Mathematical Programming manuscript No.
(will be inserted by the editor)

Neural Spectrahedra and Semidefinite Lifts: Global Convex Optimization of Degree-Two Polynomial Activation Neural Networks in Polynomial-Time

Burak Bartan · Mert Pilanci

Received: date / Accepted: date

Abstract The training of two-layer neural networks with nonlinear activation functions is an important non-convex optimization problem with numerous applications and promising performance in layerwise deep learning. In this paper, we develop exact convex optimization formulations for two-layer neural networks with second degree polynomial activations based on dual relaxations and semidefinite programming. Remarkably, we show that our semidefinite relaxations are always tight. Therefore, the computational complexity for global optimization is polynomial in the input dimension and sample size for all input data. The developed convex formulations are proven to achieve the same globally optimal solution set as their non-convex counterparts. Specifically, globally optimal two-layer neural networks with degree-two polynomial activations can be found by solving a semidefinite program (SDP) and decomposing the solution using a procedure we call Neural Decomposition. Moreover, the choice of regularizers plays a crucial role in the computational tractability of neural network training. We show that the standard weight decay regularization formulation is NP-hard, whereas other simple convex penalties render the problem tractable in polynomial time via convex programming. The techniques go beyond the fully connected architecture to encompass various neural network structures including those with vector outputs and convolutional architectures. We provide extensive numerical simulations showing that the standard backpropagation approach often fails to achieve the global optimum of the training loss. The proposed approach is significantly faster to obtain better test accuracy compared to the standard backpropagation procedure. [1]

B. Bartan
Department of Electrical Engineering, Stanford University, CA, USA
E-mail: bbartan@stanford.edu

M. Pilanci
Department of Electrical Engineering, Stanford University, CA, USA
E-mail: pilanci@stanford.edu

[1] The supplementary material for this work can be accessed by following the link in [4].

Keywords Neural networks Convex optimization Semidefinite programming Polynomial activation

Mathematics Subject Classification (2020) 90C22 90C25 90C26 68T07

1 Introduction

We study neural networks from the optimization perspective by deriving equivalent convex optimization formulations with identical global optimal solution sets. The derived convex problems have important theoretical and practical implications for the computational complexity of optimal neural network training. Moreover, the convex optimization perspective provides a more concise parameterization of neural networks and reveal novel insights.

In non-convex optimization, the choice of optimization method and its internal hyperparameters, such as initialization, mini-batching and step sizes, have a considerable effect on the quality of the learned model. This is in sharp contrast to convex optimization problems, where locally optimal solutions are globally optimal and optimizer parameters have no influence on the solution and therefore the model. Moreover, the solutions of convex optimization problems can be obtained in a very robust, efficient and reproducible manner thanks to the elegant and extensively studied structure of convex programs. Therefore, our convex optimization based globally optimal training procedure enables the study of the neural network model and the optimization procedure in a *decoupled* way. For instance, step sizes employed in the optimization can be considered hyperparameters of non-convex models, which affect the model quality and may require extensive tuning. For a classification task, in our convex optimization formulation, step sizes as well as the choice of the optimizers are no longer hyperparameters to obtain better classification accuracy. Any convex optimization solver can be leveraged to solve the convex problem to obtain a globally optimal model.

Various nonlinearities have been proposed in the literature as activation functions. Among the most widely adopted ones is the ReLU (rectified linear unit) activation given by $(u) = \max(0; u)$. A recently proposed alternative is the swish activation $(u) = u(1 + e^{-u})^{-1}$, which performs comparably well [40]. Another important class is the polynomial activation where the activation function is a scalar polynomial of a fixed degree. We focus on second degree polynomial activation functions, i.e., $(u) = au^2 + bu + c$. Although polynomial coefficients $a; b; c$ can be regarded as hyperparameters, it is often sufficient to select them to approximate a target nonlinear activation function such as the ReLU or swish activation. ReLU and swish activations are plotted in Figure 1 along with their second degree polynomial approximations.

Our derivation of the convex program for degree-two polynomial activations leverages convex duality and the S-procedure, and results in a simple semidefinite program (SDP). We refer the reader to [39] for a survey of the S-procedure and applications in SDPs.

