for parameter estimation in probabilistic models (Hyvärinen, 2005; Swersky et al., 2011), where the criterion is to fit parameters based on matching the data score function. Swersky et al. (2011) analyze the score matching for latent energy-based models. In deep learning, the framework of auto-encoders attempts to find encoding and decoding functions which minimize the reconstruction error under added noise; the so-called Denoising Auto-Encoders (DAE). This is an unsupervised framework involving only unlabeled samples. Alain and Bengio (2012) argue that the DAE approximately learns the first order score function of the input, as the noise variance goes to zero. Sriperumbudur et al. (2013) propose non-parametric score matching methods for density estimation in infinite dimensional exponential families with guaranteed convergence rates. Therefore, we can use any of these methods for estimating $\mathcal{S}_1(x)$ and use the recursive form (Janzamin et al., 2014)

$$\mathcal{S}_m(x) = -\mathcal{S}_{m-1}(x) \otimes \nabla_x \log p(x) - \nabla_x \mathcal{S}_{m-1}(x)$$

to estimate higher order score functions.

3.2 Tensor decomposition

The score functions are new representations (extracted features) of input data x that can be used for training neural networks. We use score functions and labels of training data to form the empirical cross-moment $\widehat{T} = \frac{1}{n} \sum_{i \in [n]} y_i \cdot \mathcal{S}_3(x_i)$. We decompose tensor $\widehat{T}$ to estimate the columns of A_1. The following discussion reveals why tensor decomposition is relevant for this task.

The score functions have the property of yielding differential operators with respect to the input distribution. More precisely, for label-function $f(x) := \mathbb{E}[y|x]$, Janzamin et al. (2014) show that

$$\mathbb{E}[y \cdot \mathcal{S}_3(x)] = \mathbb{E}[\nabla_x^{(3)} f(x)].$$

Now for the neural network output in (7), note that the function $\tilde{f}(x)$ is a non-linear function of both input x and weight matrix A_1. The expectation operator $\mathbb{E}[\cdot]$ averages out the dependency on x, and the derivative acts as a *linearization operator* as follows. In the neural network output (7), we observe that the columns of weight vector A_1 are the linear coefficients involved with input variable x. When taking the derivative of this function, by the chain rule, these linear coefficients shows up in the final form. In particular, we show in Lemma 6 (see Section 7) that for neural network in (7), we have

$$\mathbb{E}\left[\tilde{y} \cdot \mathcal{S}_3(x)\right] = \sum_{j \in [k]} \lambda_j \cdot (A_1)_j \otimes (A_1)_j \otimes (A_1)_j \in \mathbb{R}^{d \times d \times d}, \tag{9}$$

where $(A_1)_j \in \mathbb{R}^d$ denotes the j-th column of A_1, and $\lambda_j \in \mathbb{R}$ denotes the coefficient; refer to Equation (3) for the notion of tensor rank and its rank-1 components. This clarifies how the score function acts as a linearization operator while the final output is nonlinear in terms of A_1. The above form also clarifies the reason behind using tensor decomposition in the learning framework.

Tensor decomposition algorithm: The goal of tensor decomposition algorithm is to recover the rank-1 components of tensor. For this step, we use the tensor decomposition algorithm proposed in (Anandkumar et al., 2014b,c); see Appendix B for details. The main step of the tensor decomposition method is the *tensor power iteration* which is the generalization of matrix power iteration to 3rd order tensors. The tensor power iteration is given by

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

where $u \in \mathbb{R}^d$, $T(I,u,u) := \sum_{j,l \in [d]} u_j u_l T(:,j,l) \in \mathbb{R}^d$ is a *multilinear* combination of tensor *fibers*.[3] The convergence guarantees of tensor power iteration for orthogonal tensor decomposition have been developed in the literature (Zhang and Golub, 2001; Anandkumar et al., 2014b). Note that we first orthogonalize the tensor via whitening procedure and then apply the tensor power iteration. Thus, the original tensor decomposition need not to be orthogonal.

Computational Complexity: It is popular to perform the tensor decomposition in a stochastic manner which reduces the computational complexity. This is done by splitting the data into mini-batches. Starting with the first mini-batch, we perform a small number of tensor power iterations, and then use the result as initialization for the next mini-batch, and so on. As mentioned earlier, we assume that a sufficiently good approximation of score function tensor is given to us. For specific cases where we have this tensor in factor form, we can reduce the computational complexity of NN-LIFT by not computing the whole tensor explicitly. By having factor form, we mean when we can write the score function tensor in terms of summation of rank-1 components which could be the summation over samples, or from other existing structures in the model. We now state a few examples when we have the factor form, and provide the computational complexity. For example, if input follows Gaussian distribution, the score function has a simple polynomial form, and the computational complexity of tensor decomposition is $O(nkdR)$, where n is the number of samples and R is the number of initializations for the tensor decomposition. Similar argument follows when the input distribution is mixture of Gaussian distributions.

We can also analyze complexity for more complex inputs. If we fit the input data into a Restricted Boltzmann Machines (RBM) model, the computational complexity of our method is

[3]Tensor fibers are the vectors which are derived by fixing all the indices of the tensor except one of them, e.g., $T(:,j,l)$ in the above expansion.

Procedure 2 Fourier method for estimating b_1

```
input Labeled samples $\{(x_i, y_i) : i \in [n]\}$, estimate $\hat{A}_1$, parameter $\tilde{\epsilon}_1$.
input Probability density function $p(x)$ of the input $x$.
1: for $l = 1$ to $k$ do
2:     Let $\Omega_l := \left\{\omega \in \mathbb{R}^d : \|\omega\| = \frac{1}{2}, \left|\langle \omega, (\hat{A}_1)_l \rangle\right| \geq \frac{1-\tilde{\epsilon}_1^2/2}{2}\right\}$, and $|\Omega_l|$ denotes the surface area of $d-1$ dimensional manifold $\Omega_l$.
3:     Draw $n$ i.i.d. random frequencies $\omega_i, i \in [n]$, uniformly from set $\Omega_l$.
4:     Let $v := \frac{1}{n} \sum_{i\in[n]} \frac{y_i}{p(x_i)} e^{-j\langle \omega_i, x_i \rangle}$ which is a complex number as $v = |v|e^{j\angle v}$. The operators $|\cdot|$ and $\angle\cdot$ respectively denote the magnitude and phase operators.
5:     Let $\hat{b}_1(l) := \frac{1}{\pi}(\angle v - \angle\Sigma(1/2))$, where $\sigma(x) \xleftrightarrow{\text{Fourier}} \Sigma(\omega)$.
6: end for
7: return $\hat{b}_1$.
```

$O(nkdd_hR)$. Here d_h is the number of neurons of the first layer of the RBM used for approximating the input distribution. Tensor methods are also embarrassingly parallelizable. When performed in parallel, the computational complexity would be $O(\log(\min\{d, d_h\}))$ with $O(nkdd_hR/\log(\min(d, d_h)))$ processors. Alternatively, we can also exploit recent tensor sketching approaches (Wang et al., 2015) for computing tensor decompositions efficiently. Wang et al. (2015) build on the idea of count sketches and show that the running time is linear in the input dimension and the number of samples, and is independent in the order of the tensor. Thus, tensor decompositions can be computed efficiently.

3.3 Fourier method

The second part of the algorithm estimates the first layer bias vector $b_1 \in \mathbb{R}^k$. This step is very different from the previous step for estimating A_1 which was based on tensor decomposition methods. This is a Fourier-based method where complex variables are formed using labeled data and random frequencies in the Fourier domain; see Procedure 2. We prove in Lemma 7 that the entries of b_1 can be estimated from the phase of these complex numbers. We also observe in Lemma 7 that the magnitude of these complex numbers can be used to estimate a_2; this is discussed in Appendix C.2.

Polynomial-time random draw from set Ω_l: Note that the random frequencies are drawn from a $d-1$ dimensional manifold denoted by Ω_l which is the intersection of sphere $\|\omega\| = \frac{1}{2}$ and cone $\left|\langle \omega, (\hat{A}_1)_l \rangle\right| \geq \frac{1-\tilde{\epsilon}_1^2/2}{2}$ in $\mathbb{R}^d$. This manifold is actually the surface of a spherical cap. In order to draw these frequencies in polynomial time, we consider the d-dimensional spherical coordinate system such that one of the angles is defined based on the cone axis. We can then directly impose the cone constraint by limiting the corresponding angle in the random draw. In addition, Kothari and Meka (2014) propose a method for generating pseudo-random variables from the spherical cap in logarithmic time.

Remark 1 (Knowledge of input distribution only up to normalization factor)**.** *The computation of score function and the Fourier method both involve knowledge about input pdf $p(x)$. However, we do not need to know the normalization factor, also known as partition function, of the input pdf. For the score function, it is immediately seen from the definition in* (8) *since the normalization factor*

Procedure 3 Ridge regression method for estimating a_2 and b_2

```
input Labeled samples $\{(x_i, y_i) : i \in [n]\}$, estimates $\hat{A}_1$ and $\hat{b}_1$, regularization parameter $\lambda$.
1: Let $\hat{h}_i := \sigma(\hat{A}_1^\top x_i + \hat{b}_1)$, $i \in [n]$, denote the estimation of the neurons.
2: Append each neuron $\hat{h}_i$ by the dummy variable 1 to represent the bias, and thus, $\hat{h}_i \in \mathbb{R}^{k+1}$.
3: Let $\hat{\Sigma}_{\hat{h}} := \frac{1}{n} \sum_{i \in [n]} \hat{h}_i \hat{h}_i^\top \in \mathbb{R}^{(k+1)\times(k+1)}$ denote the empirical covariance of $\hat{h}$.
4: Let $\hat{\beta}_\lambda \in \mathbb{R}^{k+1}$ denote the estimated parameters by $\lambda$-regularized ridge regression as
       $$\hat{\beta}_\lambda = \left(\hat{\Sigma}_{\hat{h}} + \lambda I_{k+1}\right)^{-1} \cdot \frac{1}{n} \left( \sum_{i \in [n]} y_i \hat{h}_i \right), \qquad (10)$$
   where $I_{k+1}$ denotes the $(k+1)$-dimensional identity matrix.
5: return $\hat{a}_2 := \hat{\beta}_\lambda(1:k)$, $\hat{b}_2 := \hat{\beta}_\lambda(k+1)$.
```

is canceled out by the division by $p(x)$, and thus, the estimation of score function is at most as hard as estimation of input pdf up to normalization factor. In the Fourier method, we can use the non-normalized estimation of input pdf which leads to a normalization mismatch in the estimation of corresponding complex number. This is not a problem since we only use the phase information of these complex numbers.

3.4 Ridge regression method

For the neural network model in (7), given a good estimation of neurons, we can estimate the parameters of last layer by linear regression. We provide Procedure 3 in which we use ridge regression algorithm to estimate the parameters of last layer a_2 and b_2. See Appendix C.3 for the details of ridge regression and the corresponding analysis and guarantees.

4 Risk Bound in the Realizable Setting

In this section, we provide guarantees in the realizable setting, where the function $\tilde{f}(x) := \mathbb{E}[\tilde{y}|x]$ is generated by a neural network as in (7). We provide the estimation error bound on the overall function recovery $\mathbb{E}_x[|\tilde{f}(x) - \hat{f}(x)|^2]$ when the estimation is done by Algorithm 1.

We provide guarantees in the following settings. 1) In the basic case, we consider the *undercomplete* regime $k \leq d$, and provide the results assuming A_1 is full column rank. 2) In the second case, we form higher order cross-moments and tensorize it into a lower order tensor. This enables us to learn the network in the overcomplete regime $k > d$, when the Khatri-Rao product $A_1 \odot A_1 \in \mathbb{R}^{d^2 \times k}$ is full column rank. We call this the *overcomplete* setting and this can handle up to $k = O(d^2)$. Similarly, we can extend to larger k by tensorizing higher order moments in the expense of additional computational complexity.

We define the following quantity for label function $\tilde{f}(\cdot)$ as

$$\tilde{\zeta}_{\tilde{f}} := \int_{\mathbb{R}^d} \tilde{f}(x) dx.$$

Note that in the binary classification setting ($\tilde{y} \in \{0,1\}$), we have $\mathbb{E}[\tilde{y}|x] := \tilde{f}(x) \in [0,1]$ which is always positive, and there is no square of $\tilde{f}(x)$ considered in the above quantity.

Let η denote the noise in the neural network model in (7) such that the output is

$$\tilde{y} = \tilde{f}(x) + \eta.$$

Note that the noise η is not necessarily independent of x; for instance, in the classification setting or binary output $\tilde{y} \in \{0, 1\}$, the noise in dependent on x.

Conditions for Theorem 3:

- **Non-degeneracy of weight vectors**: In the undercomplete setting ($k \leq d$), the weight matrix $A_1 \in \mathbb{R}^{d \times k}$ is full column rank and $s_{\min}(A_1) > \epsilon$, where $s_{\min}(\cdot)$ denotes the minimum singular value, and $\epsilon > 0$ is related to the target error in recovering the columns of A_1. In the overcomplete setting ($k \leq d^2$), the Khatri-Rao product $A_1 \odot A_1 \in \mathbb{R}^{d^2 \times k}$ is full column rank[4], and $s_{\min}(A_1 \odot A_1) > \epsilon$; see Remark 3 for generalization.
- **Conditions on nonlinear activating function** $\sigma(\cdot)$: the coefficients

  $$\lambda_j := \mathbb{E}\left[\sigma'''(z_j)\right] \cdot a_2(j), \quad \tilde{\lambda}_j := \mathbb{E}\left[\sigma''(z_j)\right] \cdot a_2(j), \quad j \in [k],$$

  in (20) and (40) are nonzero. Here, $z := A_1^\top x + b_1$ is the input to the nonlinear operator $\sigma(\cdot)$. In addition, $\sigma(\cdot)$ satisfies the *Lipschitz* property[5] with constant L such that $|\sigma(u) - \sigma(u')| \leq L \cdot |u - u'|$, for $u, u' \in \mathbb{R}$. Suppose that the nonlinear activating function $\sigma(z)$ satisfies the property such that $\sigma(z) = 1 - \sigma(-z)$. Many popular activating functions such as step function, sigmoid function and tanh function satisfy this last property.
- **Subgaussian noise**: There exists a finite $\sigma_{\text{noise}} \geq 0$ such that, almost surely,

  $$\mathbb{E}_\eta[\exp(\alpha\eta)|x] \leq \exp(\alpha^2 \sigma_{\text{noise}}^2/2), \quad \forall \alpha \in \mathbb{R},$$

  where η denotes the noise in the output $\tilde{y}$.
- **Bounded statistical leverage**: There exists a finite $\rho_\lambda \geq 1$ such that, almost surely,

  $$\frac{\sqrt{k}}{\sqrt{(\inf\{\lambda_j\} + \lambda) k_{1,\lambda}}} \leq \rho_\lambda,$$

  where $k_{1,\lambda}$ denotes the effective dimensions of the hidden layer $h := \sigma(A_1^\top x + b_1)$ as $k_{1,\lambda} := \sum_{j \in [k]} \frac{\lambda_j}{\lambda_j + \lambda}$. Here, λ_j's denote the (positive) eigenvalues of the hidden layer covariance matrix Σ_h, and λ is the regularization parameter of ridge regression.

We now elaborate on these conditions. The *non-degeneracy of weight vectors* are required for the tensor decomposition analysis in the estimation of A_1. In the undercomplete setting, the algorithm first orthogonalizes (through whitening procedure) the tensor given in (9), and then decomposes it through tensor power iteration. Note that the convergence of power iteration for orthogonal tensor decomposition is guaranteed (Zhang and Golub, 2001; Anandkumar et al., 2014b). For the

[4]It is shown in Bhaskara et al. (2013) that this condition is satisfied under smoothed analysis.

[5] If the step function $\sigma(u) = 1_{\{u>0\}}(u)$ is used as the activating function, the Lipschitz property does not hold because of the non-continuity at $u = 0$. But, we can assume the Lipschitz property holds in the linear continuous part, i.e., when $u, u' > 0$ or $u, u' < 0$. We then argue that the input to the step function $1_{\{u>0\}}(u)$ is w.h.p. in the linear interval (where the Lipschitz property holds).

orthogonalization procedure to work, we need the tensor components (the columns of matrix A_1) to be linearly independent. In the overcomplete setting, the algorithm performs the same steps with the additional tensorizing procedure; see Appendix B for details. In this case, a higher order tensor is given to the algorithm and it is first tensorized before performing the same steps as in the undercomplete setting. Thus, the same conditions are now imposed on $A_1 \odot A_1$.

In addition to the non-degeneracy condition on weight matrix A_1, the *coefficients condition* on λ_j's is also required to ensure the corresponding rank-1 components in (9) do not vanish, and thus, the tensor decomposition algorithm recovers them. Similarly, the coefficients $\tilde{\lambda}_j$ should be also nonzero to enable us using the second order moment $\tilde{M}_2$ in (39) in the whitening step of tensor decomposition algorithm. If one of the coefficients vanishes, we use the other option to perform the whitening; see Remark 5 and Procedure 5 for details. Note that the amount of non-linearity of $\sigma(\cdot)$ affects the magnitude of the coefficients. It is also worth mentioning that although we use the third derivative notation $\sigma'''(\cdot)$ in characterizing the coefficients λ_j (and similarly $\sigma''(\cdot)$ in $\tilde{\lambda}_j$), we do not need the differentiability of non-linear function $\sigma(\cdot)$ in all points. In particular, when input x is a continuous variable, we use Dirac delta function $\delta(\cdot)$ as the derivative in non-continuous points; for instance, for the derivative of step function $1_{\{x>0\}}(x)$, we have $\frac{d}{dx}1_{\{x>0\}}(x) = \delta(x)$. Thus, in general, we only need the expectations $\mathbb{E}\left[\sigma'''(z_j)\right]$ and $\mathbb{E}\left[\sigma''(z_j)\right]$ to exist for these type of functions and the corresponding higher order derivatives.

We impose the *Lipschitz* condition on the non-linear activating function to limit the error propagated in the hidden layer, when the first layer parameters are estimated by the neural network and Fourier methods. The condition $\sigma(z) = 1 - \sigma(-z)$ is also assumed to tackle the sign issue in recovering the columns of A_1; see Remark 6 for the details. The *subgaussian noise* and the *bounded statistical leverage* conditions are standard conditions, required for ridge regression, which is used for estimating the parameters of the second layer of the neural network. Both parameters σ_{noise}, and ρ_λ affect the sample complexity in the final guarantees.