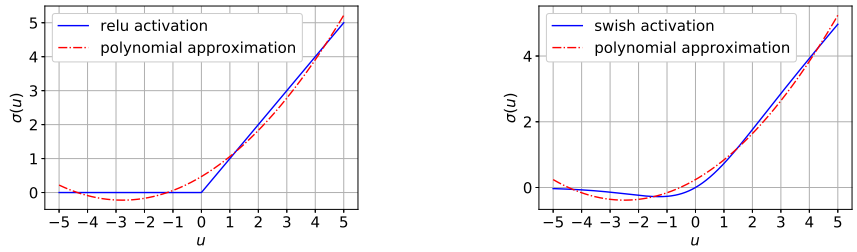


Fig. 1 ReLU (left) and swish (right) activation functions and their second degree polynomial approximations. ReLU activation: $(u) = \max(0; u)$ and its polynomial approximation: $(u) = 0{:}09u^2 + 0{:}5u + 0{:}47$. Swish activation: $(u) = u(1 + e^{\ u})^{\ 1}$ and its polynomial approximation: $(u) = 0{:}1u^2 + 0{:}5u + 0{:}24$.

Main aspects of our work that differ from others in the literature that study the *optimization landscape* of two-layer neural networks (e.g. see prior work section) are the following: Our results (1) provide global optimal solutions in polynomial time, (2) uncover an important role of the regularizer in computational tractability, (3) hold for arbitrary convex loss functions and other network architectures such as vector output, convolutional and pooling, (4) are independent of the choice of the numerical optimizer and its parameters.

1.1 Overview of Our Contributions

– We show that the standard optimization formulation for training neural networks $f\ (x) = \sum_{j=1}^{m} (x^T u_j)\ _j$ with trainable parameters $= (u_1; \dots; u_m;\ _1; \dots;\ _m)$, degree-two polynomial activations $(u) = au^2 + bu + c$, training data $X = [x_1; \dots; x_n]^T\ 2\ \mathrm{R}^{n\ \ d}$, $y\ 2\ \mathrm{R}^n$ and weight decay (WD) regularization given by

$$(P_{WD}) \quad \min_{\substack{u_j 2\mathrm{R}^d; \\ {}_j 2\mathrm{R}; 8j\,2[m]}} \ \sum_{j=1} (Xu_j)\ _j; y\ + \ \sum_{j=1} (k u_j k_2^2 + \ _j^2) \qquad (1)$$

is computationally intractable via reduction from the NP-hard subset sum problem for all values of m. The loss function $\hat{y}; y)$ is a function of the network output $\hat{y}\ 2\ \mathrm{R}^n$. The hardness result is valid for any loss function that satisfies $\arg\min_{\hat{y} 2\mathrm{R}^n} \hat{y}; y) = y$.

– Surprisingly, for square activation, i.e., $(u) = u^2$, we show that modifying the quadratic weight decay regularization to *cubic regularization*

$$(P_{SQ}) \quad \min_{\substack{u_j 2\mathrm{R}^d; \\ {}_j 2\mathrm{R}; 8j\,2[m]}} \ \sum_{j=1} (Xu_j)\ _j; y\ + \ \sum_{j=1} (k u_j k_2^3 + j\ _j j^3) \qquad (2)$$

enables global optimization in polynomial time via convex semidefinite programming when the number of neurons m is above a threshold m .

The threshold m is proportional to the rank of the solution of the convex semidefinite program. An exact expression is provided in the sequel. The computational complexity is polynomial in all problem parameters $(n; d; m)$. The result holds for any convex and closed function $\grave{}(\hat{y}; y)$; a standard example is the squared loss $\grave{}(\hat{y}; y) = k\hat{y} \quad yk_2^2$.

– Furthermore, for any degree-two polynomial activation , the non-convex neural network training problem

$$(P) \qquad \min_{u_j 2 R^d;\ _j 2 R;\, 8j\, 2 [m]} \quad \grave{} \Big(\sum_{j=1} (X u_j)\ _j ; y \Big) + \sum_{j=1} j\ _j j \qquad \text{s.t.} \quad k u_j k_2 = 1 ; 8j\ 2\ [m] \tag{3}$$

can be equivalently stated as a convex semidefinite problem and solved in polynomial time. In fact, the cubic regularization strategy in (2) is a special case of this convex program. The result holds universally for all input data without any conditions and also holds when $\quad ! \ 0$.

– In deriving the convex formulations, we identify a concise re-parameterization of the neural network parameters that enables exact convexification by removing the redundancy in their classical formulation. This is similar in spirit to the semidefinite lifting procedure in relaxations of combinatorial optimization problems. In contrast to these relaxations, we show that our lifting is always exact as soon as the network width exceeds a critical threshold which can be efficiently determined.
– We develop a matrix decomposition procedure called Neural Decomposition to extract the optimal network parameters from the solution of our convex program, which is guaranteed to produce an optimal neural network. Neural Decomposition transforms the convex re-parameterization to the overparameterized, i.e., redundant, formulation in a similar spirit to (a non-orthogonal version of) Eigenvalue Decomposition.
– In addition to the fully connected architecture, we derive equivalent convex programs for other architectures such as convolutional, pooling and vector output architectures (see the supplementary material [4] for details).
– We provide extensive numerical simulations showing that the standard backpropagation approach fails to achieve the global optimum of the training loss. Moreover, the test accuracy of the proposed convex optimization is considerably higher in standard datasets as well as random planted models. Our convex optimization solver is significantly faster in total computation time to achieve similar or better test accuracy.

1.2 Prior Work

A considerable fraction of recent works on the analysis of optimization landscape of neural networks focuses on explaining why gradient descent performs well. The works [12,44] consider the optimization landscape of a restricted class of neural networks with square activation and quadratic regularization

where the second layer weights are *xed*. They show that when the neural network is overparameterized, i.e., $m \quad d$, the non-convex loss function has benign properties: all local minima are global and all saddle points have a direction of negative curvature. However, in this paper we show that training both the first and second layer weights with quadratic regularization in fact makes global optimization NP-hard for any value of m. In contrast, we provide a different formulation to obtain the global optimal solution via convex optimization in the more general case when the second layer weights are also optimized, the activation function is any arbitrary degree-two polynomial, and global optimum is achieved for all values of m. The work in [30] similarly studies two-layer neural networks with square activation function and squared loss and states results on both optimization and generalization properties. The authors in [17] focus on square activation networks from the perspectives of optimization landscape and generalization performance, where the setting is based on a planted model with a full rank weight matrix. In [25,28] it was shown that sufficiently wide ReLU networks have a benign landscape when each layer is sufficiently wide, satisfying $m \quad n + 1$.

Another recent work analyzing the training of neural networks with quadratic-like activations for deeper architectures is [2]. Authors in [2] consider degree-two polynomial activation functions and investigate layerwise training and compare with end-to-end training of layers. It is demonstrated in [2] that the degree-two polynomial activation function performs comparably to ReLU activation in deep networks. More specifically, it is reported in [2] that for deep neural networks, ReLU activation achieves a classification accuracy of 0:96 and a degree-two polynomial activation yields an accuracy of 0:95 on the Cifar-10 dataset. Similarly for the Cifar-100 dataset, they obtain an accuracy of 0:81 for ReLU activation and 0:76 for the degree-two polynomial activation. These numerical results are obtained for the activation $(u) = u + 0{:}1u^2$, which the authors prefer over the square activation $(u) = u^2$ to make the neural network training stable. Moreover, the performance of layerwise learning with such activation functions is considerably high, although there is a gap between end-to-end trained models. In addition, neural networks with polynomial activations have immediate applications in encrypted computing [22,18,31,34]. In encrypted computing, it is desirable to have a low degree polynomial as the activation function. For instance, homomorphic encryption can only support additions and multiplications in a straightforward way, which necessitates low degree polynomials as activations. In [18], degree-two polynomial approximations were shown to be effective for accurate neural network predictions with encryption.

In a recent series of papers, the authors derived convex formulations for training ReLU neural networks to global optimality [38,14,15,13,41,42]. Our work takes a similar convex duality approach in deriving the convex equivalents of non-convex neural network training problems. In particular, the previous work in this area deals with ReLU activations while in this work we focus on degree-two polynomial activations. Hence, the techniques involved in deriving the convex programs and the resulting convex programs are substan-

tially different. The convex program derived for ReLU activation in [38] has polynomial time trainability for fixed rank data matrices, whereas the convex programs developed in this work are polynomial time trainable with respect to all problem dimensions. More specifically, the convex program in [38] is

$$\begin{aligned} \min_{v_i, w_i \in \mathbb{R}^d, \forall i \in [H]} \quad & \ell\Big(\sum_{i=1}^{H} D_i X (v_i - w_i), y\Big) + \beta \sum_{i=1}^{H} (\|v_i\|_2 + \|w_i\|_2) \\ \text{s.t.} \quad & (2D_i - I_n) X v_i \geq 0, \ (2D_i - I_n) X w_i \geq 0, \forall i \in [H], \end{aligned} \tag{4}$$