Imposing additional bounds on the parameters of the neural network are useful in learning these parameters with computationally efficient algorithms since it limits the searching space for training these parameters. In particular, for the Fourier analysis, we assume the following conditions. Suppose the columns of weight matrix A_1 are normalized, i.e., $\|(A_1)_j\| = 1, j \in [k]$, and the entries of first layer bias vector b_1 are bounded as $|b_1(l)| \leq 1, l \in [k]$. Note that the normalization condition on the columns of A_1 is also needed for identifiability of the parameters. For instance, if the non-linear operator is the step function $\sigma(z) = 1_{\{z>0\}}(z)$, then matrix A_1 is only identifiable up to its norm, and thus, such normalization condition is required for identifiability. The estimation of entries of the bias vector b_1 is obtained from the phase of a complex number through Fourier analysis; see Procedure 2 for details. Since there is ambiguity in the phase of a complex number[6], we impose the bounded assumption on the entries of b_1 to avoid this ambiguity.

Let $p(x)$ satisfy some mild regularity conditions on the boundaries of the support of $p(x)$. In particular, all the entries of (matrix-output) functions

$$\tilde{f}(x) \cdot \nabla^{(2)} p(x), \quad \nabla \tilde{f}(x) \cdot \nabla p(x)^\top, \quad \nabla^{(2)} \tilde{f}(x) \cdot p(x) \tag{11}$$

should go to zero on the boundaries of support of $p(x)$. These regularity conditions are required for the properties of the score function to hold; see Janzamin et al. (2014) for more details.

In addition to the above main conditions, we also need some mild conditions which are not crucial for the recovery guarantees and are mostly assumed to simplify the presentation of the

[6]A complex number does not change if an integer multiple of 2π is added to its phase.

main results. These conditions can be relaxed more. Suppose the input x is bounded, i.e., $x \in B_r$, where $B_r := \{x : \|x\| \leq r\}$. Assume the input probability density function $p(x) \geq \psi$ for some $\psi > 0$, and for any $x \in B_r$. The regularity conditions in (11) might seem contradictory with the lower bound condition $p(x) \geq \psi$, but there is an easy fix for that. The lower bound on $p(x)$ is required for the analysis of the Fourier part of the algorithm. We can have a continuous $p(x)$, while in the Fourier part, we only use x's such that $p(x) \geq \psi$, and ignore the rest. This only introduces a probability factor $\Pr[x : p(x) \geq \psi]$ in the analysis.

Settings of algorithm in Theorem 3:

- No. of iterations in Algorithm 7: $N = \Theta\left(\log \frac{1}{\epsilon}\right)$.
- No. of initializations in Procedure 8: $R \geq \operatorname{poly}(k)$.
- Parameter $\tilde{\epsilon}_1 = \tilde{O}\left(\frac{1}{\sqrt{n}}\right)$ in Procedure 2, where n is the number of samples.
- We exploit the empirical second order moment $\widehat{M}_2 := \frac{1}{n}\sum_{i\in[n]} y_i \cdot \mathcal{S}_2(x_i)$, in the whitening Procedure 5, which is the first option stated in the procedure. See Remark 5 for further discussion about the other option.

Theorem 3 (NN-LIFT guarantees: estimation bound in the realizable setting)**.** *Assume the above settings and conditions hold. For $\epsilon > 0$, suppose the number of samples n satisfies (up to* log *factors)*

$$n \geq \tilde{O}\Bigg(\frac{k}{\epsilon^2} \cdot \mathbb{E}\left[\left\|M_3(x)M_3^\top(x)\right\|\right] \tag{12}$$

$$\cdot \operatorname{poly}\left(\tilde{y}_{\max}, \frac{\mathbb{E}\left[\left\|\mathcal{S}_2(x)\mathcal{S}_2^\top(x)\right\|\right]}{\mathbb{E}\left[\left\|M_3(x)M_3^\top(x)\right\|\right]}, \frac{\tilde{\zeta}_{\tilde{f}}}{\psi}, \frac{\tilde{\lambda}_{\max}}{\tilde{\lambda}_{\min}}, \frac{1}{\lambda_{\min}}, \frac{s_{\max}(A_1)}{s_{\min}(A_1)}, |\Omega_l|, L, \frac{\|a_2\|}{(a_2)_{\min}}, |b_2|, \sigma_{noise}, \rho_\lambda\right)\Bigg).$$

See (32), (55), (57) *and* (59) *for the complete form of sample complexity. Here, $M_3(x) \in \mathbb{R}^{d\times d^2}$ denotes the matricization of score function tensor $\mathcal{S}_3(x) \in \mathbb{R}^{d\times d\times d}$; see* (1) *for the definition of matricization. Furthermore, $\lambda_{\min} := \min_{j\in[k]} |\lambda_j|$, $\tilde{\lambda}_{\min} := \min_{j\in[k]} |\tilde{\lambda}_j|$, $\tilde{\lambda}_{\max} := \max_{j\in[k]} |\tilde{\lambda}_j|$, $(a_2)_{\min} := \min_{j\in[k]} |a_2(j)|$, and $\tilde{y}_{\max}$ is such that $|\tilde{f}(x)| \leq \tilde{y}_{\max}$, for $x \in B_r$. Then the function estimate $\hat{f}(x) := \langle \hat{a}_2, \sigma(\hat{A}_1^\top x + \hat{b}_1)\rangle + \hat{b}_2$ using the estimated parameters $\hat{A}_1, \hat{b}_1, \hat{a}_2, \hat{b}_2$ (output of NN-LIFT Algorithm 1) satisfies the estimation error*

$$\mathbb{E}_x[|\hat{f}(x) - \tilde{f}(x)|^2] \leq \tilde{O}(\epsilon^2).$$

See Section 7 and Appendix C for the proof of theorem. Thus, we estimate the neural network in polynomial time and sample complexity. This is one of the first results to provide a guaranteed method for training neural networks with efficient computational and statistical complexity. Note that although the sample complexity in (Barron, 1993) is smaller as $n \geq \tilde{O}\left(\frac{kd}{\epsilon^2}\right)$, the proposed algorithm in (Barron, 1993) is not computationally efficient.

Remark 2 (Sample complexity for Gaussian input)**.** *If the input x is Gaussian as $x \sim \mathcal{N}(0, I_d)$, then we know that $\mathbb{E}\left[\left\|M_3(x)M_3^\top(x)\right\|\right] = \tilde{O}\left(d^3\right)$ and $\mathbb{E}\left[\left\|\mathcal{S}_2(x)\mathcal{S}_2^\top(x)\right\|\right] = \tilde{O}\left(d^2\right)$, and the above sample complexity is simplified.*

Remark 3 (Higher order tensorization). *We stated that by tensorizing higher order tensors to lower order ones, we can estimate overcomplete models where the hidden layer dimension k is larger than the input dimension d. We can generalize this idea to higher order tensorizing such that m modes of the higher order tensor are tensorized into a single mode in the resulting lower order tensor. This enables us to estimate the models up to $k = O(d^m)$ assuming the matrix $A_1 \odot \cdots \odot A_1$ (m Khatri-Rao products) is full column rank. This is possible with the higher computational complexity.*

Remark 4 (Effect of erroneous estimation of $p(x)$). *The input probability density function $p(x)$ is directly used in the Fourier part of the algorithm, and also indirectly used in the tensor decomposition part to compute the score function $\mathcal{S}_3(x)$; see* (8). *In the above analysis, to simplify the presentation, we assume we exactly know these functions, and thus, there is no additional error introduced by estimating them. It is straightforward to incorporate the corresponding errors in estimating input density into the final bound.*

Remark 5 (Alternative whitening prodecure). *In whitening Procedure 5, two options are provided for constructing the second order moment M_2. In the above analysis, we used the first option which exploits the second order score function. If any coefficient $\tilde{\lambda}_j, j \in [k]$, in* (39) *vanishes, we cannot use the second order score function in the whitening procedure, and we use the other option for whitening; see Procedure 5 for the details.*

5 Risk Bound in the Non-realizable Setting

In this section, we provide the risk bound for training the neural network with respect to an arbitrary target function; see Section 2.2 for the definition of the risk.

In order to provide the risk bound with respect to an arbitrary target function, we also need to argue the approximation error in addition to the estimation error. For an arbitrary function $f(x)$, we need to find a neural network whose error in approximating the function can be bounded. We then combine it with the estimation error in training that neural network. This yields the final risk bound.

The approximation problem is about finding a neural network that approximates an arbitrary function $f(x)$ with bounded error. Thus, this is different from the realizable setting where there is a fixed neural network and we only analyze its estimation. Barron (1993) provides an approximation bound for the two-layer neural network and we exploit that here. His result is based on the Fourier properties of function $f(x)$. Recall from (5) the definition of Fourier transform of $f(x)$, denoted by $F(\omega)$, where ω is called the frequency variable. Define the first absolute moment of the Fourier magnitude distribution as

$$C_f := \int_{\mathbb{R}^d} \|\omega\|_2 \cdot |F(\omega)| d\omega. \tag{13}$$

Barron (1993) analyzes the approximation properties of

$$\tilde{f}(x) = \sum_{j \in [k]} a_2(j) \sigma\big(\langle (A_1)_j, x \rangle + b_1(j)\big), \quad \|(A_1)_j\| = 1, |b_1(j)| \leq 1, |a_2(j)| \leq 2C_f, j \in [k], \tag{14}$$

where the columns of weight matrix A_1 are the normalized version of random frequencies drawn from the Fourier magnitude distribution $|F(\omega)|$ weighted by the norm of the frequency vector. More precisely,

$$\omega_j \overset{\text{i.i.d.}}{\sim} \frac{\|\omega\|}{C_f} |F(\omega)|, \quad (A_1)_j = \frac{\omega_j}{\|\omega_j\|}, \quad j \in [k]. \tag{15}$$

See Section 7.2.1 for a detailed discussion on this connection between the columns of weight matrix A_1 and the random frequency draws from the Fourier magnitude distribution, and see how this is argued in the proof of the approximation bound. The other parameters a_2, b_1 need to be also found. He then shows the following approximation bound for (14).

Theorem 4 (Approximation bound, Theorem 3 of Barron (1993))**.** *For a function $f(x)$ with bounded C_f, there exists an approximation $\tilde{f}(x)$ in the form of* (14) *that satisfies the approximation bound*

$$\mathbb{E}_x[|\overline{f}(x) - \tilde{f}(x)|^2] \leq O(r^2 C_f^2) \cdot \left(\frac{1}{\sqrt{k}} + \delta_1 \right)^2 ,$$

where $\overline{f}(x) = f(x) - f(0)$. Here, for $\tau > 0$,

$$\delta_\tau := \inf_{0<\xi\leq 1/2} \left\{ 2\xi + \sup_{|z|\geq\xi} \left| \sigma(\tau z) - 1_{\{z>0\}}(z) \right| \right\} \tag{16}$$

is a distance between the unit step function $1_{\{z>0\}}(z)$ and the scaled sigmoidal function $\sigma(\tau z)$.

See Barron (1993) for the complete proof of the above theorem. For completeness, we have also reviewed the main ideas of this proof in Section 7.2. We now provide the formal statement of our risk bound.

Conditions for Theorem 5:

- The nonlinear activating function $\sigma(\cdot)$ is an arbitrary sigmoidal function satisfying the aforementioned Lipschitz condition. Note that a sigmoidal function is a bounded measurable function on the real line for which $\sigma(z) \to 1$ as $z \to \infty$ and $\sigma(z) \to 0$ as $z \to -\infty$.
- For $\epsilon > 0$, suppose the number of samples n satisfies (up to log factors)

$$n \geq \tilde{O}\left(\frac{k}{\epsilon^2} \cdot \mathbb{E}\left[\left\| M_3(x) M_3^\top(x) \right\| \right] \right. \tag{17}$$
$$\left. \cdot \operatorname{poly}\left(y_{\max}, \frac{\mathbb{E}\left[\left\| \mathcal{S}_2(x) \mathcal{S}_2^\top(x) \right\| \right]}{\mathbb{E}\left[\left\| M_3(x) M_3^\top(x) \right\| \right]}, \frac{\zeta_f}{\psi}, \frac{\tilde{\lambda}_{\max}}{\tilde{\lambda}_{\min}}, \frac{1}{\lambda_{\min}}, \frac{s_{\max}(A_1)}{s_{\min}(A_1)}, |\Omega_l|, L, \frac{\|a_2\|}{(a_2)_{\min}}, |b_2|, \sigma_{\text{noise}}, \rho_\lambda \right) \right),$$

  where $\zeta_f := \int_{\mathbb{R}^d} f(x)^2 dx$; notice the difference with $\tilde{\zeta}_{\tilde{f}}$. Note that this is the same sample complexity as in Theorem 3 with $\tilde{y}_{\max}$ substituted with $y_{\max}$ and $\tilde{\zeta}_{\tilde{f}}$ substituted with ζ_f.
- The target function $f(x)$ is bounded, and for $\epsilon > 0$, it has bounded C_f as

$$C_f \leq \tilde{O}\left(\left(\frac{1}{\sqrt{k}} + \delta_1 \right)^{-1} \cdot \left(\frac{1}{\sqrt{k}} + \epsilon \right) \cdot \frac{1}{\sqrt{\mathbb{E}\left[\|\mathcal{S}_3(x)\|^2 \right]}} \right. \tag{18}$$
$$\left. \cdot \operatorname{poly}\left(\frac{1}{r}, \frac{\mathbb{E}\left[\|\mathcal{S}_3(x)\|^2 \right]}{\mathbb{E}\left[\|\mathcal{S}_2(x)\|^2 \right]}, \psi, \frac{\tilde{\lambda}_{\min}}{\tilde{\lambda}_{\max}}, \lambda_{\min}, \frac{s_{\min}(A_1)}{s_{\max}(A_1)}, \frac{1}{|\Omega_l|}, \frac{1}{L}, \frac{(a_2)_{\min}}{\|a_2\|}, \frac{1}{|b_2|}, \frac{1}{\sigma_{\text{noise}}}, \frac{1}{\rho_\lambda} \right) \right).$$

  See (60) and (62) for the complete form of bound on C_f. For Gaussian input $x \sim \mathcal{N}(0, I_d)$, we have $\sqrt{\mathbb{E}\left[\|\mathcal{S}_3(x)\|^2 \right]} = \tilde{O}\left(d^{1.5} \right)$, and $r = \tilde{O}(\sqrt{d})$.

  See Corollary 1 for examples of functions that satisfy this bound, and thus, we can learn them by the proposed method.

- The coefficients $\lambda_j := \mathbb{E}\left[\sigma'''(z_j)\right] \cdot a_2(j)$, and $\tilde{\lambda}_j := \mathbb{E}\left[\sigma''(z_j)\right] \cdot a_2(j)$, $j \in [k]$, in (20) and (40) are non-zero.
- k random i.i.d. draws of frequencies in Equation (15) are linearly independent. Note that the draws are from Fourier magnitude distribution[7] $\|\omega\| \cdot |F(\omega)|$. For more discussions on this condition, see Section 7.2.1 and earlier explanations in this section. In the overcomplete regime, $(k > d)$, the linear independence property needs to hold for appropriate tensorizations of the frequency draws.

The above requirements on the number of samples n and parameter C_f depend on the parameters of the neural network A_1, a_2, b_1 and b_2. Note that there is also a dependence on these parameters through coefficients λ_j and $\tilde{\lambda}_j$. Since this is the non-realizable setting, these neural network parameters correspond to the neural networks that satisfy the approximation bound proposed in Theorem 4 and are generated via random draws from the frequency spectrum of the function $f(x)$.

The proposed bound on C_f in (18) is stricter when the number of hidden units k increases. This might seem counter-intuitive, since the approximation result in Theorem 4 suggests that increasing k leads to smaller approximation error. But, note that the approximation result in Theorem 4 does not consider efficient training of the neural network. The result in Theorem 5 also deals with the efficient estimation of the neural network. This imposes additional constraint on the parameter C_f such that when the number of neurons increases, the problem of learning the network weights is more challenging for the tensor method to resolve.

Theorem 5 (NN-LIFT guarantees: risk bound)**.** *Suppose the above conditions hold. Then the target function f is approximated by the neural network $\hat{f}$ which is learnt using NN-LIFT in Algorithm 1 satisfying w.h.p.*

$$\mathbb{E}_x[|f(x) - \hat{f}(x)|^2] \leq O(r^2 C_f^2) \cdot \left(\frac{1}{\sqrt{k}} + \delta_1\right)^2 + O(\epsilon^2),$$

where δ_τ is defined in (16)*. Recall $x \in B_r$, where $B_r := \{x : \|x\| \leq r\}$.*

The theorem is mainly proved by combining the estimation bound guarantees in Theorem 3, and the approximation bound results for neural networks provided in Theorem 4. But note that the approximation bound provided in Theorem 4 holds for a specific class of neural networks which are not generally recovered by the NN-LIFT algorithm. In addition, the estimation guarantees in Theorem 3 is for the realizable setting where the observations are the outputs of a fixed neural network, while in Theorem 5 we observe samples of arbitrary function $f(x)$. Thus, the approximation analysis in Theorem 4 can not be directly applied to Theorem 3. For this, we need additional assumptions to ensure the NN-LIFT algorithm recovers a neural network which is close to one of the neural networks that satisfy the approximation bound in Theorem 4. Therefore, we impose the bound on quantity C_f, and the full column rank assumption proposed in Theorem 4. See Appendix D for the complete proof of Theorem 5.

The above risk bound includes two terms. The first term $O(r^2 C_f^2) \cdot \left(\frac{1}{\sqrt{k}} + \delta_1\right)^2$ represents the approximation error on how the arbitrary function $f(x)$ with quantity C_f can be approximated by the neural network, whose weights are drawn from the Fourier magnitude distribution; see

[7]Note that it should be normalized to be a probability distribution as in (15).

Theorem 4 for the formal statement. From the definition of C_f in (13), this bound is weaker when the Fourier spectrum of target $f(x)$ has more energy in higher frequencies. This makes intuitive sense since it should be easier to approximate a function which is more smooth and has less fluctuations. The second term $O(\epsilon^2)$ is from estimation error for NN-LIFT algorithm, which is analyzed in Theorem 3. The polynomial factors for sample complexity in our estimation error are slightly worse than the bound provided in Barron (1994), but note that we provide an estimation method which is both computationally and statistically efficient, while the method in Barron (1994) is not computationally efficient. Thus, for the first time, we have a computationally efficient method with guaranteed risk bounds for training neural networks.