where the neural network weights are constructed from $v_i \in \mathbb{R}^d$ and $w_i \in \mathbb{R}^d$, $i = 1, \ldots, H$. The matrices D_i are diagonal matrices whose diagonal entries consist of $\mathbb{1}_{(Xu \geq 0)}, \mathbb{1}_{(Xu \geq 0)}, \ldots, \mathbb{1}_{(Xu \geq 0)}$ for all possible $u \in \mathbb{R}^d$. The number of distinct D_i matrices, denoted by H is the number of hyperplane arrangements corresponding to the data matrix X. It is known that H is bounded by $2r\left(\frac{e(n-1)}{r}\right)^r$ where $r = \operatorname{rank}(X)$ (see [38] for the details). In particular, convolutional neural networks have a fixed value of r, for instance m filters of size 3×3 yield $r = 9$. This is an exponential improvement over previously known methods that train optimal ReLU networks which are exponential in the number of neurons m and/or the number of samples n [3,19,6].

The work in [7] presents formulations for convex factorization machines with nuclear norm regularization, which is known to obtain low rank solutions. Vector output extension for factorization machines and *polynomial networks*, which are different from *polynomial activation networks*, is developed in [8]. Polynomial networks are equivalent to square activation networks with an addition of a linear neuron. In [8], the authors consider learning an infinitely wide square activation layer by a greedy algorithm. The proposed method in [8] does not provide optimal finite width networks. Furthermore, [29] presents a greedy algorithm for training polynomial networks. The algorithm provided in [29] is based on gradually adding neurons to the neural network to reduce the loss. More recently, [43] considers applying lifting for square activation neural networks and presents non-convex algorithms for low rank matrix estimation for two-layer neural network training.

1.3 Notation

Throughout the text, $\sigma : \mathbb{R} \to \mathbb{R}$ denotes the activation function of the hidden layer. We refer to the function $\sigma(u) = u^2$ as square activation and $\sigma(u) = au^2 + bu + c$ where $a, b, c \in \mathbb{R}$ as degree-two polynomial activation. We use $X \in \mathbb{R}^{n \times d}$ to denote the data matrix, where its rows $x_i \in \mathbb{R}^d$ correspond to data samples and columns are the features. In the text, whenever we have a function mapping from $\mathbb{R}$ to $\mathbb{R}$ with a vector argument (e.g., $\sigma(v)$ or v^2 where v is a vector), this means the elementwise application of that function to all the components of the vector v. We denote a column vector of ones by $\bar{1}$ and its dimension can be understood from the context. vec( ) denotes the vectorized

version of its argument. In writing optimization problems, we use min and max to refer to "minimize" and "maximize". We use the notations $[m]$ and $1;:::;m$ interchangeably.

We use $`(\hat{y};y)$ for convex loss functions throughout the text. The notation $`\ (v) = \sup_z(v^T z \quad `(z;y))$ denotes the Fenchel conjugate of the function $`(\ ;y)$. Furthermore, we assume $`\ \ = `$ which holds when $`$ is a convex and closed function [9]. We use $Z \quad 0$ for positive semidefinite matrices (PSD). $S^{d\ \ d}$ refers to the set of $d \quad d$ dimensional symmetric matrices. tr refers to matrix trace. is used for outer product. The operator **conv** stands for the convex hull of a set.

1.4 Preliminaries on Semidefinite Lifting

We defer the discussion of semidefinite lifting for two-layer neural networks to next section. We now briefly discuss a class of problems where SDP relaxations lead to exact convex optimization solutions of the original non-convex problem and also instances where they fail to be exact. Let us consider a quadratic objective problem with a single quadratic constraint:

$$\begin{aligned} \min_{u2R^d} \quad & u^T Q_1 u + b_1^T u + c_1 \\ \text{s.t.} \quad & u^T Q_2 u + b_2^T u + c_2 \quad 0 \end{aligned} \tag{5}$$

where $Q_1;Q_2\ 2\ R^{d\ \ d}$ are indefinite, i.e., not assumed to be positive semidefinite and $b_1;b_2\ 2\ R^d$, $c_1;c_2\ 2\ R$. Due to the indefinite quadratics, this is a non-convex optimization problem. By introducing a matrix variable $U = uu^T$, one can equivalently state this problem as

$$\begin{aligned} \min_{\substack{U2R^{d\ \ d} \\ u2R^d}} \quad & \mathrm{tr}(Q_1 U) + b_1^T u + c_1 \\ \text{s.t.} \quad & \mathrm{tr}(Q_2 U) + b_2^T u + c_2 \quad 0 \\ & U = uu^T : \end{aligned} \tag{6}$$

This problem can be relaxed by replacing the equality by the matrix inequality $U \quad uu^T$. Re-writing the expression $U \quad uu^T$ as a linear matrix inequality via the Schur complement formula yields the following SDP

$$\begin{aligned} \min_{\substack{U2R^{d\ \ d} \\ u2R^d}} \quad & \mathrm{tr}(Q_1 U) + b_1^T u + c_1 \\ \text{s.t.} \quad & \mathrm{tr}(Q_2 U) + b_2^T u + c_2 \quad 0 \\ & \begin{matrix} 1 & u^T \\ u & U \end{matrix} \quad 0 : \end{aligned} \tag{7}$$

Remarkably, it can be shown that the original non-convex problem in (5) can be solved exactly by solving the convex SDP in (7) via duality, under the

mild assumption that the original problem is strictly feasible (see [9]). This shows that the SDP relaxation is exact in this problem, returning a globally optimal solution when one exists. We note that there are alternative numerical procedures to compute the global optimum of quadratic programs with one quadratic constraint [9].

As an alternative to the lifting based primal approach, it is possible to obtain the same convex SDP formulation via Lagrangian duality. First, we note that the dual of the original non-convex problem in (5) is the SDP

$$\begin{aligned} \max_{;\ 2\mathsf{R}} \quad & \frac{1}{4} \quad + c_2 \quad + c_1 \\ \text{s.t.} \quad & \begin{matrix} Q_1 + \ Q_2 & b_1 + \ b_2 \\ (b_1 + \ b_2)^T & \end{matrix} \qquad 0 \\ & 0 : \end{aligned} \tag{8}$$

Taking the dual of this SDP, we obtain the bidual of (5), which gives us the same SDP relaxation in (7).

We also note that the lifting approach $U = uu^T$ followed by the relaxation $U \quad uu^T$ is not tight for quadratic programs with more than two quadratic constraints [33,10]. A notable case with multiple constraints is the NP-hard Max-Cut problem and its SDP relaxation [20]

$$\max_{\substack{u 2 \mathsf{R}^d \\ u_i^2 = 1; 8 i 2 [d]}} u^T Q u = \max_{\substack{u 2 \mathsf{R}^d \\ u_i^2 = 1; 8 i 2 [d]}} \operatorname{tr}(Q u u^T) \qquad \max_{\substack{U 2 \mathsf{R}^{d \quad d}; U \quad 0 \\ U_{i\,i} = 1; 8 i 2 [d]}} \operatorname{tr}(Q U) : \tag{9}$$

The SDP relaxation of Max-Cut is not tight in general since its feasible set contains the cut polytope

$$\mathbf{conv} \quad uu^T \ : u_i \ 2 \ f \quad 1; +1 g \, 8 i$$

and other non-integral extreme points [26]. Nevertheless, an approximation ratio of 0:878 can be obtained via the Goemans-Williamson randomized rounding procedure under certain restrictions on Q [20]. It is conjectured that this is the best approximation ratio for Max-Cut [23], whereas it can be formally proven to be NP-hard to approximate within a factor of $\frac{16}{17}$ [21,45]. Hence, in general we cannot expect to obtain exact solutions to problems of combinatorial nature, such as Max-Cut and variants using computationally efficient SDP relaxations.

It is instructive to note that a naive application of the SDP lifting strategy is not immediately tractable for two-layer neural networks. For simplicity, consider a scalar output degree-two polynomial activation network $f(x) = \sum_{j=1}^{m} (x^T u_j) \ _j$ where $(u) = u^2 + u$, and $f u_j ; \ _j g_{j=1}^m$ are trainable parameters. The corresponding training problem for a given loss function $`(; y)$ and

its SDP relaxation are as follows

$$\min_{\substack{u_j \in \mathbb{R}^d; \\ \alpha_j \in \mathbb{R}, \forall j \in [m]}} \left\| \sum_{j=1}^{m} ((Xu_j)_+ + Xu_j)\alpha_j - y \right\|_2^2 + \beta \sum_{j=1}^{m} \|u_j\|_2^2 + \gamma \sum_{j=1}^{m} |\alpha_j|$$

$$\min_{\substack{u_j \in \mathbb{R}^d, \alpha_j \in \mathbb{R}, U_j \in \mathbb{R}^{d \times d}, \\ U_j = u_j u_j^T, \forall j \in [m]}} \left\| \sum_{j=1}^{m} (D_j X U_j + D_j X u_j)\alpha_j - y \right\|_2^2 + \beta \sum_{j=1}^{m} \|u_j\|_2^2 + \gamma \sum_{j=1}^{m} |\alpha_j|, \quad (10)$$

The above problem is *non-convex* due to the bilinear terms $\{U_j, \alpha_j\}_{j=1}^m$. Moreover, a variable change $\hat{U}_j = U_j \alpha_j$ does not respect semidefinite constraints $U_j = u_j u_j^T$ when $\alpha_j \in \mathbb{R}$. Another limitation is the prohibitively high number of variables in the lifted space, which is $d^2m + dm + m$ as opposed to $dm + m$ in the original problem. Therefore, a different convex analytic formulation is needed to address all these problems.