Discussion on δ_τ in the approximation bound: The approximation bound involves a term δ_τ which is a constant and does not shrink with increasing the neuron size k. Recall that δ_τ measures the distance between the unit step function $1_{\{z>0\}}(z)$ and the scaled sigmoidal function $\sigma(\tau z)$ (which is used in the neural network specified in (7)). We now provide the following two observations

The above risk bound is only provided for the case $\tau = 1$. We can generalize this result by imposing different constraint on the norm of columns of A_1 in (14). In general, if we impose $\|(A_1)_j\| = \tau, j \in [k]$, for some $\tau > 0$, then we have the approximation bound[8] $O(r^2 C_f^2) \cdot \left(\frac{1}{\sqrt{k}} + \delta_\tau \right)^2$. Note that $\delta_\tau \to 0$ when $\tau \to \infty$ (the scaled sigmoidal function $\sigma(\tau z)$ converges to the unit step function), and thus, this constant approximation error vanishes.

If the sigmoidal function is the unit step function as $\sigma(z) = 1_{\{z>0\}}(z)$, then $\delta_\tau = 0$ for all $\tau > 0$, and hence, there is no such constant approximation error.

6 Discussions and Extensions

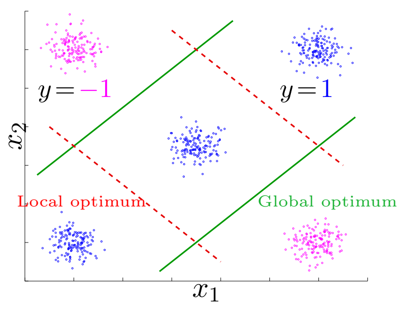
In this section, we provide additional discussions. We first propose a toy example contrasting the hardness of optimization problems backpropagation and tensor decomposition. We then discuss the generalization of learning guarantees to higher dimensional output, and also the continuous output case. We then discuss an alternative approach for estimating the low-dimensional parameters of the model.

6.1 Contrasting the loss surface of backpropagation with tensor decomposition

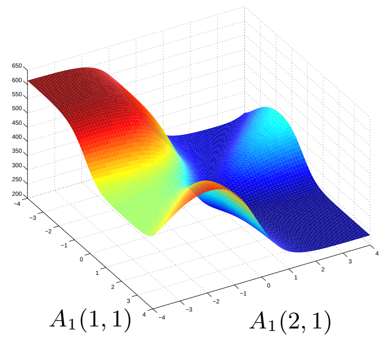
We discussed that the computational hardness of training a neural network is due to the non-convexity of the loss function, and thus, popular local search methods such as backpropagation can get stuck in spurious local optima. We now provide a toy example highlighting this issue, and contrast it with the tensor decomposition approach.

We consider a simple binary classification task shown in Figure 2.a, where blue and magneta data points correspond to two different classes. It is clear that these two classes can be classified by a mixture of two linear classifiers which are drawn as green solid lines in the figure. For this task, we consider a two-layer neural network with two hidden neurons. The loss surfaces for backpropagation and tensor decomposition are shown in Figures 2.b and 2.c, respectively. They are shown in terms

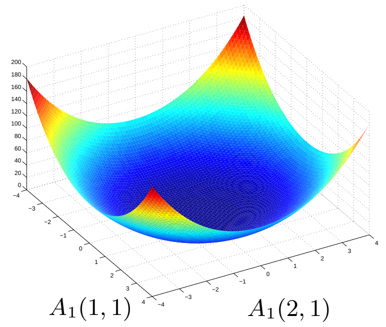
[8]Note that this change also needs some straightforward appropriate modifications in the algorithm.



(a) Classification setup



(b) Loss surface for backprop.



(c) Loss surface for tensor method

Figure 2: (a) Classification task: two colors correspond to binary labels. A two-layer neural network with two hidden neurons is used. Loss surface in terms of the first layer weights of one neuron (i.e., weights connecting the inputs to the neuron) is plotted while other parameters are fixed. (b) Loss surface for usual square loss objective has spurious local optima. In part (a), one of the spurious local optima is drawn as red dashed lines and the global optimum is drawn as green solid lines. (c) Loss surface for tensor factorization objective is free of spurious local optima.

of the weight parameters of inputs to the first neuron, i.e., the first column of matrix A_1, while the weight parameters to the second neuron are randomly drawn, and then fixed.

The stark contrast between the optimization landscape of tensor objective function, and the usual square loss objective used for backpropagation are observed, where even for a very simple classification task, backpropagation suffers from spurious local optima (one set of them is drawn as red dashed lines), which is not the case with tensor methods that is at least locally convex. This comparison highlights the algorithmic advantage of tensor decomposition compared to backpropagation in terms of the optimization they are performing.

6.2 Extensions to cases beyond binary classification

We earlier limited ourselves to the case where the output of neural network $\tilde{y} \in \{0,1\}$ is binary. These results can be easily extended to more complicated cases such as higher dimensional output (multi-label and multi-class), and also the continuous outputs (i.e., regression setting). In the rest of this section, we discuss about the necessary changes in the algorithm to adapt it for these cases.

In the multi-dimensional case, the output label $\tilde{y}$ is a vector generated as

$$\mathbb{E}[\tilde{y}|x] = A_2^\top \sigma(A_1^\top x + b_1) + b_2,$$

where the output is either discrete (multi-label and multi-class) or continuous. Recall that the algorithm includes three main parts: tensor decomposition, Fourier and ridge regression components.

Tensor decomposition: For the tensor decomposition part, we first form the empirical version of $\tilde{T} = \mathbb{E}\left[\tilde{y} \otimes \mathcal{S}_3(x)\right]$; note that $\otimes$ is used here (instead of scalar product used earlier) since $\tilde{y}$ is not a scalar anymore. By the properties of score function, this tensor has decomposition form

$$\tilde{T} = \mathbb{E}\left[\tilde{y} \otimes \mathcal{S}_3(x)\right] = \sum_{j\in[k]} \mathbb{E}\left[\sigma'''(z_j)\right] \cdot (A_2)^j \otimes (A_1)_j \otimes (A_1)_j \otimes (A_1)_j,$$

where $(A_2)^j$ denotes the j^{th} row of matrix A_2. This is proved similar to Lemma 6. The tensor $\tilde{T}$ is a fourth order tensor, and we contract the first mode by multiplying it with a random vector θ as $\tilde{T}(\theta, I, I, I)$ leading to the same form in (19) as

$$\tilde{T}(\theta, I, I, I) = \sum_{j \in [k]} \lambda_j \cdot (A_1)_j \otimes (A_1)_j \otimes (A_1)_j,$$

with λ_j changed to $\lambda_j = \mathbb{E}\left[\sigma'''(z_j)\right] \cdot \langle (A_2)^j, \theta \rangle$. Therefore, the same tensor decomposition guarantees in the binary case also hold here when the empirical version of $\tilde{T}(\theta, I, I, I)$ is the input to the algorithm.

Fourier method: Similar to the scalar case, we can use one of the entries of output to estimate the entries of b_1. There is an additional difference in the continuous case. Suppose that the output is generated as $\tilde{y} = \tilde{f}(x) + \eta$ where η is noise vector which is independent of input x. In this case, the parameter $\tilde{\zeta}_{\tilde{f}}$ corresponding to l^{th} entry of output $\tilde{y}_l$ is changed to $\tilde{\zeta}_{\tilde{f}} := \int_{\mathbb{R}^d} \tilde{f}(x)_l^2 dx + \int_{\mathbb{R}} \eta_l^2 dt$.

Ridge regression: The ridge regression method and analysis can be immediately generalized to non-scalar output by applying the method independently to different entries of output vector to recover different columns of matrix A_2 and different entries of vector b_2.

6.3 An alternative for estimating low-dimensional parameters

Once we have an estimate of the first layer weights A_1, we can greedily (i.e., incrementally) add neurons with the weight vectors $(A_1)_j$ for $j \in [k]$, and choose the bias $b_1(j)$ through grid search, and learn its contribution $a_2(j)$ for its final output. This is on the lines of the method proposed in Barron (1993), with one crucial difference that in our case, the first layer weights A_1 are already estimated by the tensor method. Barron (1993) proposes optimizing for each weight vector $(A_1)_j$ in d-dimensional space, whose computational complexity can scale exponentially in d in the worst case. But, in our setup here, since we have already estimated the high-dimensional parameters (i.e., the columns of A_1), we only need to estimate a few low dimensional parameters. For the new hidden unit indexed by j, these parameters include the bias from input layer to the neuron (i.e., $b_1(j)$), and the weight from the neuron to the output (i.e., $a_2(j)$). This makes the approach computationally tractable, and we can even use brute-force or exhaustive search to find the best parameters on a finite set and get guarantees akin to (Barron, 1993).

7 Proof Sketch

In this section, we provide key ideas for proving the main results in Theorems 3 and 4.

7.1 Estimation bound

The estimation bound is proposed in Theorem 3, and the complete proof is provided in Appendix C. Recall that NN-LIFT algorithm includes a tensor decomposition part for estimating A_1, a Fourier technique for estimating b_1, and a linear regression for estimating a_2, b_2. The application of linear regression in the last layer is immediately clear. In this section, we propose two main lemmas which clarify why the other methods are useful for estimating the unknown parameters A_1, b_1

in the realizable setting, where the label $\tilde{y}$ is generated by the neural network with the given architecture.

In the following lemma, we show how the cross-moment between label and score function as $\mathbb{E}[\tilde{y} \cdot \mathcal{S}_3(x)]$ leads to a tensor decomposition form for estimating weight matrix A_1.

Lemma 6. *For the two-layer neural network specified in* (7)*, we have*

$$\mathbb{E}\left[\tilde{y} \cdot \mathcal{S}_3(x)\right] = \sum_{j \in [k]} \lambda_j \cdot (A_1)_j \otimes (A_1)_j \otimes (A_1)_j, \tag{19}$$

where $(A_1)_j \in \mathbb{R}^d$ denotes the j-th column of A_1, and

$$\lambda_j = \mathbb{E}\left[\sigma'''(z_j)\right] \cdot a_2(j), \tag{20}$$

for vector $z := A_1^\top x + b_1$ as the input to the nonlinear operator $\sigma(\cdot)$.

This is proved by the main property of score functions as yielding differential operators, where for label-function $f(x) := \mathbb{E}[y|x]$, we have $\mathbb{E}[y \cdot \mathcal{S}_3(x)] = \mathbb{E}[\nabla_x^{(3)} f(x)]$ (Janzamin et al., 2014); see Section C.1 for a complete proof of the lemma. This lemma shows that by decomposing the cross-moment tensor $\mathbb{E}[\tilde{y} \cdot \mathcal{S}_3(x)]$, we can recover the columns of A_1.

We also exploit the phase of complex number v to estimate the bias vector b_1; see Procedure 2. The following lemma clarifies this. The perturbation analysis is provided in the appendix.

Lemma 7. *Let*

$$\tilde{v} := \frac{1}{n} \sum_{i \in [n]} \frac{\tilde{y}_i}{p(x_i)} e^{-j\langle \omega_i, x_i \rangle}. \tag{21}$$

Notice this is a realizable of v in Procedure 2 where the output corresponds to a neural network $\tilde{y}$. If ω_i's are uniformly i.i.d. drawn from set Ω_l, then $\tilde{v}$ has mean (which is computed over x, $\tilde{y}$ and ω)

$$\mathbb{E}[\tilde{v}] = \frac{1}{|\Omega_l|} \Sigma\left(\frac{1}{2}\right) a_2(l) e^{j\pi b_1(l)}, \tag{22}$$

where $|\Omega_l|$ denotes the surface area of $d-1$ dimensional manifold Ω_l, and $\Sigma(\cdot)$ denotes the Fourier transform of $\sigma(\cdot)$.

This lemma is proved in Appendix C.2.

7.2 Approximation bound

We exploit the approximation bound argued in Barron (1993) provided in Theorem 4. We first discuss his main result arguing an approximation bound $O(r^2 C_f^2/k)$ for a function $f(x)$ with bounded parameter C_f; see (13) for the definition of C_f. Note that this corresponds to the first term in the approximation error proposed in Theorem 4. For this result, Barron (1993) does not consider any bound on the parameters of first layer A_1 and b_1. He then provides a refinement of this result where he also bounds the parameters of neural network as we also do in (14). This leads to the additional term involving δ_τ in the approximation error as seen in Theorem 4. Note that bounding the parameters of neural network is also useful in learning these parameters with computationally efficient algorithms since it limits the searching space for training these parameters. We now provide the main ideas of proving these bounds as follows.

7.2.1 No bounds on the parameters of the neural network

We first provide the proof outline when there is no additional constraints on the parameters of neural network; see set G defined in (24), and compare it with the form we use in (14) where there are additional bounds. In this case, Barron (1993) argues approximation bound $O(r^2C_f^2/k)$ which is proved based on two main results. The first result says that if a function f is in the closure of the convex hull of a set G in a Hilbert space, then for every $k \geq 1$, there is an f_k as the convex combination of k points in G such that

$$\mathbb{E}[|f - f_k|^2] \leq \frac{c'}{k}, \tag{23}$$

for any constant c' satisfying some lower bound related to the properties of set G and function f; see Lemma 1 in Barron (1993) for the precise statement and the proof of this result.

The second part of the proof is to argue that arbitrary function $f \in \Gamma$ (where Γ denotes the set of functions with bounded C_f) is in the closure of the convex hull of sigmoidal functions

$$G := \big\{\gamma\sigma\big(\langle\alpha, x\rangle + \beta\big) : \alpha \in \mathbb{R}^d, \beta \in \mathbb{R}, |\gamma| \leq 2C\big\}. \tag{24}$$

Barron (1993) proves this result by arguing the following chain of inclusions as

$$\Gamma \subset \operatorname{cl} G_{\cos} \subset \operatorname{cl} G_{\text{step}} \subset \operatorname{cl} G,$$

where $\operatorname{cl} G$ denotes the closure of set G, and sets $G_{\cos}$ and G_{step} respectively denote set of some sinusoidal and step functions. See Theorem 2 in Barron (1993) for the precise statement and the proof of this result.

Random frequency draws from Fourier magnitude distribution: Recall from Section 5 that the columns of weight matrix A_1 are the normalized version of random frequencies drawn from Fourier magnitude distribution $\|\omega\| \cdot |F(\omega)|$; see Equation (15). This connection is along the proof of relation $\Gamma \subset \operatorname{cl} G_{\cos}$ that we recap here; see proof of Lemma 2 in Barron (1993) for more details. By expanding the Fourier transform as magnitude and phase parts $F(\omega) = e^{j\theta(\omega)}|F(\omega)|$, we have

$$\overline{f}(x) := f(x) - f(0) = \int g(x, \omega)\Lambda(d\omega), \tag{25}$$

where

$$\Lambda(\omega) := \|\omega\| \cdot |F(\omega)|/C_f \tag{26}$$

is the normalized Fourier magnitude distribution (as a probability distribution) weighted by the norm of frequency vector, and

$$g(x, \omega) := \frac{C_f}{\|\omega\|}\left(\cos(\langle\omega, x\rangle + \theta(\omega)) - \cos(\theta(\omega))\right).$$

The integral in (25) is an infinite convex combination of functions in the class

$$G_{\cos} := \left\{\frac{\gamma}{\|\omega\|}\left(\cos(\langle\omega, x\rangle + \beta) - \cos(\beta)\right) : \omega \neq 0, |\gamma| \leq C, \beta \in \mathbb{R}\right\}.$$

Now if $\omega_1, \omega_2, \ldots, \omega_k$ is a random sample of k points independently drawn from Fourier magnitude distribution Λ, then by Fubini's Theorem, we have

$$\mathbb{E} \int_{B_r} \left(f(x) - \frac{1}{k} \sum_{j \in [k]} g(x, \omega_j) \right)^2 \mu(dx) \leq \frac{C^2}{k},$$

where $\mu(\cdot)$ is the probability measure for x. This shows function $\overline{f}$ is in the convex hull of $G_{\cos}$. Note that the bound $\frac{C^2}{k}$ complies the bound in (23).

7.2.2 Bounding the parameters of the neural network

Barron (1993) then imposes additional bounds on the weights of first layer, considering the following class of sigmoidal functions as

$$G_\tau := \left\{ \gamma \sigma\big(\tau(\langle \alpha, x \rangle + \beta)\big) : \|\alpha\| \leq 1, |\beta| \leq 1, |\gamma| \leq 2C \right\}. \tag{27}$$

Note that the approximation proposed in (14) is a convex combination of k points in (27) with $\tau = 1$. Barron (1993) concludes Theorem 4 by the following lemma.

Lemma 8 (Lemma 5 in Barron (1993)). *If g is a function on $[-1, 1]$ with derivative bounded*[9] *by a constant C, then for every $\tau > 0$, we have*

$$\inf_{g_\tau \in \operatorname{cl} G_\tau} \sup_{|z| \leq \tau} |g(z) - g_\tau(z)| \leq 2C\delta_\tau.$$

Finally Theorem 4 is proved by applying triangle inequality to bounds argued in the above two cases.

8 Conclusion

We have proposed a novel algorithm based on tensor decomposition for training two-layer neural networks. This is a computationally efficient method with guaranteed risk bounds with respect to the target function under polynomial sample complexity in the input and neuron dimensions. The tensor method is embarrassingly parallel and has a parallel time computational complexity which is logarithmic in input dimension which is comparable with parallel stochastic backpropagation. There are number of open problems to consider in future. Extending this framework to a multi-layer network is of great interest. Exploring the score function framework to train other discriminative models is also interesting.

Acknowledgements

We thank Ben Recht for pointing out to us the Barron's work on approximation bounds for neural networks (Barron, 1993), and thank Arthur Gretton for discussion about estimation of score function. We also acknowledge fruitful discussion with Roi Weiss about the presentation of proof

[9] Note that the condition on having bounded derivative does not rule out cases such as step function as the sigmoidal function. This is because similar to the analysis for the main case (no bounds on the weights), we first argue that function f is in the closure of functions in $G_{\cos}$ which are univariate functions with bounded derivative.

of Theorem 5 on combining estimation and approximation bounds, and his detailed editorial comments about the preliminary version of the draft. We thank Peter Bartlett for detailed discussions and pointing us to several classical results on neural networks. We are very thankful for Andrew Barron for detailed discussion and for encouraging us to explore alternate incremental training methods for estimating remaining parameters after the tensor decomposition step and we have added this discussion to the paper. We thank Daniel Hsu for discussion on random design analysis of ridge regression. We thank Percy Liang for discussion about score function.

M. Janzamin is supported by NSF BIGDATA award FG16455. H. Sedghi is supported by NSF Career award FG15890. A. Anandkumar is supported in part by Microsoft Faculty Fellowship, NSF Career award CCF-1254106, ONR award N00014-14-1-0665, ARO YIP award W911NF-13-1-0084, and AFOSR YIP award FA9550-15-1-0221.

A Tensor Notation

In this Section, we provide the additional tensor notation required for the analysis provided in the supplementary material.

Multilinear form: The multilinear form for a tensor $T \in \mathbb{R}^{q_1 \times q_2 \times q_3}$ is defined as follows. Consider matrices $M_r \in \mathbb{R}^{q_r \times p_r}, r \in \{1,2,3\}$. Then tensor $T(M_1, M_2, M_3) \in \mathbb{R}^{p_1 \times p_2 \times p_3}$ is defined as

$$T(M_1, M_2, M_3) := \sum_{j_1 \in [q_1]} \sum_{j_2 \in [q_2]} \sum_{j_3 \in [q_3]} T_{j_1, j_2, j_3} \cdot M_1(j_1, :) \otimes M_2(j_2, :) \otimes M_3(j_3, :). \tag{28}$$

As a simpler case, for vectors $u, v, w \in \mathbb{R}^d$, we have [10]

$$T(I, v, w) := \sum_{j,l \in [d]} v_j w_l T(:, j, l) \ \in \mathbb{R}^d, \tag{29}$$

which is a multilinear combination of the tensor mode-1 fibers.

B Details of Tensor Decomposition Algorithm

The goal of tensor decomposition algorithm is to recover the rank-1 components of tensor; refer to Equation (3) for the notion of tensor rank and its rank-1 components. We exploit the tensor decomposition algorithm proposed in (Anandkumar et al., 2014b,c). Figure 3 depicts the flowchart of this method where the corresponding algorithms and procedures are also specified. Similarly, Algorithm 4 states the high-level steps of tensor decomposition algorithm. The main step of the tensor decomposition method is the *tensor power iteration* which is the generalization of matrix power iteration to 3rd order tensors. The tensor power iteration is given by

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

where $u \in \mathbb{R}^d, T(I, u, u) := \sum_{j,l \in [d]} u_j u_l T(:, j, l) \in \mathbb{R}^d$ is a *multilinear* combination of tensor *fibers.* Note that tensor fibers are the vectors which are derived by fixing all the indices of the tensor

[10]Compare with the matrix case where for $M \in \mathbb{R}^{d \times d}$, we have $M(I, u) = Mu := \sum_{j \in [d]} u_j M(:, j)$.

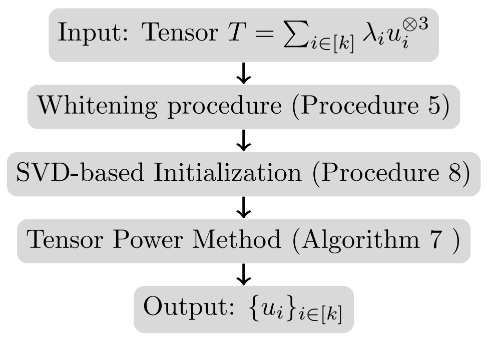

Input: Tensor $T = \sum_{i\in[k]} \lambda_i u_i^{\otimes 3}$
Whitening procedure (Procedure 5)
SVD-based Initialization (Procedure 8)
Tensor Power Method (Algorithm 7)
Output: $\{u_i\}_{i\in[k]}$

Figure 3: Overview of tensor decomposition algorithm for third order tensor (without tensorization).

Algorithm 4 Tensor Decomposition Algorithm Setup

```
input symmetric tensor $T$.
 1: if Whitening then
 2:     Calculate $T$ =Whiten($T$); see Procedure 5.
 3: else if Tensorizing then
 4:     Tensorize the input tensor.
 5:     Calculate $T$ =Whiten($T$); see Procedure 5.
 6: end if
 7: for $j = 1$ to $k$ do
 8:     $(v_j, \mu_j, T)$ = tensor power decomposition($T$); see Algorithm 7.
 9: end for
10: $(A_1)_j$ = Un-whiten$(v_j), j \in [k]$; see Procedure 6.
11: return $\{(A_1)_j\}_{j\in[k]}$.
```

except one of them, e.g., $T(:, j, l)$ in the above expansion. The initialization for different runs of tensor power iteration is performed by the SVD-based technique proposed in Procedure 8. This helps to initialize non-convex tensor power iteration with good initialization vectors. The whitening preprocessing is applied to orthogonalize the components of input tensor. Note that the convergence guarantees of tensor power iteration for orthogonal tensor decomposition have been developed in the literature (Zhang and Golub, 2001; Anandkumar et al., 2014b).

The tensorization step works as follows.

Tensorization: The tensorizing step is applied when we want to decompose overcomplete tensors where the rank k is larger than the dimension d. For instance, for getting rank up to $k = O(d^2)$, we first form the 6th order input tensor with decomposition as

$$T = \sum_{j\in[k]} \lambda_j a_j^{\otimes 6} \in \bigotimes^{6} \mathbb{R}^d.$$

Given T, we form the 3rd order tensor $\tilde{T} \in \bigotimes^3 \mathbb{R}^{d^2}$ which is the tensorization of T such that

$$\tilde{T}\big(i_2 + d(i_1 - 1), j_2 + d(j_1 - 1), l_2 + d(l_1 - 1)\big) := T(i_1, i_2, j_1, j_2, l_1, l_2). \qquad (30)$$

Procedure 5 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 $T$ (see Section C.1.1 for more discussions):
     • Option 1: constructed using second order score function; see Equation (39).
     • Option 2: computed as $M_2 := T(I, I, \theta) \in \mathbb{R}^{d\times d}$, where $\theta \sim \mathcal{N}(0, I_d)$ is a random standard
       Gaussian vector.
2: Compute the rank-k SVD, $M_2 = U\,\mathrm{Diag}(\gamma)U^\top$, where $U \in \mathbb{R}^{d\times k}$ and $\gamma \in \mathbb{R}^k$.
3: Compute the whitening matrix $W := U\,\mathrm{Diag}(\gamma^{-1/2}) \in \mathbb{R}^{d\times k}$.
4: return $T(W, W, W) \in \mathbb{R}^{k\times k\times k}$.
```

Procedure 6 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 5, and let $\tilde{\lambda}_j, j \in [k]$ denote
   the corresponding coefficients as $M_2 = A_1\,\mathrm{Diag}(\tilde{\lambda})A_1^\top$; see (39).
2: Compute the rank-k SVD, $M_2 = U\,\mathrm{Diag}(\gamma)U^\top$, where $U \in \mathbb{R}^{d\times k}$ and $\gamma \in \mathbb{R}^k$.
3: Compute
     $$(A_1)_j = \frac{1}{\sqrt{\tilde{\lambda}_j}} U\,\mathrm{Diag}(\gamma^{1/2})v_j, \quad j \in [k].$$
4: return $\{(A_1)_j\}_{j\in[k]}$.
```

This leads to $\tilde{T}$ having decomposition

$$\tilde{T} = \sum_{j\in[k]} \lambda_j (a_j \odot a_j)^{\otimes 3}.$$

We then apply the tensor decomposition algorithm to this new tensor $\tilde{T}$. This now clarifies why the full column rank condition is applied to the columns of $A \odot A = [a_1 \odot a_1 \cdots a_k \odot a_k]$. Similarly, we can perform higher order tensorizations leading to more overcomplete models by exploiting initial higher order tensor T; see also Remark 3.

Efficient implementation of tensor decomposition given samples: The main update steps in the tensor decomposition algorithm is the tensor power iteration for which a multilinear operation is performed on tensor T. However, the tensor is not available beforehand, and needs to be estimated using the samples (as in Algorithm 1 in the main text). Computing and storing the tensor can be enormously expensive for high-dimensional problems. But, it is essential to note that since we can form a factor form of tensor T using the samples and other parameters in the model, we can manipulate the samples directly to perform the power update as *multi-linear* operations without explicitly forming the tensor. This leads to efficient computational complexity. See (Anandkumar et al., 2014a) for details on these implicit update forms.

Algorithm 7 Robust tensor power method (Anandkumar et al., 2014b)

```
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 in Procedure 8.
3:     for $t = 1$ to $N$ do
4:         Compute power iteration update
               $$\widehat{v}_t^{(\tau)} := \frac{\tilde{T}(I, \widehat{v}_{t-1}^{(\tau)}, \widehat{v}_{t-1}^{(\tau)})}{\|\tilde{T}(I, \widehat{v}_{t-1}^{(\tau)}, \widehat{v}_{t-1}^{(\tau)})\|} \qquad (31)$$
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 (31) 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 $\tilde{T} - \hat{\mu} \cdot \hat{v}^{\otimes 3}$.
```

Procedure 8 SVD-based initialization (Anandkumar et al., 2014c)

```
input Tensor $T \in \mathbb{R}^{d' \times d' \times d'}$.
1: for $\tau = 1$ to $\log(1/\hat{\delta})$ do
2:     Draw a random standard Gaussian vector $\theta^{(\tau)} \sim \mathcal{N}(0, I_{d'})$.
3:     Compute $u_1^{(\tau)}$ as the top left singular vector of $T(I, I, \theta^{(\tau)}) \in \mathbb{R}^{d' \times d'}$.
4: end for
5: $\widehat{v}_0 \leftarrow \max_{\tau \in [\log(1/\hat{\delta})]} \left(u_1^{(\tau)}\right)_{\min}$.
6: return $\widehat{v}_0$.
```

C Proof of Theorem 3

Proof of Theorem 3 includes three main pieces which is about arguing the recovery guarantees of three different parts of the algorithm: tensor decomposition, Fourier method, and linear regression. As the first piece, we show that the tensor decomposition algorithm for estimating weight matrix A_1 (see Algorithm 1 for the details) recovers it with the desired error. In the second part, we analyze the performance of Fourier technique for estimating bias vector b_1 (see Algorithm 1 and Procedure 2 for the details) proving the error in the recovery is small. Finally as the last step, the ridge regression is analyzed to ensure that the parameters of last layer of the neural network are well estimated leading to the estimation of overall function $\tilde{f}(x)$. We now provide the analysis of these three parts.

C.1 Tensor decomposition guarantees

We first provide a short proof for Lemma 6 which shows how the rank-1 components of third order tensor $\mathbb{E}[\tilde{y} \cdot \mathcal{S}_3(x)]$ are the columns of weight matrix A_1.

Proof of Lemma 6: It is shown by Janzamin et al. (2014) that the score function yields differential operator such that for label-function $f(x) := \mathbb{E}[y|x]$, we have

$$\mathbb{E}[y \cdot \mathcal{S}_3(x)] = \mathbb{E}[\nabla_x^{(3)} f(x)].$$

Applying this property to the form of label function $f(x)$ in (7) denoted by $\tilde{f}(x)$, we have

$$\mathbb{E}\left[\tilde{y} \cdot \mathcal{S}_3(x)\right] = \mathbb{E}[\sigma'''(\cdot)(a_2, A_1^\top, A_1^\top, A_1^\top)],$$

where $\sigma'''(\cdot)$ denotes the third order derivative of element-wise function $\sigma(z) : \mathbb{R}^k \to \mathbb{R}^k$. More concretely, with slightly abuse of notation, $\sigma'''(z) \in \mathbb{R}^{k\times k\times k\times k}$ is a diagonal 4th order tensor with its j-th diagonal entry equal to $\frac{\partial^3 \sigma(z_j)}{\partial z_j^3} : \mathbb{R} \to \mathbb{R}$. Here two properties are used to compute the third order derivative $\nabla_x^{(3)} \tilde{f}(x)$ on the R.H.S. of above equation as follows. 1) We apply chain rule to take the derivatives which generates a new factor of A_1 for each derivative. Since we take 3rd order derivative, we have 3 factors of A_1. 2) The linearity of next layers leads to the derivatives from them being vanished, and thus, we only have the above term as the derivative. Expanding the above multilinear form finishes the proof; see (28) for the definition of multilinear form. □

We now provide the recovery guarantees of weight matrix A_1 through tensor decomposition as follows.

Lemma 9. *Among the conditions for Theorem 3, consider the rank constraint on A_1, and the non-vanishing assumption on coefficients λ_j's. Let the whitening to be performed using empirical version of second order score function as specified in* (39)*, and assume the coefficients $\tilde{\lambda}_j$'s do not vanish. Suppose the sample complexity*

$$\begin{aligned} n \geq \max\Bigg\{ &\tilde{O}\left(\tilde{y}_{\max}^2 \mathbb{E}\left[\left\|M_3(x)M_3^\top(x)\right\|\right] \frac{\tilde{\lambda}_{\max}^4}{\tilde{\lambda}_{\min}^4} \frac{s_{\max}^2(A_1)}{\lambda_{\min}^2 \cdot s_{\min}^6(A_1)} \cdot \frac{1}{\tilde{\epsilon}_1^2}\right), \\ &\tilde{O}\left(\tilde{y}_{\max}^2 \cdot \mathbb{E}\left[\left\|M_3(x)M_3^\top(x)\right\|\right] \cdot \left(\frac{\tilde{\lambda}_{\max}}{\tilde{\lambda}_{\min}}\right)^3 \frac{1}{\lambda_{\min}^2 \cdot s_{\min}^6(A_1)} \cdot k\right), \\ &\tilde{O}\left(\tilde{y}_{\max}^2 \cdot \frac{\mathbb{E}\left[\left\|\mathcal{S}_2(x)\mathcal{S}_2^\top(x)\right\|\right]^{3/2}}{\mathbb{E}\left[\left\|M_3(x)M_3^\top(x)\right\|\right]^{1/2}} \cdot \frac{1}{\tilde{\lambda}_{\min}^2 \cdot s_{\min}^3(A_1)}\right)\Bigg\}, \end{aligned} \tag{32}$$

holds, where $M_3(x) \in \mathbb{R}^{d\times d^2}$ denotes the matricization of score function tensor $\mathcal{S}_3(x) \in \mathbb{R}^{d\times d\times d}$; see (1) *for the definition of matricization. Then the estimate $\hat{A}_1$ by NN-LIFT Algorithm 1 satisfies w.h.p.*

$$\min_{z\in\{\pm 1\}} \|(A_1)_j - z \cdot (\hat{A}_1)_j\| \leq \tilde{O}\left(\tilde{\epsilon}_1\right), \quad j \in [k],$$

where the recovery guarantee is up to the permutation of columns of A_1.

Remark 6 (Sign ambiguity)**.** *We observe that in addition to the permutation ambiguity in the recovery guarantees, there is also a sign ambiguity issue in recovering the columns of matrix A_1 through the decomposition of third order tensor in* (19)*. This is because the sign of $(A_1)_j$ and coefficient λ_j can both change while the overall tensor is still fixed. Note that the coefficient λ_j can be positive or negative. According to the Fourier method for estimating b_1, mis-calculating the sign of $(A_1)_j$ also leads to sign of $b_1(j)$ recovered in the opposite manner. In other words, the recovered sign of the bias $b_1(j)$ is consistent with the recovered sign of $(A_1)_j$.*

Recall we assume that the nonlinear activating function $\sigma(z)$ satisfies the property such that $\sigma(z) = 1 - \sigma(-z)$. Many popular activating functions such as step function, sigmoid function and tanh function satisfy this property. Given this property, the sign ambiguity in parameters A_1 and b_1 which leads to opposite sign in input z to the activating function $\sigma(\cdot)$ can be now compensated by the sign of a_2 and value of b_2, which is recovered through least squares.

Proof of Lemma 9: From Lemma 6, we know that the exact cross-moment $\tilde{T} = \mathbb{E}[\tilde{y} \cdot \mathcal{S}_3(x)]$ has rank-one components as columns of matrix A_1; see Equation (19) for the tensor decomposition form. We apply a tensor decomposition method in NN-LIFT to estimate the columns of A_1. We employ noisy tensor decomposition guarantees in Anandkumar et al. (2014c). They show that when the perturbation tensor is small, the tensor power iteration initialized by the SVD-based Procedure 8 recovers the rank-1 components up to some small error. We also analyze the whitening step and combine it with this result leading to Lemma 10.