Although SDP relaxations are extensively studied for various non-convex problems (see e.g. [46] for a survey of applications), instances with exact SDP relaxations are exceptionally rare. As will be discussed in the sequel, our main result for two-layer neural networks is another instance of an SDP relaxation leading to *exact* formulations where the semidefinite relaxation is tight.

In convex geometry, a *spectrahedron* is a convex body that can be represented as a linear matrix inequality which are the feasible sets of semidefinite programs. An example is the *elliptope* defined as the feasible set of the Max-Cut relaxation given by $U \succeq 0, U_{ii} = 1 \forall i$, which is a subset of $n \times n$ symmetric positive-definite matrices. Due to the existence of efficient projection operators and barrier functions of linear matrix inequalities, optimizing convex objectives over spectrahedra can be efficiently implemented, which renders SDPs tractable. We will show that degree-two polynomial activation networks can be represented via a class of linear matrix inequalities, dubbed *neural spectrahedra* (see Figure 2 for an example), which enables global optimization in polynomial time and elucidates their parameterization in convex analytic terms.

1.5 Paper Organization

The paper is structured as follows. Section 2 gives an overview of the theory developed in this work. Section 3 describes the convex optimization formulation via duality and S-procedure for degree-two polynomial activation neural networks. Section 4 establishes via the neural decomposition method that the convex problem developed in Section 3 can be used to train two-layer neural networks to global optimality. The hardness result for the weight decay regularization is studied in Section 5. We discuss the implementation details for solving the convex programs and give experimental results in Section 6. Omitted proofs can be found in the appendix. The supplementary material [4] has sections on the special case of square activation neural networks with cubic regularization, and on vector output and convolutional architectures, as well as additional numerical results.

Table 1 List of the neural network architectures considered in this work and the corresponding convex programs. Abbreviations are as follows. Poly (scalar): Degree-two polynomial activation scalar output, Poly (vector): Degree-two polynomial activation vector output, Convolutional: Convolutional neural network (CNN) with degree-two polynomial activation, Pooling: CNN with degree-two polynomial activation and average pooling, Quad (scalar, cubic reg): Square activation scalar output with cubic regularization, Quad (scalar, quad reg): Square activation scalar output with quadratic regularization. K is the number of patches and f is the filter size for the convolutional architecture. C is the output dimension for the vector output case. P is the pool size for average pooling. (u) is defined as u^2 for square activation, and au^2+bu+c for degree-two polynomial activation.

	Non-convex objective	Convex formulation	Upper bound on critical width m
Poly (scalar)	$\ell\left(\sum_{j=1}^{m}(Xu_j)_j, y\right) + \sum_{j=1}^{m}\lambda_j\lvert\alpha_j\rvert$ s.t. $\lVert u_j\rVert_2 = 1$	(21)	$2(d+1)$
Poly (vector)	$\ell\left(\sum_{j=1}^{m}(Xu_j)\alpha_j^T, Y\right) + \sum_{j=1}^{m}\lVert\alpha_j\rVert_1$ s.t. $\lVert u_j\rVert_2 = 1$	Supp. Mat. [4]	$2(d+1)C$
CNN	$\ell\left(\sum_{j=1}^{m}\sum_{k=1}^{K}(X_k u_j)_k, y\right) + \sum_{j=1}^{m}\lVert\alpha_j\rVert_1$ s.t. $\lVert u_j\rVert_2 = 1$	Supp. Mat. [4]	$2(f+1)K^2$
Pooling	$\ell\left(\sum_{j=1}^{m}\sum_{k=1}^{K=P}\frac{1}{P}\sum_{l=1}^{P}(X_{(k-1)P+l}u_j)_k, y\right) + \sum_{j=1}^{m}\lVert\alpha_j\rVert_1$ s.t. $\lVert u_j\rVert_2 = 1$	Supp. Mat. [4]	$2(f+1)\frac{K^2}{P^2}$
Quad (scalar, cubic reg)	$\ell\left(\sum_{j=1}^{m}(Xu_j)_j, y\right) + \sum_{j=1}^{m}\lvert\alpha_j\rvert$ s.t. $\lVert u_j\rVert_2 = 1$, or $\ell\left(\sum_{j=1}^{m}(Xu_j)_j, y\right) + \frac{1}{c}\sum_{j=1}^{m}(\lvert\alpha_j\rvert^3 + \lVert u_j\rVert_2^3)$	Supp. Mat. [4]	d
Quad (scalar, quad reg)	$\ell\left(\sum_{j=1}^{m}(Xu_j)_j, y\right) + \sum_{j=1}^{m}\lvert\alpha_j\rvert^{2=3}$ s.t. $\lVert u_j\rVert_2 = 1$, or $\ell\left(\sum_{j=1}^{m}(Xu_j)_j, y\right) + \frac{1}{c}\sum_{j=1}^{m}(\lvert\alpha_j\rvert^2 + \lVert u_j\rVert_2^2)$	NP-hard (intractable)	-

We summarize the types of neural network architectures considered in this work and the corresponding convex problems in Table 1. The fourth column of Table 1 shows the upper bounds for critical width m , i.e., the optimal number of neurons that one needs for global optimization of any problems with number of neurons $m \quad m$.

2 Lifted Representations of Networks with Degree-Two Polynomial Activations

Consider the network $f(x) = \sum_{j=1}^{m}(x^T u_j)_j$ where the activation function is the degree-two polynomial $(u) = au^2+bu+c$. First, we note that the neural network output can be written as

$$
\begin{aligned}
f(x) &= \sum_{j=1}^{m} a(x^T u_j)^2 + bx^T u_j + c \;{}_j = \sum_{j=1}^{m} \langle axx^T; u_j u_j^T\rangle + \langle bx; u_j\rangle + c \;{}_j \\
&= \begin{bmatrix} \mathrm{vec}(axx^T) \\ bx \\ c \end{bmatrix}^T \begin{bmatrix} \mathrm{vec}\left(\sum_{j=1}^{m} u_j u_j^T \;{}_j\right) \\ \sum_{j=1}^{m} u_j \;{}_j \\ \sum_{j=1}^{m} {}_j \end{bmatrix} \\
&= \langle (x);\; (fu_j;\; {}_j g_{j=1}^m)\rangle ; \qquad (11)
\end{aligned}
$$

where $: \mathbb{R}^d \;!\; \mathbb{R}^{d^2+d+1}$ and $: \mathbb{R}^{m(d+1)} \;!\; \mathbb{R}^{d^2+d+1}$ are formally defined in the sequel. The above identity shows that the nonlinear neural network output is linear over the *lifted features*

$$(x) := \; axx^T; bx; c \;\; 2 \mathbb{R}^{d^2+d+1}:$$

Fig. 2 (Left) The Neural Cone $\mathcal{C}_2^1$ described by $(u^2 \; ; u \; ; \;) \; 2 \; \mathbb{R}^3$ where $u; \quad 2 \; \mathbb{R}; juj$ $_1$ $_2$. (Right) Neural Spectrahedron $M(1)$ described by $(Z_{11}; Z_{12}; Z_{22}) \; 2 \; \mathbb{R}^3$ where $Z = \begin{bmatrix} Z_{11} & Z_{12} & Z_{13} \\ Z_{12} & Z_{22} & Z_{23} \\ Z_{13} & Z_{23} & Z_{33} \end{bmatrix} \; 0; \; Z_{11} + Z_{22} = Z_{33} \quad 1$ (constrained to the slice $Z_{22} = Z_{11}$ and $Z^0 = 0$ in (14)). Note that we display a slice of the spectrahedron, which is higher dimensional.

In turn, the nonlinear model $f(x)$ is completely characterized by the *lifted parameters* which we define as the following matrix-vector-scalar triplet

$$(fu_j; \; _j g_{j=1}^m) := \sum_{j=1} u_j u_j^T \; _j; \sum_{j=1} u_j \; _j; \sum_{j=1} _j \quad 2 \; \mathbb{R}^{d^2+d+1}:$$

Optimizing over the lifted parameter space initially appears as hard as the original non-convex neural network training problem. This is due to the cubic and quadratic terms involving the weights of the hidden and output layer in the lifted parameters. Furthermore, norms of the network weights are nonlinear in the lifted parameters, which complicates regularization terms, e.g., $\sum_{j=1}^m ku_j k_2^2$ typically included in training. Nevertheless, one of our main results shows that the lifted parameters can be exactly described using linear matrix inequalities.