Let us now characterize the perturbation matrix and tensor. By Lemma 6, the CP decomposition form is given by $\tilde{T} = \mathbb{E}[\tilde{y} \cdot \mathcal{S}_3(x)]$, and thus, the perturbation tensor is written as

$$E := \tilde{T} - \widehat{T} = \mathbb{E}[\tilde{y} \cdot \mathcal{S}_3(x)] - \frac{1}{n} \sum_{i \in [n]} \tilde{y}_i \cdot \mathcal{S}_3(x_i), \tag{33}$$

where $\widehat{T} = \frac{1}{n} \sum_{i \in [n]} \tilde{y}_i \cdot \mathcal{S}_3(x_i)$ is the empirical form used in NN-LIFT Algorithm 1. Notice that in the realizable setting, the neural network output $\tilde{y}$ is observed and thus, it is used in forming the empirical tensor. Similarly, the perturbation of second order moment $\tilde{M}_2 = \mathbb{E}[\tilde{y} \cdot \mathcal{S}_2(x)]$ is given by

$$E_2 := \tilde{M}_2 - \widehat{M}_2 = \mathbb{E}[\tilde{y} \cdot \mathcal{S}_2(x)] - \frac{1}{n} \sum_{i \in [n]} \tilde{y}_i \cdot \mathcal{S}_2(x_i). \tag{34}$$

In order to bound $\|E\|$, we matricize it to apply matrix Bernstein's inequality. We have the matricized version as

$$\tilde{E} := \mathbb{E}[\tilde{y} \cdot M_3(x)] - \frac{1}{n} \sum_{i \in [n]} \tilde{y}_i \cdot M_3(x_i) = \sum_{i \in [n]} \frac{1}{n} \Big(\mathbb{E}[\tilde{y} \cdot M_3(x)] - \tilde{y}_i \cdot M_3(x_i) \Big),$$

where $M_3(x) \in \mathbb{R}^{d \times d^2}$ is the matricization of $\mathcal{S}_3(x) \in \mathbb{R}^{d \times d \times d}$; see (1) for the definition of matricization. Now the norm of $\tilde{E}$ can be bounded by the matrix Bernstein's inequality. The norm of each (centered) random variable inside the summation is bounded as $\frac{\tilde{y}_{\max}}{n} \mathbb{E}[\|M_3(x)\|]$, where $\tilde{y}_{\max}$ is the bound on $|\tilde{y}|$. The variance term is also bounded as

$$\frac{1}{n^2} \Big\| \sum_{i \in [n]} \mathbb{E}\left[\tilde{y}_i^2 \cdot M_3(x_i) M_3^\top(x_i) \right] \Big\| \leq \frac{1}{n} \tilde{y}_{\max}^2 \mathbb{E}\left[\left\| M_3(x) M_3^\top(x) \right\| \right].$$

Applying matrix Bernstein's inequality, we have w.h.p.

$$\|E\| \leq \|\tilde{E}\| \leq \tilde{O}\left(\frac{\tilde{y}_{\max}}{\sqrt{n}} \sqrt{\mathbb{E}\left[\left\| M_3(x) M_3^\top(x) \right\| \right]} \right). \tag{35}$$

For the second order perturbation E_2, it is already a matrix, and by applying matrix Bernstein's inequality, we similarly argue that w.h.p.

$$\|E_2\| \leq \tilde{O}\left(\frac{\tilde{y}_{\max}}{\sqrt{n}} \sqrt{\mathbb{E}\left[\left\| \mathcal{S}_2(x) \mathcal{S}_2^\top(x) \right\| \right]} \right). \tag{36}$$

There is one more remaining piece to complete the proof of tensor decomposition part. The analysis in Anandkumar et al. (2014c) does not involve any whitening step, and thus, we need to adapt the perturbation analysis of Anandkumar et al. (2014c) to our additional whitening procedure. This is done in Lemma 10. In the final recovery bound (46) in Lemma 10, there are two terms; one involving $\|E\|$, and the other involving $\|E_2\|$. We first impose a bound on sample complexity such that the bound involving $\|E\|$ dominates the bound involving $\|E_2\|$ as follows. Considering the bounds on $\|E\|$ and $\|E_2\|$ in (35) and (36), and imposing the lower bound on the number of samples (third bound stated in the lemma) as

$$n \geq \tilde{O}\left(\tilde{y}_{\max}^2 \cdot \frac{\mathbb{E}\left[\left\|\mathcal{S}_2(x)\mathcal{S}_2^\top(x)\right\|\right]^{3/2}}{\mathbb{E}\left[\left\|M_3(x)M_3^\top(x)\right\|\right]^{1/2}} \cdot \frac{1}{\tilde{\lambda}_{\min}^2 \cdot s_{\min}^3(A_1)}\right),$$

leads to this goal. By doing this, we do not need to impose the bound on $\|E_2\|$ anymore, and applying the perturbation bound in (35) to the required bound on $\|E\|$ in Lemma 10 leads to sample complexity bound (second bound stated in the lemma)

$$n \geq \tilde{O}\left(\tilde{y}_{\max}^2 \cdot \mathbb{E}\left[\left\|M_3(x)M_3^\top(x)\right\|\right] \cdot \left(\frac{\tilde{\lambda}_{\max}}{\tilde{\lambda}_{\min}}\right)^3 \frac{1}{\tilde{\lambda}_{\min}^2 \cdot s_{\min}^6(A_1)} \cdot k\right).$$

Finally, applying the result of Lemma 10, we have the column-wise error guarantees (up to permutation)

$$\|(A_1)_j - (\hat{A}_1)_j\| \leq \tilde{O}\left(\frac{s_{\max}(A_1)}{\lambda_{\min}} \frac{\tilde{\lambda}_{\max}^2}{\sqrt{\tilde{\lambda}_{\min}}} \frac{\tilde{y}_{\max}}{\tilde{\lambda}_{\min}^{1.5} \cdot s_{\min}^3(A_1)} \frac{\sqrt{\mathbb{E}\left[\left\|M_3(x)M_3^\top(x)\right\|\right]}}{\sqrt{n}}\right) \leq \tilde{O}\left(\tilde{\epsilon}_1\right),$$

where in the first inequality we also substituted the bound on $\|E\|$ in (35), and the first bound on n stated in the lemma is used in the last inequality. □

C.1.1 Whitening analysis

The perturbation analysis of proposed tensor decomposition method in Algorithm 7 with the corresponding SVD-based initialization in Procedure 8 is provided in Anandkumar et al. (2014c). But, they do not consider the effect of whitening proposed in Procedures 5 and 6. Thus, we need to adapt the perturbation analysis of Anandkumar et al. (2014c) when the whitening procedure is incorporated. We perform it in this section.

We first elaborate on the whitening step, and analyze how the proposed Procedure 5 works. We then analyze the inversion of whitening operator showing how the components in the whitened space are translated back to the original space as stated in Procedure 6. We finally provide the perturbation analysis of whitening step when estimations of moments are given.

Whitening procedure: Consider second order moment $\tilde{M}_2$ which is used to whiten third order tensor

$$\tilde{T} = \sum_{j\in[k]} \lambda_j \cdot (A_1)_j \otimes (A_1)_j \otimes (A_1)_j \tag{37}$$

in Procedure 5. It is constructed such that it has the same decomposition form as target tensor $\tilde{T}$, i.e., we have

$$\tilde{M}_2 = \sum_{j\in[k]} \tilde{\lambda}_j \cdot (A_1)_j \otimes (A_1)_j. \tag{38}$$

We propose two options for constructing $\tilde{M}_2$ in Procedure 5. First option is to use second order score function and construct $\tilde{M}_2 := \mathbb{E}\left[\tilde{y}\cdot \mathcal{S}_2(x)\right]$ for which we have

$$\tilde{M}_2 := \mathbb{E}\left[\tilde{y}\cdot \mathcal{S}_2(x)\right] = \sum_{j\in[k]} \tilde{\lambda}_j \cdot (A_1)_j \otimes (A_1)_j, \tag{39}$$

where

$$\tilde{\lambda}_j = \mathbb{E}\left[\sigma''(z_j)\right]\cdot a_2(j), \tag{40}$$

for vector $z := A_1^\top x + b_1$ as the input to the nonlinear operator $\sigma(\cdot)$. This is proved similar to Lemma 6. Second option leads to the same form for $\tilde{M}_2$ as (38) with coefficient modified as $\tilde{\lambda}_j = \lambda_j \cdot \langle (A_1)_j, \theta\rangle$.

Let matrix $W \in \mathbb{R}^{d\times k}$ denote the whitening matrix in the noiseless case, i.e., the whitening matrix W in Procedure 5 is constructed such that $W^\top \tilde{M}_2 W = I_k$. Applying whitening matrix W to the noiseless tensor $\tilde{T} = \sum_{j\in[k]} \lambda_j \cdot (A_1)_j \otimes (A_1)_j \otimes (A_1)_j$, we have

$$\tilde{T}(W,W,W) = \sum_{j\in[k]} \lambda_j \left(W^\top (A_1)_j\right)^{\otimes 3} = \sum_{j\in[k]} \frac{\lambda_j}{\tilde{\lambda}_j^{3/2}} \left(W^\top (A_1)_j \sqrt{\tilde{\lambda}_j}\right)^{\otimes 3} = \sum_{j\in[k]} \mu_j v_j^{\otimes 3}, \tag{41}$$

where we define

$$\mu_j := \frac{\lambda_j}{\tilde{\lambda}_j^{3/2}}, \quad v_j := W^\top (A_1)_j \sqrt{\tilde{\lambda}_j}, \quad j\in[k], \tag{42}$$

in the last equality. Let $V := [v_1\ v_2\ \cdots\ v_k] \in \mathbb{R}^{k\times k}$ denote the factor matrix for $\tilde{T}(W,W,W)$. We have

$$V := W^\top A_1 \operatorname{Diag}(\tilde{\lambda}^{1/2}), \tag{43}$$

and thus,

$$VV^\top = W^\top A_1 \operatorname{Diag}(\tilde{\lambda}) A_1^\top W = W^\top \tilde{M}_2 W = I_k.$$

Since V is a square matrix, it is also concluded that $V^\top V = I_k$, and therefore, tensor $\tilde{T}(W,W,W)$ is whitened such that the rank-1 components v_j's form an orthonormal basis. This discussion clarifies how the whitening procedure works.

Inversion of the whitening procedure: Let us also analyze the inversion procedure on how to transform v_j's to $(A_1)_j$'s. The main step is stated in Procedure 6. According to whitening Procedure 5, let $\tilde{M}_2 = U \operatorname{Diag}(\gamma) U^\top$, $U\in\mathbb{R}^{d\times k}$, $\gamma\in\mathbb{R}^k$, denote the rank-k SVD of $\tilde{M}_2$. Substituting whitening matrix $W := U\operatorname{Diag}(\gamma^{-1/2})$ in (43), and multiplying $U\operatorname{Diag}(\gamma^{1/2})$ from left, we have

$$U\operatorname{Diag}(\gamma^{1/2}) V = UU^\top A_1 \operatorname{Diag}(\tilde{\lambda}^{1/2}).$$

Since the column spans of $A_1 \in \mathbb{R}^{d\times k}$ and $U\in\mathbb{R}^{d\times k}$ are the same (given their relations to $\tilde{M}_2$), A_1 is a fixed point for the projection operator on the subspace spanned by the columns of U.

This projector operator is $UU^\top$ (since columns of U form an orthonormal basis), and therefore, $UU^\top A_1 = A_1$. Applying this to the above equation, we have

$$A_1 = U \operatorname{Diag}(\gamma^{1/2}) V \operatorname{Diag}(\tilde{\lambda}^{-1/2}),$$

i.e.,

$$(A_1)_j = \frac{1}{\sqrt{\tilde{\lambda}_j}} U \operatorname{Diag}(\gamma^{1/2}) v_j, \quad j \in [k]. \tag{44}$$

The above discussions describe the details of whitening and unwhitening procedures. We now provide the guarantees of tensor decomposition given noisy versions of moments $\tilde{M}_2$ and $\tilde{T}$.

Lemma 10. *Let $\widehat{M}_2 = \tilde{M}_2 - E_2$ and $\widehat{T} = \tilde{T} - E$ respectively denote the noisy versions of*

$$\tilde{M}_2 = \sum_{j \in [k]} \tilde{\lambda}_j \cdot (A_1)_j \otimes (A_1)_j, \quad \tilde{T} = \sum_{j \in [k]} \lambda_j \cdot (A_1)_j \otimes (A_1)_j \otimes (A_1)_j. \tag{45}$$

Assume the second and third order perturbations satisfy the bounds

$$\|E_2\| \leq \tilde{O}\left(\lambda_{\min}^{1/3} \frac{\tilde{\lambda}_{\min}^{7/6}}{\sqrt{\tilde{\lambda}_{\max}}} s_{\min}^2(A_1) \frac{1}{k^{1/6}} \right),$$
$$\|E\| \leq \tilde{O}\left(\lambda_{\min} \left(\frac{\tilde{\lambda}_{\min}}{\tilde{\lambda}_{\max}} \right)^{1.5} s_{\min}^3(A_1) \frac{1}{\sqrt{k}} \right).$$

Then, the proposed tensor decomposition algorithm recovers estimations of rank-1 components $(A_1)_j$'s satisfying error

$$\|(A_1)_j - (\hat{A}_1)_j\| \leq \tilde{O}\left(\frac{s_{\max}(A_1)}{\lambda_{\min}} \cdot \frac{\tilde{\lambda}_{\max}^2}{\sqrt{\tilde{\lambda}_{\min}}} \cdot \left[\frac{\|E_2\|^3}{\tilde{\lambda}_{\min}^{3.5} \cdot s_{\min}^6(A_1)} + \frac{\|E\|}{\tilde{\lambda}_{\min}^{1.5} \cdot s_{\min}^3(A_1)} \right] \right), \quad j \in [k]. \tag{46}$$

Proof: We do not have access to the true matrix $\tilde{M}_2$ and the true tensor $\tilde{T}$, and the perturbed versions $\widehat{M}_2 = \tilde{M}_2 - E_2$ and $\widehat{T} = \tilde{T} - E$ are used in the whitening procedure. Here, $E_2 \in \mathbb{R}^{d \times d}$ denotes the perturbation matrix, and $E \in \mathbb{R}^{d \times d \times d}$ denotes the perturbation tensor. Similar to the noiseless case, let $\widehat{W} \in \mathbb{R}^{d \times k}$ denotes the whitening matrix constructed by Procedure 5 such that $\widehat{W}^\top \widehat{M}_2 \widehat{W} = I_k$, and thus it orthogonalizes the noisy matrix $\widehat{M}_2$. Applying the whitening matrix $\widehat{W}$ to the tensor $\widehat{T}$, we have

$$\begin{aligned} \widehat{T}(\widehat{W}, \widehat{W}, \widehat{W}) &= \tilde{T}(W, W, W) - \tilde{T}(W - \widehat{W}, W - \widehat{W}, W - \widehat{W}) - E(\widehat{W}, \widehat{W}, \widehat{W}) \\ &= \sum_{j \in [k]} \mu_j v_j^{\otimes 3} - E_W, \end{aligned} \tag{47}$$

where we used Equation (41), and we defined

$$E_W := \tilde{T}(W - \widehat{W}, W - \widehat{W}, W - \widehat{W}) + E(\widehat{W}, \widehat{W}, \widehat{W}) \tag{48}$$

as the perturbation tensor after whitening. Note that the perturbation is from two sources; one is from the error in computing whitening matrix reflected in $W - \widehat{W}$, and the other is the error in tensor $\widehat{T}$ reflected in E.

We know that the rank-1 components v_j's form an orthonormal basis, and thus, we have a noisy orthogonal tensor decomposition problem in (47). We apply the result of Anandkumar et al. (2014c) where they show that if

$$\|E_W\| \leq \frac{\mu_{\min}\sqrt{\log k}}{\alpha_0\sqrt{k}},$$

for some constant $\alpha_0 > 1$, then the tensor power iteration (applied to the whitened tensor) recovers the tensor rank-1 components with bounded error (up to the permutation of columns)

$$\|v_j - \widehat{v}_j\| \leq \tilde{O}\left(\frac{\|E_W\|}{\mu_{\min}}\right). \tag{49}$$

We now relate the norm of E_W to the norm of original perturbations E and E_2. For the first term in (48), from Lemmata 4 and 5 of Song et al. (2013), we have

$$\|\tilde{T}(W - \widehat{W}, W - \widehat{W}, W - \widehat{W})\| \leq \frac{64\|E_2\|^3}{\tilde{\lambda}_{\min}^{3.5} \cdot s_{\min}^6(A_1)}.$$

For the second term, by the sub-multiplicative property we have

$$\|E(\widehat{W}, \widehat{W}, \widehat{W})\| \leq \|E\| \cdot \|\widehat{W}\|^3 \leq 8\|E\| \cdot \|W\|^3 \leq \frac{8\|E\|}{s_{\min}^3(A_1)\tilde{\lambda}_{\min}^{3/2}}.$$

Here in the last inequality, we used

$$\|W\| = \frac{1}{\sqrt{s_k(\tilde{M}_2)}} \leq \frac{1}{s_{\min}(A_1)\sqrt{\tilde{\lambda}_{\min}}},$$

where $s_k(\tilde{M}_2)$ denotes the k-th largest singular value of $\tilde{M}_2$. Here, the equality is from the definition of W based on rank-k SVD of $\tilde{M}_2$ in Procedure 5, and the inequality is from $\tilde{M}_2 = A_1 \operatorname{Diag}(\tilde{\lambda}) A_1^\top$.

Substituting these bounds, we finally need the condition

$$\frac{64\|E_2\|^3}{\tilde{\lambda}_{\min}^{3.5} \cdot s_{\min}^6(A_1)} + \frac{8\|E\|}{\tilde{\lambda}_{\min}^{1.5} \cdot s_{\min}^3(A_1)} \leq \frac{\lambda_{\min}\sqrt{\log k}}{\alpha_0 \tilde{\lambda}_{\max}^{1.5}\sqrt{k}},$$

where we also substituted bound $\mu_{\min} \geq \lambda_{\min}/\tilde{\lambda}_{\max}^{1.5}$, given Equation (42). The bounds stated in the lemma ensures that each of the terms on the left hand side of the inequality are bounded by the right hand side. Thus, by the result of Anandkumar et al. (2014c), we have $\|v_j - \widehat{v}_j\| \leq \tilde{O}(\|E_W\|/\mu_{\min})$. On the other hand, by the unwhitening relationship in (44), we have

$$\|(A_1)_j - (\hat{A}_1)_j\| = \frac{1}{\sqrt{\tilde{\lambda}_j}}\|\operatorname{Diag}(\gamma^{1/2}) \cdot [v_j - \widehat{v}_j]\| \leq \sqrt{\frac{\gamma_{\max}}{\tilde{\lambda}_{\min}}} \cdot \|v_j - \widehat{v}_j\| \leq s_{\max}(A_1) \cdot \sqrt{\frac{\tilde{\lambda}_{\max}}{\tilde{\lambda}_{\min}}} \cdot \|v_j - \widehat{v}_j\|. \tag{50}$$

where in the equality, we use the fact that orthonormal matrix U preserves the ℓ_2 norm, and the sub-multiplicative property is exploited in the first inequality. The last inequality is also from $\gamma_{\max} = s_{\max}(\tilde{M}_2) \leq s_{\max}^2(A_1) \cdot \tilde{\lambda}_{\max}$, which is from $\tilde{M}_2 = A_1 \operatorname{Diag}(\tilde{\lambda}) A_1^\top$. Incorporating the error bound on $\|v_j - \widehat{v}_j\|$ in (49), we have

$$\|(A_1)_j - (\hat{A}_1)_j\| \leq \tilde{O}\left(s_{\max}(A_1) \cdot \sqrt{\frac{\tilde{\lambda}_{\max}}{\tilde{\lambda}_{\min}}} \cdot \frac{\|E_W\|}{\mu_{\min}}\right) \leq \tilde{O}\left(\frac{s_{\max}(A_1)}{\lambda_{\min}} \cdot \frac{\tilde{\lambda}_{\max}^2}{\sqrt{\tilde{\lambda}_{\min}}} \cdot \|E_W\|\right),$$

where we used the bound $\mu_{\min} \geq \lambda_{\min}/\tilde{\lambda}_{\max}^{1.5}$ in the last step. $\square$

C.2 Fourier analysis guarantees

The analysis of Fourier method for estimating parameter b_1 includes the following two lemmas. In the first lemma, we argue the mean of random variable v introduced in Algorithm 1 in the realizable setting. This clarifies why the phase of v is related to unknown parameter b_1. In the second lemma, we argue the concentration of v around its mean leading to the sample complexity result. Note that v is denoted by $\tilde{v}$ in the realizable setting.

The Fourier method can be also used to estimate the weight vector a_2 since it appears in the magnitude of complex number v. In this section, we also provide the analysis of estimating a_2 with Fourier method which can be used as an alternative, while we primarily estimate a_2 by the ridge regression analyzed in Appendix C.3.

Lemma 7 (Restated). *Let*

$$\tilde{v} := \frac{1}{n} \sum_{i \in [n]} \frac{\tilde{y}_i}{p(x_i)} e^{-j\langle \omega_i, x_i \rangle}. \tag{51}$$

Notice this is a realizable of v in Procedure 2 where the output corresponds to a neural network $\tilde{y}$. If ω_i's are uniformly i.i.d. drawn from set Ω_l, then $\tilde{v}$ has mean (which is computed over x, $\tilde{y}$ and ω)

$$\mathbb{E}[\tilde{v}] = \frac{1}{|\Omega_l|} \Sigma\left(\frac{1}{2}\right) a_2(l) e^{j\pi b_1(l)}, \tag{52}$$

where $|\Omega_l|$ denotes the surface area of $d-1$ dimensional manifold Ω_l, and $\Sigma(\cdot)$ denotes the Fourier transform of $\sigma(\cdot)$.

Proof: Let $\tilde{F}(\omega)$ denote the Fourier transform of label function $\tilde{f}(x) := \mathbb{E}[\tilde{y}|x] = \langle a_2, \sigma(A_1^\top x + b_1)\rangle$ which is (Marks II and Arabshahi, 1994)

$$\tilde{F}(\omega) = \sum_{j \in [k]} \frac{a_2(j)}{|A_1(d,j)|} \Sigma\left(\frac{\omega_d}{A_1(d,j)}\right) e^{j2\pi b_1(j) \frac{\omega_d}{A_1(d,j)}} \delta\left(\omega_- - \frac{\omega_d}{A_1(d,j)} A_1(\backslash d, j)\right), \tag{53}$$

where $\Sigma(\cdot)$ is the Fourier transform of $\sigma(\cdot)$, $u_-^\top = [u_1, u_2, \ldots, u_{d-1}]$ is vector $u^\top$ with the last entry removed, $A_1(\backslash d, j) \in \mathbb{R}^{d-1}$ is the j-th column of matrix A_1 with the d-th (last) entry removed, and finally $\delta(u) = \delta(u_1)\delta(u_2)\cdots\delta(u_d)$.

Let $p(\omega)$ denote the probability density function of frequency ω. We have

$$\begin{aligned}
\mathbb{E}[\tilde{v}] &= \mathbb{E}_{x,\tilde{y},\omega}\left[\frac{\tilde{y}}{p(x)} e^{-j\langle \omega, x\rangle}\right] \\
&= \mathbb{E}_{x,\omega}\left[\mathbb{E}_{\tilde{y}|\{x,\omega\}}\left[\frac{\tilde{y}}{p(x)} e^{-j\langle \omega, x\rangle} \middle| x, \omega\right]\right] \\
&= \mathbb{E}_{x,\omega}\left[\frac{\tilde{f}(x)}{p(x)} e^{-j\langle \omega, x\rangle}\right] \\
&= \int_{\Omega_l} \int \tilde{f}(x) e^{-j\langle \omega, x\rangle} p(\omega) dx d\omega \\
&= \int_{\Omega_l} \tilde{F}(\omega) p(\omega) d\omega,
\end{aligned}$$

where the second equality uses the law of total expectation, the third equality exploits the label-generating function definition $\tilde{f}(x) := \mathbb{E}[\tilde{y}|x]$, and the final equality is from the definition of Fourier transform. The variable $\omega \in \mathbb{R}^d$ is drawn from a $d-1$ dimensional manifold $\Omega_l \subset \mathbb{R}^d$. In order to compute the above integral, we define d dimensional set

$$\Omega_{l;\nu} := \left\{ \omega \in \mathbb{R}^d : \frac{1}{2} - \frac{\nu}{2} \leq \|\omega\| \leq \frac{1}{2} + \frac{\nu}{2}, \left| \langle \omega, (\hat{A}_1)_l \rangle \right| \geq \frac{1 - \tilde{\epsilon}_1^2/2}{2} \right\},$$

for which $\Omega_l = \lim_{\nu \to 0^+} \Omega_{l;\nu}$. Assuming ω's are uniformly drawn from $\Omega_{l;\nu}$, we have

$$\begin{aligned}
\mathbb{E}[\tilde{v}] &= \lim_{\nu \to 0^+} \int_{\Omega_{l;\nu}} \tilde{F}(\omega) p(\omega) d\omega \\
&= \lim_{\nu \to 0^+} \frac{1}{|\Omega_{l;\nu}|} \int_{-\infty}^{+\infty} \tilde{F}(\omega) 1_{\Omega_{l;\nu}}(\omega) d\omega.
\end{aligned}$$

The second equality is from uniform draws of ω from set $\Omega_{l;\nu}$ such that $p(\omega) = \frac{1}{|\Omega_{l;\nu}|} 1_{\Omega_{l;\nu}}(\omega)$, where $1_S(\cdot)$ denotes the indicator function for set S. Here, $|\Omega_{l;\nu}|$ denotes the volume of d dimensional subspace $\Omega_{l;\nu}$, for which in the limit $\nu \to 0^+$, we have $|\Omega_{l;\nu}| = \nu \cdot |\Omega_l|$, where $|\Omega_l|$ denotes the surface area of $d-1$ dimensional manifold Ω_l.

For small enough $\tilde{\epsilon}_1$ in the definition of $\Omega_{l;\nu}$, only the delta function for $j = l$ in the expansion of $\tilde{F}(\omega)$ in (53) is survived from the above integral, and thus,

$$\mathbb{E}[\tilde{v}] = \lim_{\nu \to 0^+} \frac{1}{|\Omega_{l;\nu}|} \int_{-\infty}^{+\infty} \frac{a_2(l)}{|A_1(d,l)|} \Sigma \left(\frac{\omega_d}{A_1(d,l)} \right) e^{j2\pi b_1(l) \frac{\omega_d}{A_1(d,l)}} \delta \left(\omega_- - \frac{\omega_d}{A_1(d,l)} A_1(\backslash d, l) \right) 1_{\Omega_{l;\nu}}(\omega) d\omega.$$

In order to simplify the notations, in the rest of the proof we denote l-th column of matrix A_1 by vector α, i.e., $\alpha := (A_1)_l$. Thus, the goal is to compute the integral

$$I := \int_{-\infty}^{+\infty} \frac{1}{|\alpha_d|} \Sigma \left(\frac{\omega_d}{\alpha_d} \right) e^{j2\pi b_1(l) \frac{\omega_d}{\alpha_d}} \delta \left(\omega_- - \frac{\omega_d}{\alpha_d} \alpha_- \right) 1_{\Omega_{l;\nu}}(\omega) d\omega,$$

and note that $\mathbb{E}[\tilde{v}] = a_2(l) \cdot \lim_{\nu \to 0^+} \frac{I}{|\Omega_{l;\nu}|}$. The rest of the proof is about computing the above integral. The integral involves delta functions where the final value is expected to be computed at a single point specified by the intersection of line $\omega_- = \frac{\omega_d}{\alpha_d} \alpha_-$, and sphere $\|\omega\| = \frac{1}{2}$ (when we consider the limit $\nu \to 0^+$). This is based on the following integration property of delta functions such that for function $g(\cdot) : \mathbb{R} \to \mathbb{R}$,

$$\int_{-\infty}^{+\infty} g(t) \delta(t) dt = g(0). \tag{54}$$

We first expand the delta function as follows.

$$\begin{aligned}
I &= \int_{-\infty}^{+\infty} \frac{1}{|\alpha_d|} \Sigma \left(\frac{\omega_d}{\alpha_d} \right) e^{j2\pi b_1(l) \frac{\omega_d}{\alpha_d}} \delta \left(\omega_1 - \frac{\alpha_1}{\alpha_d} \omega_d \right) \cdots \delta \left(\omega_{d-1} - \frac{\alpha_{d-1}}{\alpha_d} \omega_d \right) 1_{\Omega_{l;\nu}}(\omega) d\omega, \\
&= \int \cdots \int_{-\infty}^{+\infty} \Sigma \left(\frac{\omega_d}{\alpha_d} \right) e^{j2\pi b_1(l) \frac{\omega_d}{\alpha_d}} \delta \left(\omega_1 - \frac{\alpha_1}{\alpha_d} \omega_d \right) \cdots \delta \left(\omega_{d-2} - \frac{\alpha_{d-2}}{\alpha_d} \omega_d \right) \\
&\qquad\qquad 1_{\Omega_{l;\nu}}(\omega) \cdot \delta \left(\alpha_d \omega_{d-1} - \alpha_{d-1} \omega_d \right) d\omega_1 \cdots \omega_d,
\end{aligned}$$

where we used the property $\frac{1}{|\beta|}\delta(t) = \delta(\beta t)$ in the second equality. Introducing new variable z, and applying the change of variable $\omega_d = \frac{1}{\alpha_{d-1}}(\alpha_d \omega_{d-1} - z)$, we have

$$\begin{aligned} I = \int \cdots \int_{-\infty}^{+\infty} &\Sigma\left(\frac{\omega_d}{\alpha_d}\right) e^{j2\pi b_1(l)\frac{\omega_d}{\alpha_d}} \delta\left(\omega_1 - \frac{\alpha_1}{\alpha_d}\omega_d\right) \cdots \delta\left(\omega_{d-2} - \frac{\alpha_{d-2}}{\alpha_d}\omega_d\right) \\ &\qquad 1_{\Omega_{l;\nu}}(\omega) \cdot \delta(z) d\omega_1 \cdots d\omega_{d-1} \frac{dz}{\alpha_{d-1}}, \\ = \int \cdots \int_{-\infty}^{+\infty} &\frac{1}{\alpha_{d-1}} \Sigma\left(\frac{\omega_{d-1}}{\alpha_{d-1}}\right) e^{j2\pi b_1(l)\frac{\omega_{d-1}}{\alpha_{d-1}}} \delta\left(\omega_1 - \frac{\alpha_1}{\alpha_{d-1}}\omega_{d-1}\right) \cdots \delta\left(\omega_{d-2} - \frac{\alpha_{d-2}}{\alpha_{d-1}}\omega_{d-1}\right) \\ &\qquad 1_{\Omega_{l;\nu}}\left(\left[\omega_1, \omega_2, \ldots, \omega_{d-1}, \frac{\alpha_d}{\alpha_{d-1}}\omega_{d-1}\right]\right) d\omega_1 \cdots d\omega_{d-1}. \end{aligned}$$

For the sake of simplifying the mathematical notations, we did not substitute all the ω_d's with z in the first equality, but note that all ω_d's are implicitly a function of z which is finally considered in the second equality where the delta integration property in (54) is applied to variable z (note that $z = 0$ is the same as $\frac{\omega_d}{\alpha_d} = \frac{\omega_{d-1}}{\alpha_{d-1}}$). Repeating the above process several times, we finally have

$$I = \int_{-\infty}^{+\infty} \frac{1}{\alpha_1} \Sigma\left(\frac{\omega_1}{\alpha_1}\right) e^{j2\pi b_1(l)\frac{\omega_1}{\alpha_1}} \cdot 1_{\Omega_{l;\nu}}\left(\left[\omega_1, \frac{\alpha_2}{\alpha_1}\omega_1, \ldots, \frac{\alpha_{d-1}}{\alpha_1}\omega_1, \frac{\alpha_d}{\alpha_1}\omega_1\right]\right) d\omega_1.$$

There is a line constraint as $\frac{\omega_1}{\alpha_1} = \frac{\omega_2}{\alpha_2} = \cdots = \frac{\omega_d}{\alpha_d}$ in the argument of indicator function. This implies that $\|\omega\| = \frac{\|\alpha\|}{\alpha_1}\omega_1 = \frac{\omega_1}{\alpha_1}$, where we used $\|\alpha\| = \|(A_1)_l\| = 1$. Incorporating this in the norm bound imposed by the definition of $\Omega_{l;\nu}$, we have $\frac{\alpha_1}{2}(1-\nu) \leq \omega_1 \leq \frac{\alpha_1}{2}(1+\nu)$, and hence,

$$I = \int_{\frac{\alpha_1}{2}(1-\nu)}^{\frac{\alpha_1}{2}(1+\nu)} \frac{1}{\alpha_1} \Sigma\left(\frac{\omega_1}{\alpha_1}\right) e^{j2\pi b_1(l)\frac{\omega_1}{\alpha_1}} d\omega_1.$$

We know $\mathbb{E}[\tilde{v}] = a_2(l) \cdot \lim_{\nu \to 0^+} \frac{I}{|\Omega_{l;\nu}|}$, and thus,

$$\mathbb{E}[\tilde{v}] = a_2(l) \cdot \frac{1}{\nu \cdot |\Omega_l|} \cdot \alpha_1 \nu \frac{1}{\alpha_1} \Sigma\left(\frac{1}{2}\right) e^{j2\pi b_1(l)\frac{1}{2}} = \frac{1}{|\Omega_l|} a_2(l) \Sigma\left(\frac{1}{2}\right) e^{j\pi b_1(l)},$$

where in the first step we use $|\Omega_{l;\nu}| = \nu \cdot |\Omega_l|$, and write the integral I in the limit $\nu \to 0^+$. This finishes the proof. $\square$

In the following lemma, we argue the concentration of v around its mean which leads to the sample complexity bound for estimating the parameter b_1 (and also a_2) within the desired error.

Lemma 11. *If the sample complexity*

$$n \geq O\left(\frac{\tilde{\zeta}_{\tilde{f}}}{\psi \tilde{\epsilon}_2^2} \log \frac{k}{\delta}\right) \tag{55}$$

holds for small enough $\tilde{\epsilon}_2 \leq \tilde{\zeta}_{\tilde{f}}$, then the estimates $\hat{a}_2(l) = \frac{|\Omega_l|}{|\Sigma(1/2)|}|\tilde{v}|$, and $\hat{b}_1(l) = \frac{1}{\pi}(\angle\tilde{v} - \angle\Sigma(1/2))$ for $l \in [k]$, in NN-LIFT Algorithm 1 (see the definition of $\tilde{v}$ in (51)*) satisfy with probability at least $1 - \delta$,*

$$|a_2(l) - \hat{a}_2(l)| \leq \frac{|\Omega_l|}{|\Sigma(1/2)|} O(\tilde{\epsilon}_2), \quad |b_1(l) - \hat{b}_1(l)| \leq \frac{|\Omega_l|}{\pi |\Sigma(1/2)| |a_2(l)|} O(\tilde{\epsilon}_2).$$

Proof: The result is proved by arguing the concentration of variable $\tilde{v}$ in (51) around its mean characterized in (52). We use the Bernstein's inequality to do this. Let $\tilde{v} := \sum_{i \in [n]} \tilde{v}_i$ where $\tilde{v}_i = \frac{1}{n} \frac{\tilde{y}_i}{p(x_i)} e^{-j\langle \omega_i, x_i \rangle}$. By the lower bound $p(x) \geq \psi$ assumed in Theorem 3 and labels $\tilde{y}_i$'s being bounded, the magnitude of centered $\tilde{v}_i$'s ($\tilde{v}_i - \mathbb{E}[\tilde{v}_i]$) are bounded by $O(\frac{1}{\psi n})$. The variance term is also bounded as

$$\sigma^2 = \left| \sum_{i \in [n]} \mathbb{E}\left[(\tilde{v}_i - \mathbb{E}[\tilde{v}_i]) (\overline{\tilde{v}_i - \mathbb{E}[\tilde{v}_i]}) \right] \right|,$$

where $\overline{u}$ denotes the complex conjugate of complex number u. This is bounded as

$$\sigma^2 \leq \sum_{i \in [n]} \mathbb{E}\left[\tilde{v}_i \overline{\tilde{v}_i} \right] = \frac{1}{n^2} \sum_{i \in [n]} \mathbb{E}\left[\frac{\tilde{y}_i^2}{p(x_i)^2} \right]$$

Since output $\tilde{y}$ is a binary label ($\tilde{y} \in \{0,1\}$), we have $\mathbb{E}[\tilde{y}^2 | x] = \mathbb{E}[\tilde{y}|x] = \tilde{f}(x)$, and thus,

$$\mathbb{E}\left[\frac{\tilde{y}^2}{p(x)^2} \right] = \mathbb{E}\left[\mathbb{E}\left[\frac{\tilde{y}^2}{p(x)^2} | x \right] \right] = \mathbb{E}\left[\frac{\tilde{f}(x)}{p(x)^2} \right] \leq \frac{1}{\psi} \int_{\mathbb{R}^d} \tilde{f}(x) dx = \frac{\tilde{\zeta}_{\tilde{f}}}{\psi},$$

where the inequality uses the bound $p(x) \geq \psi$ and the last equality is from definition of $\tilde{\zeta}_{\tilde{f}}$. This provides us the bound on variance as

$$\sigma^2 \leq \frac{\tilde{\zeta}_{\tilde{f}}}{\psi n}.$$

Applying Bernstein's inequality concludes the concentration bound such that with probability at least $1 - \delta$, we have

$$|\tilde{v} - \mathbb{E}[\tilde{v}]| \leq O\left(\frac{1}{\psi n} \log \frac{1}{\delta} + \sqrt{\frac{\tilde{\zeta}_{\tilde{f}}}{\psi n} \log \frac{1}{\delta}} \right) \leq O(\tilde{\epsilon}_2),$$

where the last inequality is from sample complexity bound. This implies that $||\tilde{v}| - |\mathbb{E}[\tilde{v}]|| \leq O(\tilde{\epsilon}_2)$. Substituting $|\mathbb{E}[\tilde{v}]|$ from (52) and considering estimate $\hat{a}_2(l) = \frac{|\Omega_l|}{|\Sigma(1/2)|} |\tilde{v}|$, we have

$$|\hat{a}_2(l) - a_2(l)| \leq \frac{|\Omega_l|}{|\Sigma(1/2)|} O(\tilde{\epsilon}_2),$$

which finishes the first part of the proof. For the phase, we have $\phi := \angle \tilde{v} - \angle \mathbb{E}[\tilde{v}] = \pi(\hat{b}_1(l) - b_1(l))$. On the other hand, for small enough error $\tilde{\epsilon}_2$ (and thus small ϕ), we have the approximation $\phi \sim \tan(\phi) \sim \frac{|\tilde{v} - \mathbb{E}[\tilde{v}]|}{|\mathbb{E}[\tilde{v}]|}$ (note that this is actually an upper bound such that $\phi \leq \tan(\phi)$). Thus,

$$|\hat{b}_1(l) - b_1(l)| \leq \frac{1}{\pi |\mathbb{E}[\tilde{v}]|} O(\tilde{\epsilon}_2) \leq \frac{|\Omega_l|}{\pi |\Sigma(1/2)| |a_2(l)|} O(\tilde{\epsilon}_2).$$

This finishes the proof of second bound. □

C.3 Ridge regression analysis and guarantees

Let $h := \sigma(A_1^\top x + b_1)$ denote the neuron or hidden layer variable. With slightly abuse of notation, in the rest of analysis in this section, we append variable h by the dummy variable 1 to represent the bias, and thus, $h \in \mathbb{R}^{k+1}$. We write the output as $\tilde{y} = h^\top \beta + \eta$, where

$$\beta := [a_2, b_2] \in \mathbb{R}^{k+1}.$$

Given the estimated parameters of first layer denoted by $\hat{A}_1$ and $\hat{b}_1$, the neurons are estimated as $\hat{h} := \sigma(\hat{A}_1^\top x + \hat{b}_1)$. In addition, the dummy variable 1 is also appended, and thus, $\hat{h} \in \mathbb{R}^{k+1}$. Because of this estimated encoding of neurons, we expand the output $\tilde{y}$ as

$$\tilde{y} = \hat{h}^\top \beta + \underbrace{(h^\top - \hat{h}^\top)\beta}_{\text{bias (approximation): } b(\hat{h})} + \underbrace{\eta}_{\text{noise}} = \tilde{f}(\hat{h}) + \eta, \tag{56}$$

where $\tilde{f}(\hat{h}) := \mathbb{E}[\tilde{y}|\hat{h}] = \hat{h}^\top \beta + b(\hat{h})$. Here, we have a noisy linear model with additional bias (approximation). Let $\hat{\beta}_\lambda$ denote the ridge regression estimator for some regularization parameter $\lambda \geq 0$, which is defined as the minimizer of the regularized empirical mean squared error, i.e.,

$$\hat{\beta}_\lambda := \arg\min_\beta \frac{1}{n} \sum_{i \in [n]} \left(\langle \beta, \hat{h}_i \rangle - \tilde{y}_i \right)^2 + \lambda \|\beta\|^2.$$

We know this estimator is given by (when $\hat{\Sigma}_{\hat{h}} + \lambda I \succ 0$)

$$\hat{\beta}_\lambda = \left(\hat{\Sigma}_{\hat{h}} + \lambda I \right)^{-1} \cdot \hat{\mathbb{E}}(\hat{h}\tilde{y}),$$

where $\hat{\Sigma}_{\hat{h}} := \frac{1}{n} \sum_{i \in [n]} \hat{h}_i \hat{h}_i^\top$ is the empirical covariance of $\hat{h}$, and $\hat{\mathbb{E}}$ denotes the empirical mean operator. The analysis of ridge regression leads to the following expected prediction error (risk) bound on the estimation of the output.

Lemma 12 (Expected prediction error of ridge regression)**.** *Suppose the parameter recovery results in Lemmata 9 and 11 on A_1 and b_1 hold. In addition, assume the nonlinear activating function $\sigma(\cdot)$ satisfies the Lipschitz property such that $|\sigma(u) - \sigma(u')| \leq L \cdot |u - u'|$, for $u, u' \in \mathbb{R}$. The following noise, approximation and statistical leverage conditions also hold. Then, by choosing the optimal $\lambda > 0$ in the λ-regularized ridge regression (which estimates the parameters $\hat{a}_2$ and $\hat{b}_2$), the estimated output as $\hat{f}(x) = \hat{a}_2^\top \sigma(\hat{A}_1^\top x + \hat{b}_1) + \hat{b}_2$ satisfies the risk bound*

$$\mathbb{E}[|\hat{f}(x) - \tilde{f}(x)|^2] \leq O\left(\frac{k\|\beta\|^2}{n} \right) + O\left(\sqrt{\frac{2k\|\beta\|^2}{n} \left(\mathbb{E}[b(\hat{h})^2] + \sigma_{noise}^2/2 \right)} \right) + \mathbb{E}[b(\hat{h})^2],$$

where

$$\mathbb{E}[b(\hat{h})^2] \leq \left[r + \frac{|\Omega_l|}{\pi |\Sigma(1/2)||a_2(l)|} \right]^2 \|\beta\|^2 L^2 k O(\tilde{\epsilon}^2).$$

Proof: Since $\hat{h} := \sigma(\hat{A}_1^\top x + \hat{b}_1)$, we equivalently argue the bound on $\mathbb{E}[(\hat{h}^\top \hat{\beta}_\lambda - \tilde{f}(\hat{h}))^2]$, where $\hat{f}(x) = \hat{f}(\hat{h}) = \hat{h}^\top \hat{\beta}_\lambda$. From standard results in the study of inverse problems, we know (see Proposition 5 in Hsu et al. (2014))

$$\mathbb{E}[(\hat{h}^\top \hat{\beta}_\lambda - \tilde{f}(\hat{h}))^2] = \mathbb{E}[(\hat{h}^\top \beta - \tilde{f}(\hat{h}))^2] + \|\hat{\beta}_\lambda - \beta\|_{\Sigma_{\hat{h}}}^2.$$

Here, for positive definite matrix $\Sigma \succ 0$, the vector norm $\|\cdot\|_\Sigma$ is defined as $\|v\|_\Sigma := \sqrt{v^\top \Sigma v}$. For the first term, by the definition of $\tilde{f}(\hat{h})$ as $\tilde{f}(\hat{h}) := \mathbb{E}[\tilde{y}|\hat{h}] = \hat{h}^\top \beta + b(\hat{h})$, we have

$$\mathbb{E}[(\hat{h}^\top \beta - \tilde{f}(\hat{h}))^2] = \mathbb{E}[b(\hat{h})^2].$$

Lemma 13 bounds $\mathbb{E}[b(\hat{h})^2]$ and bounding $\|\hat{\beta}_\lambda - \beta\|^2_{\Sigma_{\hat{h}}}$ is argued in Lemma 14 and Remark 7. Combining these bounds finishes the proof. $\square$

In order to have final risk bounded as $\mathbb{E}[|\hat{f}(x) - \tilde{f}(x)|^2] \le \tilde{O}(\epsilon^2)$, for some $\epsilon > 0$, the above lemma imposes sample complexity as (some of other parameters considered in (32), (55) are not repeated here)

$$n \ge \tilde{O}\left(L \frac{k\|\beta\|^2}{\epsilon^2}(1 + \sigma^2_{\text{noise}}) \right). \tag{57}$$

Lemma 13 (Bounded approximation)**.** *Suppose the parameter recovery results in Lemmata 9 and 11 on A_1 and b_1 hold. In addition, assume the nonlinear activating function $\sigma(\cdot)$ satisfies the Lipschitz property such that $|\sigma(u) - \sigma(u')| \le L \cdot |u - u'|$, for $u, u' \in \mathbb{R}$. Then, the approximation term is bounded as*

$$\mathbb{E}[b(\hat{h})^2] \le \left[r + \frac{|\Omega_l|}{\pi|\Sigma(1/2)||a_2(l)|} \right]^2 \|\beta\|^2 L^2 k O(\tilde{\epsilon}^2).$$

Proof: We have

$$\mathbb{E}[b(\hat{h})^2] = \mathbb{E}[\langle h - \hat{h}, \beta \rangle^2] \le \|\beta\|^2 \cdot \mathbb{E}[\|h - \hat{h}\|^2]. \tag{58}$$

Define $\tilde{\epsilon} := \max\{\tilde{\epsilon}_1, \tilde{\epsilon}_2\}$, where $\tilde{\epsilon}_1$ and $\tilde{\epsilon}_2$ are the corresponding bounds in Lemmata 9 and 11, respectively. Using the Lipschitz property of nonlinear function $\sigma(\cdot)$, we have

$$\begin{aligned} |h_l - \hat{h}_l| &= |\sigma(\langle (A_1)_l, x \rangle + b_1(l)) - \sigma(\langle (\hat{A}_1)_l, x \rangle + \hat{b}_1(l))| \\ &\le L \cdot \left[|\langle (A_1)_l - (\hat{A}_1)_l, x \rangle| + |b_1(l) - \hat{b}_1(l)| \right] \\ &\le L \cdot \left[r O(\tilde{\epsilon}) + \frac{|\Omega_l|}{\pi|\Sigma(1/2)||a_2(l)|} O(\tilde{\epsilon}) \right], \end{aligned}$$

where in the second inequality, we use the bounds in Lemmata 9 and 11, and bounded x such that $\|x\| \le r$. Applying this to (58) concludes the proof. $\square$

We now assume the following additional conditions to bound $\|\hat{\beta}_\lambda - \beta\|^2_{\Sigma_{\hat{h}}}$. The following discussions are along the results of Hsu et al. (2014).

We define the effective dimensions of the covariate $\hat{h}$ as

$$k_{p,\lambda} := \sum_{j \in [k]} \left(\frac{\lambda_j}{\lambda_j + \lambda} \right)^p, \quad p \in \{1, 2\},$$

where λ_j's denote the (positive) eigenvalues of $\Sigma_{\hat{h}}$, and λ is the regularization parameter of ridge regression.

- Subgaussian noise: there exists a finite $\sigma_{\text{noise}} \ge 0$ such that, almost surely,

  $$\mathbb{E}_\eta[\exp(\alpha\eta)|\hat{h}] \le \exp(\alpha^2 \sigma^2_{\text{noise}}/2), \quad \forall \alpha \in \mathbb{R},$$

  where η denotes the noise in the output $\tilde{y}$.

- Bounded statistical leverage: there exists a finite $\rho_\lambda \geq 1$ such that, almost surely,

$$\frac{\sqrt{k}}{\sqrt{(\inf\{\lambda_j\} + \lambda) k_{1,\lambda}}} \leq \rho_\lambda.$$

- Bounded approximation error at λ: there exists a finite $B_{\text{bias},\lambda} \geq 0$ such that, almost surely,

$$\rho_\lambda \left(B_{\max} + \sqrt{k} \|\beta\| \right) \leq B_{\text{bias},\lambda},$$

  where $|b(\hat{h})| \leq B_{\max}$. Note that the approximation term $b(\hat{h})$ is bounded in Lemma 13. The parameter $B_{\text{bias},\lambda}$ only contributes to the lower order terms in the analysis of ridge regression.

Lemma 14 (Bounding excess mean squared error: Theorem 2 of Hsu et al. (2014))**.** *Fix some $\lambda \geq 0$, and suppose the above noise, approximation and statistical leverage conditions hold, and in addition,*

$$n \geq \tilde{O}(\rho_\lambda^2 k_{1,\lambda}). \tag{59}$$

Then, we have

$$\|\hat{\beta}_\lambda - \beta\|^2_{\Sigma_{\hat{h}}} \leq \tilde{O} \left(\frac{k}{\lambda n} \left(\mathbb{E}[b(\hat{h})^2] + \sigma^2_{noise}/2 \right) + \frac{\lambda \|\beta\|^2}{2} \left(1 + \frac{k/\lambda + 1}{n} \right) \right) + o(1/n),$$

where $\mathbb{E}[b(\hat{h})^2]$ is bounded in Lemma 13.

In the above lemma, we also used the discussions in Remarks 12 and 15 of Hsu et al. (2014) which include comments on the simplification of the general result.

Remark 7 (Optimal λ)**.** *In addition, along the discussion in Remark 15 of Hsu et al. (2014), by choosing the optimal $\lambda > 0$ that minimizes the bound in the above lemma, we have*

$$\|\hat{\beta}_\lambda - \beta\|^2_{\Sigma_{\hat{h}}} \leq O\left(\frac{k\|\beta\|^2}{n} \right) + O\left(\sqrt{\frac{2k\|\beta\|^2}{n} \left(\mathbb{E}[b(\hat{h})^2] + \sigma^2_{noise}/2 \right)} \right).$$

D Proof of Theorem 5

Before we provide the proof, we first state the details of bound on C_f. We require

$$\begin{aligned} C_f \leq \min \Bigg\{ & \tilde{O}\left(\frac{1}{r} \left(\frac{1}{\sqrt{k}} + \delta_1 \right)^{-1} \frac{1}{\sqrt{\mathbb{E}[\|\mathcal{S}_3(x)\|^2]}} \cdot \frac{\tilde{\lambda}^2_{\min}}{\tilde{\lambda}^2_{\max}} \cdot \lambda_{\min} \cdot \frac{s^3_{\min}(A_1)}{s_{\max}(A_1)} \cdot \tilde{\epsilon}_1 \right), \\ & \tilde{O}\left(\frac{1}{r} \left(\frac{1}{\sqrt{k}} + \delta_1 \right)^{-1} \frac{1}{\sqrt{\mathbb{E}[\|\mathcal{S}_3(x)\|^2]}} \cdot \lambda_{\min} \left(\frac{\tilde{\lambda}_{\min}}{\tilde{\lambda}_{\max}} \right)^{1.5} s^3_{\min}(A_1) \cdot \frac{1}{\sqrt{k}} \right), \\ & O\left(\frac{1}{r} \left(\frac{1}{\sqrt{k}} + \delta_1 \right)^{-1} \frac{\mathbb{E}[\|\mathcal{S}_3(x)\|^2]^{1/4}}{\mathbb{E}[\|\mathcal{S}_2(x)\|^2]^{3/4}} \cdot \tilde{\lambda}_{\min} \cdot s^{1.5}_{\min}(A_1) \right) \Bigg\}. \end{aligned} \tag{60}$$

Proof of Theorem 5: We first argue that the perturbation involves both estimation and approximation parts.

Perturbation decomposition into approximation and estimation parts: Similar to the estimation part analysis, we need to ensure the perturbation from exact means is small enough to apply the analysis of Lemmas 9 and 11. Here, in addition to the empirical estimation of quantities (estimation error), the approximation error also contributes to the perturbation. This is because there is no realizable setting here, and the observations are from an arbitrary function $f(x)$. We address this for both the tensor decomposition and the Fourier parts as follows.

Recall that we use notation $\tilde{f}(x)$ (and $\tilde{y}$) to denote the output of a neural network. For arbitrary function $f(x)$, we refer to the neural network satisfying the approximation error provided in Theorem 4 by $\tilde{y}_f$. The ultimate goal of our analysis is to show that NN-LIFT recovers the parameters of this specific neural network with small error. More precisely, note that these are a class of neural networks satisfying the approximation bound in Theorem 5, and it suffices to say that the output of the algorithm is close enough to one of them.

Tensor decomposition: There are two perturbation sources in the tensor analysis. One is from the approximation part and the other is from the estimation part. By Lemma 6, the CP decomposition form is given by $\tilde{T}_f = \mathbb{E}[\tilde{y}_f \cdot \mathcal{S}_3(x)]$, and thus, the perturbation tensor is written as

$$E := \tilde{T}_f - \widehat{T} = \mathbb{E}[\tilde{y}_f \cdot \mathcal{S}_3(x)] - \frac{1}{n} \sum_{i \in [n]} y_i \cdot \mathcal{S}_3(x_i),$$

where $\widehat{T} = \frac{1}{n} \sum_{i \in [n]} y_i \cdot \mathcal{S}_3(x_i)$ is the empirical form used in NN-LIFT Algorithm 1. Note that the observations are from the arbitrary function $y = f(x)$. The perturbation tensor can be expanded as

$$E = \underbrace{\mathbb{E}[\tilde{y}_f \cdot \mathcal{S}_3(x)] - \mathbb{E}[y \cdot \mathcal{S}_3(x)]}_{:=E_{\text{apx.}}} + \underbrace{\mathbb{E}[y \cdot \mathcal{S}_3(x)] - \frac{1}{n} \sum_{i \in [n]} y_i \cdot \mathcal{S}_3(x_i)}_{:=E_{\text{est.}}},$$

where $E_{\text{apx.}}$ and $E_{\text{est.}}$ respectively denote the perturbations from approximation and estimation parts.

We also desire to use the exact second order moment $\tilde{M}_{2,f} = \mathbb{E}[\tilde{y}_f \cdot \mathcal{S}_2(x)]$ for the whitening Procedure 5 in the tensor decomposition method. But, we have an empirical version for which the perturbation matrix $E_2 := \tilde{M}_{2,f} - \widehat{M}_2$ is expanded as

$$E_2 = \underbrace{\mathbb{E}[\tilde{y}_f \cdot \mathcal{S}_2(x)] - \mathbb{E}[y \cdot \mathcal{S}_2(x)]}_{:=E_{2,\text{apx.}}} + \underbrace{\mathbb{E}[y \cdot \mathcal{S}_2(x)] - \frac{1}{n} \sum_{i \in [n]} y_i \cdot \mathcal{S}_2(x_i)}_{:=E_{2,\text{est.}}},$$

where $E_{2,\text{apx.}}$ and $E_{2,\text{est.}}$ respectively denote the perturbations from approximation and estimation parts.

In Theorem 3 where there is no approximation error, we only need to analyze the estimation perturbations characterized in (33) and (34) since the neural network output is directly observed (and thus, we use $\tilde{y}$ to denote the output). Now, the goal is to argue that the norm of perturbations E and E_2 are small enough (see Lemma 10), ensuring the tensor power iteration recovers the rank-1 components of $\tilde{T}_f = \mathbb{E}[\tilde{y}_f \cdot \mathcal{S}_3(x)]$ with bounded error. Again recall from Lemma 6 that the rank-1 components of tensor $\tilde{T}_f = \mathbb{E}[\tilde{y}_f \cdot \mathcal{S}_3(x)]$ are the desired components to recover.

The estimation perturbations $E_{\text{est.}}$ and $E_{2,\text{est.}}$ are similarly bounded as in Lemma 9 (see (35) and (36)), and thus, we have w.h.p.

$$\|E_{\text{est.}}\| \leq \tilde{O}\left(\frac{y_{\max}}{\sqrt{n}}\sqrt{\mathbb{E}\left[\left\|M_3(x)M_3^\top(x)\right\|\right]}\right),$$
$$\|E_{2,\text{est.}}\| \leq \tilde{O}\left(\frac{y_{\max}}{\sqrt{n}}\sqrt{\mathbb{E}\left[\left\|\mathcal{S}_2(x)\mathcal{S}_2^\top(x)\right\|\right]}\right),$$

where $M_3(x) \in \mathbb{R}^{d\times d^2}$ denotes the matricization of score function tensor $\mathcal{S}_3(x) \in \mathbb{R}^{d\times d\times d}$, and $y_{\max}$ is the bound on $|f(x)| = |y|$.

The norm of approximation perturbation $E_{\text{apx.}} := \mathbb{E}[(\tilde{y}_f - y)\cdot \mathcal{S}_3(x)]$ is bounded as

$$\begin{aligned}\|E_{\text{apx.}}\| &= \|\mathbb{E}[(\tilde{y}_f - y)\cdot \mathcal{S}_3(x)]\| \\ &\leq \mathbb{E}[\|(\tilde{y}_f - y)\cdot \mathcal{S}_3(x)\|] \\ &= \mathbb{E}[|\tilde{y}_f - y|\cdot \|\mathcal{S}_3(x)\|] \\ &\leq \left(\mathbb{E}[|\tilde{y}_f - y|^2]\cdot \mathbb{E}[\|\mathcal{S}_3(x)\|^2]\right)^{1/2},\end{aligned}$$

where the first inequality is from the Jensen's inequality applied to convex norm function, and we used Cauchy-Schwartz in the last inequality. Applying the approximation bound in Theorem 4, we have

$$\|E_{\text{apx.}}\| \leq O(rC_f)\cdot\left(\frac{1}{\sqrt{k}} + \delta_1\right)\cdot\sqrt{\mathbb{E}[\|\mathcal{S}_3(x)\|^2]}, \tag{61}$$

and similarly,

$$\|E_{2,\text{apx.}}\| \leq O(rC_f)\cdot\left(\frac{1}{\sqrt{k}} + \delta_1\right)\cdot\sqrt{\mathbb{E}[\|\mathcal{S}_2(x)\|^2]},$$

We now need to ensure the overall perturbations $E = E_{\text{est.}} + E_{\text{apx.}}$ and $E_2 = E_{2,\text{est.}} + E_{2,\text{apx.}}$ satisfies the required bounds in Lemma 10. Note that similar to what we do in Lemma 9, we first impose a bound such that the term involving $\|E\|$ is dominant in (46). Bounding the estimation part $\|E_{\text{est.}}\|$ provides similar sample complexity as in estimation Lemma 9 with $\tilde{y}_{\max}$ substituted by $y_{\max}$.

For the approximation error, by imposing (third bound stated in the theorem)

$$C_f \leq O\left(\frac{1}{r}\left(\frac{1}{\sqrt{k}} + \delta_1\right)^{-1}\frac{\mathbb{E}[\|\mathcal{S}_3(x)\|^2]^{1/4}}{\mathbb{E}[\|\mathcal{S}_2(x)\|^2]^{3/4}}\cdot\tilde{\lambda}_{\min}\cdot s_{\min}^{1.5}(A_1)\right),$$

we ensure that the term involving $\|E\|$ is dominant in the final recovery error in (46). By doing this, we do not need to impose the bound on $\|E_{2,\text{apx.}}\|$ anymore, and applying the bound in (61) to the required bound on $\|E\|$ in Lemma 10 leads to bound (second bound stated in the theorem)

$$C_f \leq \tilde{O}\left(\frac{1}{r}\left(\frac{1}{\sqrt{k}} + \delta_1\right)^{-1}\frac{1}{\sqrt{\mathbb{E}[\|\mathcal{S}_3(x)\|^2]}}\cdot\lambda_{\min}\left(\frac{\tilde{\lambda}_{\min}}{\tilde{\lambda}_{\max}}\right)^{1.5}s_{\min}^3(A_1)\cdot\frac{1}{\sqrt{k}}\right).$$

Finally, applying the result of Lemma 10, we have the column-wise error guarantees (up to permutation)

$$\begin{aligned}\|(A_1)_j - (\hat{A}_1)_j\| &\le \tilde{O}\left(\frac{s_{\max}(A_1)}{\lambda_{\min}} \frac{\tilde{\lambda}_{\max}^2}{\sqrt{\tilde{\lambda}_{\min}}} \frac{\|E_{\text{est.}}\| + \|E_{\text{apx.}}\|}{\tilde{\lambda}_{\min}^{1.5} \cdot s_{\min}^3(A_1)}\right), \\ &\le \tilde{O}\left(\frac{\tilde{\lambda}_{\max}^2}{\tilde{\lambda}_{\min}^2} \frac{s_{\max}(A_1)}{\lambda_{\min} \cdot s_{\min}^3(A_1)} \left[\frac{y_{\max}}{\sqrt{n}} \sqrt{\mathbb{E}\left[\left\|M_3(x)M_3^\top(x)\right\|\right]}\right.\right. \\ &\qquad\qquad \left.\left. + \; rC_f \cdot \left(\frac{1}{\sqrt{k}} + \delta_1\right) \cdot \sqrt{\mathbb{E}[\|\mathcal{S}_3(x)\|^2]}\right]\right) \\ &\le \tilde{O}\left(\tilde{\epsilon}_1\right),\end{aligned}$$

where in the second inequality we substituted the earlier bounds on $\|E_{\text{est.}}\|$ and $\|E_{\text{apx.}}\|$, and the first bounds on n and C_f stated in the theorem are used in the last inequality.

Fourier part: Let

$$\tilde{v}_f := \frac{1}{n} \sum_{i \in [n]} \frac{(\tilde{y}_f)_i}{p(x_i)} e^{-j\langle \omega_i, x_i \rangle}.$$

Note that this a realization of $\tilde{v}$ defined in (51) when the output is generated by a neural network satisfying approximation error provided in Theorem 4 denoted by $\tilde{y}_f$; see the discussion in the beginning of the proof.

The perturbation is now

$$e := \mathbb{E}[\tilde{v}_f] - \underbrace{\frac{1}{n} \sum_{i \in [n]} \frac{y_i}{p(x_i)} e^{-j\langle \omega_i, x_i \rangle}}_{=:v}.$$

Similar to the tensor decomposition part, it can be expanded to estimation and approximation parts as

$$e := \underbrace{\mathbb{E}[\tilde{v}_f] - \mathbb{E}[v]}_{e_{\text{apx.}}} + \underbrace{\mathbb{E}[v] - v}_{e_{\text{est.}}}.$$

Similar to Lemma 11, the estimation error is w.h.p. bounded as

$$|e_{\text{est.}}| \le O(\tilde{\epsilon}_2),$$

if the sample complexity satisfies $n \ge \tilde{O}\left(\frac{\zeta_f}{\psi \tilde{\epsilon}_2^2}\right)$, where $\zeta_f := \int_{\mathbb{R}^d} f(x)^2 dx$. Notice the difference between ζ_f and $\tilde{\zeta}_{\tilde{f}}$. The approximation part is also bounded as

$$|e_{\text{apx.}}| \le \frac{1}{\psi} \mathbb{E}[|\tilde{y}_f - y|] \le \frac{1}{\psi} \sqrt{\mathbb{E}[|\tilde{y}_f - y|^2]} \le \frac{1}{\psi} O(rC_f) \cdot \left(\frac{1}{\sqrt{k}} + \delta_1\right),$$

where the last inequality is from the approximation bound in Theorem 4. Imposing the condition

$$C_f \le \frac{1}{r} \left(\frac{1}{\sqrt{k}} + \delta_1\right)^{-1} \cdot O(\psi \tilde{\epsilon}_2) \tag{62}$$

satisfies the desired bound $|e_{\text{apx.}}| \le O(\tilde{\epsilon}_2)$. The rest of the analysis is the same as Lemma 11.

Ridge regression: It introduces an additional approximation term in the linear regression formulated in (56). Given the above bounds on C_f, the new approximation term only contributes to lower order terms.

Combining the analyzes for tensor decomposition, Fourier and ridge regression parts finishes the proof. $\square$

D.1 Discussion on Corollary 1

Similar to the specific Gaussian kernel function, we can also provide the results for other kernel functions and in general for positive definite functions as follows. $f(x)$ is said to be positive definite if $\sum_{j,l} x_i x_j f(x_j - x_l) \geq 0$, for all $x_j, x_l \in \mathbb{R}^d$. Barron (1993) shows that positive definite functions have C_f bounded as

$$C_f \leq \sqrt{-f(0) \cdot \nabla^2 f(0)},$$

where $\nabla^2 f(x) := \sum_{i\in[d]} \partial^2 f(x)/\partial x_i^2$. Note that the operator ∇^2 is different from the derivative operator $\nabla^{(2)}$ that we defined in (4). Applying this to the proposed bound in (18), we conclude that our algorithm can train a neural network which approximates a class of positive definite and kernel functions with similar bounds as in Theorem 5. Corollary 1 is for the special case of Gaussian kernel function.

Proof of Corollary 1: For the location and scale mixture $f(x) := \int K(\alpha(x+\beta))G(d\alpha, d\beta)$, we have (Barron, 1993) $C_f \leq C_K \cdot \int |\alpha| \cdot |G|(d\alpha, d\beta)$, where C_K denotes the corresponding parameter for $K(x)$. For the standard Gaussian kernel function $K(x)$ considered here, we have (Barron, 1993) $C_K \leq \sqrt{d}$, which concludes

$$C_f \leq \sqrt{d} \cdot \int |\alpha| \cdot |G|(d\alpha, d\beta).$$

We now apply the required bound in (18) to finish the proof. But, in this specific setting, we also have the following simplifications for the bound in (18).

For the Gaussian input $x \sim \mathcal{N}(0, \sigma_x^2 I_d)$, the score function is

$$\mathcal{S}_3(x) = \frac{1}{\sigma_x^6} x^{\otimes 3} - \frac{1}{\sigma_x^4} \sum_{j\in[d]} \left(x \otimes e_j \otimes e_j + e_j \otimes x \otimes e_j + e_j \otimes e_j \otimes x \right),$$

which has expected square norm as $\mathbb{E}[\|\mathcal{S}_3(x)\|^2] = \tilde{O}(d^3/\sigma_x^6)$.

Given the input is Gaussian and the activating function is the step function, we can also write the coefficients λ_j and $\tilde{\lambda}_j$ as

$$\lambda_j = a_2(j) \cdot \frac{1}{\sqrt{2\pi}\sigma_x^3} \cdot \exp\left(-\frac{b_1(j)^2}{2\sigma_x^2}\right) \cdot \left(\frac{b_1(j)^2}{\sigma_x^2} - 1\right),$$
$$\tilde{\lambda}_j = a_2(j) \cdot \frac{b_1(j)}{\sqrt{2\pi}\sigma_x^3} \cdot \exp\left(-\frac{b_1(j)^2}{2\sigma_x^2}\right).$$

Given the bounds on coefficients as $|b_1(j)| \leq 1$, $|a_2(j)| \leq 2C_f$, $j \in [k]$, we have

$$\frac{\tilde{\lambda}_{\min}}{\tilde{\lambda}_{\max}} \geq \frac{(a_2)_{\min} \cdot (b_1)_{\min}}{2C_f} \exp(-1/(2\sigma_x^2)),$$
$$\lambda_{\min} \geq \frac{(a_2)_{\min}}{\sqrt{2\pi}\sigma_x^3} \exp(-1/(2\sigma_x^2)) \cdot \min_{j\in[k]} |b_1(j)^2/\sigma_x^2 - 1|.$$

Recall that the columns of A_1 are randomly drawn from the Fourier spectrum of $f(x)$ as described in (15). Given $f(x)$ is the Gaussian kernel, the Fourier spectrum $\|\omega\|\cdot|F(\omega)|$ corresponds to a sub-gaussain distribution. Thus, the singular values of A_1 are bounded as (Rudelson and Vershynin, 2009)

$$\frac{s_{\min}(A_1)}{s_{\max}(A_1)} \geq \frac{1-\sqrt{k/d}}{1+\sqrt{k/d}} \geq O(1),$$

where the last inequality is from $k = Cd$ for some small enough $C < 1$.

Substituting these bounds in the required bound in (18) finishes the proof. □

References

A. Agarwal, A. Anandkumar, P. Jain, P. Netrapalli, and R. Tandon. Learning Sparsely Used Overcomplete Dictionaries. In *Conference on Learning Theory (COLT)*, June 2014.

Guillaume Alain and Yoshua Bengio. What regularized auto-encoders learn from the data generating distribution. *arXiv preprint arXiv:1211.4246*, 2012.

A. Anandkumar, R. Ge, D. Hsu, and S. M. Kakade. A Tensor Spectral Approach to Learning Mixed Membership Community Models. In *Conference on Learning Theory (COLT)*, June 2013.

Anima Anandkumar, Rong Ge, and Majid Janzamin. Sample Complexity Analysis for Learning Overcomplete Latent Variable Models through Tensor Methods. *arXiv preprint arXiv:1408.0553*, Aug. 2014a.

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

Animashree Anandkumar, Rong Ge, and Majid Janzamin. Guaranteed Non-Orthogonal Tensor Decomposition via Alternating Rank-1 Updates. *arXiv preprint arXiv:1402.5180*, Feb. 2014c.

Animashree Anandkumar, Rong Ge, and Majid Janzamin. Learning Overcomplete Latent Variable Models through Tensor Methods. In *Proceedings of the Conference on Learning Theory (COLT)*, Paris, France, July 2015.

Alexandr Andoni, Rina Panigrahy, Gregory Valiant, and Li Zhang. Learning polynomials with neural networks. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 1908–1916, 2014.

Martin Anthony and Peter L Bartlett. *Neural network learning: Theoretical foundations*. cambridge university press, 2009.

Sanjeev Arora, Aditya Bhaskara, Rong Ge, and Tengyu Ma. Provable bounds for learning some deep representations. *arXiv preprint arXiv:1310.6343*, 2013.

Antonio Auffinger, Gerard Ben Arous, et al. Complexity of random smooth functions on the high-dimensional sphere. *The Annals of Probability*, 41(6):4214–4247, 2013.

Pierre Baldi and Kurt Hornik. Neural networks and principal component analysis: Learning from examples without local minima. *Neural networks*, 2(1):53–58, 1989.

Andrew R. Barron. Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Transactions on Information Theory*, 39(3):930–945, May 1993.

Andrew R Barron. Approximation and estimation bounds for artificial neural networks. *Machine Learning*, 14:115–133, 1994.

Peter Bartlett and Shai Ben-David. Hardness results for neural network approximation problems. In *Computational Learning Theory*, pages 50–62. Springer, 1999.

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

A. Bhaskara, M. Charikar, A. Moitra, and A. Vijayaraghavan. Smoothed analysis of tensor decompositions. *arXiv preprint arXiv:1311.3651*, 2013.

Avrim L Blum and Ronald L Rivest. Training a 3-node neural network is np-complete. In *Machine learning: From theory to applications*, pages 9–28. Springer, 1993.

Martin L Brady, Raghu Raghavan, and Joseph Slawny. Back propagation fails to separate where perceptrons succeed. *Circuits and Systems, IEEE Transactions on*, 36(5):665–674, 1989.

Venkat Chandrasekaran, Pablo Parrilo, Alan S Willsky, et al. Latent variable graphical model selection via convex optimization. In *Communication, Control, and Computing (Allerton), 2010 48th Annual Allerton Conference on*, pages 1610–1613. IEEE, 2010.

Anna Choromanska, Mikael Henaff, Michaël Mathieu, Gérard Ben Arous, and Yann LeCun. The loss surface of multilayer networks. In *Proc. of 18th Intl. Conf. on Artificial Intelligence and Statistics (AISTATS)*, 2015.

G. Cybenko. approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, 2:303–314, 1989a.

George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989b.

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.

P Frasconi, M Gori, and A Tesi. Successes and failures of backpropagation: A theoretical investigation. *Progress in Neural Networks: Architecture*, 5:205, 1997.

Marco Gori and Alberto Tesi. On the problem of local minima in backpropagation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(1):76–86, 1992.

Benjamin D. Haeffele and René Vidal. Global optimality in tensor factorization, deep learning, and beyond. *CoRR*, abs/1506.07540, 2015.

Moritz Hardt, Benjamin Recht, and Yoram Singer. Train faster, generalize better: Stability of stochastic gradient descent. *arXiv preprint arXiv:1509.01240*, 2015.

Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

K. Hornik, M. Stinchcombe, and H. White. multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366, 1989.

Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4 (2):251–257, 1991.

Daniel Hsu, Sham M Kakade, and Tong Zhang. Random design analysis of ridge regression. *Foundations of Computational Mathematics*, 14(3):569–600, 2014.

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

Majid Janzamin, Hanie Sedghi, and Anima Anandkumar. Score Function Features for Discriminative Learning: Matrix and Tensor Frameworks. *arXiv preprint arXiv:1412.2863*, Dec. 2014.

Michael J Kearns and Umesh Virkumar Vazirani. *An introduction to computational learning theory.* MIT press, 1994.

Pravesh Kothari and Raghu Meka. Almost optimal pseudorandom generators for spherical caps. *arXiv preprint arXiv:1411.6299*, 2014.

Christian Kuhlmann. Hardness results for general two-layer neural networks. In *Proc. of COLT*, pages 275–285, 2000.

Roi Livni, Shai Shalev-Shwartz, and Ohad Shamir. On the computational efficiency of training neural networks. In *Advances in Neural Information Processing Systems*, pages 855–863, 2014.

Robert J Marks II and Payman Arabshahi. Fourier analysis and filtering of a single hidden layer perceptron. In *International Conference on Artificial Neural Networks (IEEE/ENNS), Sorrento, Italy*, 1994.

Nelson Morgan and Hervé Bourlard. Generalization and parameter estimation in feedforward nets: Some experiments. In *NIPS*, pages 630–637, 1989.

Raúl Rojas. *Neural networks: a systematic introduction.* Springer Science & Business Media, 1996.

Mark Rudelson and Roman Vershynin. Smallest singular value of a random rectangular matrix. *Communications on Pure and Applied Mathematics*, 62(12):1707–1739, 2009.

Hanie Sedghi and Anima Anandkumar. Provable methods for training neural networks with sparse connectivity. *NIPS workshop on Deep Learning and Representation Learning*, Dec. 2014.

Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning: From Theory to Algorithms.* Cambridge University Press, 2014.

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

L. Song, A. Anandkumar, B. Dai, and B. Xie. Nonparametric estimation of multi-view latent variable models. *arXiv preprint arXiv:1311.3287*, Nov. 2013.

Bharath Sriperumbudur, Kenji Fukumizu, Revant Kumar, Arthur Gretton, and Aapo Hyvärinen. Density estimation in infinite dimensional exponential families. *arXiv preprint arXiv:1312.3516*, 2013.

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.

Martin J Wainwright and Michael I Jordan. Graphical models, exponential families, and variational inference. *Foundations and Trends® in Machine Learning*, 1(1-2):1–305, 2008.

Yining Wang, Hsiao-Yu Tung, Alexander Smola, and Animashree Anandkumar. Fast and guaranteed tensor decomposition via sketching. In *Proc. of NIPS*, 2015.

T. Zhang and G. Golub. Rank-one approximation to high order tensors. *SIAM Journal on Matrix Analysis and Applications*, 23:534–550, 2001.

Yuchen Zhang, Jason D. Lee, and Michael I. Jordan. ℓ_1-regularized neural networks are improperly learnable in polynomial time. *CoRR*, abs/1510.03528, 2015.