We begin by characterizing the lifted parameter space as a non-convex cone.

Definition 1 (Neural Cone of degree two) We define the non-convex cone $\mathcal{C}_2^m \quad \mathbb{R}^{d^2+d+1}$ as

$$\mathcal{C}_2^m := \sum_{j=1} u_j u_j^T \; _j; \sum_{j=1} u_j \; _j; \sum_{j=1} _j \quad : u_j \; 2 \; \mathbb{R}^d; ku_j k_2 = 1; \; _j \; 2 \; \mathbb{R} \, 8j \; 2 \; [m] \quad : \tag{12}$$

See Figure 2 (left) for a depiction of $\mathcal{C}_2^1 \quad \mathbb{R}^3$ corresponding to the case $m = 1; d = 1$.

Surprisingly, we will show that the original non-convex neural network problem is solved exactly to *global optimality* when the optimization is performed over a convex set which we define as the *Neural Spectrahedron*, given by the convex hull of the cone $\mathcal{C}_2$. In other words, every element of the convex hull can be associated with a neural network of the form $f(x) = \sum_{j=1}^m (x^T u_j) \; _j$ through

a special matrix decomposition procedure which we introduce in the Neural Decomposition section. Moreover, a Neural Spectrahedron can be described by a simple linear matrix inequality. Consequently, these two results enable global optimization of neural networks with polynomial activations of degree two in polynomial time with respect to all problem parameters: dimension d, number of samples n and number of neurons m. To the best of our knowledge, this is the first instance of a method that globally optimizes a standard neural network architecture with computational complexity polynomial in all problem dimensions. We refer the reader to the recent work [38] for a convex optimization formulation of networks with ReLU activation, where the worst case computational complexity is $O((\frac{n}{r})^r)$ with $r = \text{rank}(X)$.

It is equally important that our results characterize neural networks as constrained linear learning methods $h\ (x);\ \ i$ in the lifted feature space $\ (x)$, where the constraints on the lifted parameters $\ $ are precisely described by a Neural Spectrahedron via linear matrix inequalities. These constraints can be easily tackled with convex semidefinite programming or closed-form projections onto these sets in iterative first-order algorithms.

Next, we describe a compact set that we call *neural spectrahedron* which describes the lifted parameter space of networks with a constraint on the $\grave{}_1$ norm of output layer weights.

Definition 2 A neural spectrahedron $S_2^m(t)\ \ \mathsf{R}^{d^2+d+1}$ is defined as the compact convex set

$$S_2^m(t) := \mathbf{conv}\left(\left\{ \sum_{j=1} u_j u_j^T \ _j;\ \sum_{j=1} u_j \ _j;\ \sum_{j=1} \ _j \ : ku_jk_2 = 1; \right. \right. \\ \left. \left. \ _j \ 2\ \mathsf{R}; 8j\ 2\ [m];\ \sum_{j=1} j \ _j j \quad t \ : \right. \right) \tag{13}$$

We will show that a neural spectrahedron can be equivalently described as a linear matrix inequality via defining $S_2^m(t) = \ M_1(t); M_2(t); M_4(t)$ with $M_1(t)\ 2\ \mathsf{R}^{d\ \ d}$, $M_2(t)\ 2\ \mathsf{R}^{d\ \ 1}$ and $M_4(t)\ 2\ \mathsf{R}$ for all $m \quad m$ where

$$M(t) = \left(Z \quad Z^0 : Z = \begin{bmatrix} Z_1 & Z_2 \\ Z_2^T & Z_4 \end{bmatrix} \ 0; Z^0 = \begin{bmatrix} Z_1^0 & Z_2^0 \\ Z_2^{0\,T} & Z_4^0 \end{bmatrix} \ 0; \right. \\ \left. \text{tr}(Z_1) = Z_4; \text{tr}(Z_1^0) = Z_4^0; Z_4 + Z_4^0 \quad t \ ; \right) \tag{14}$$

$Z; Z^0\ 2\ \mathsf{S}^{(d+1)\ \ (d+1)}$, $Z_1; Z_1^0\ 2\ \mathsf{S}^{d\ \ d}$, $Z_2; Z_2^0\ 2\ \mathsf{R}^{d\ \ 1}$ and $Z_4; Z_4^0\ 2\ \mathsf{R}_+$, and $m \ = m\ (t)$ is a critical number of neurons that satisfies $m\ (0) = 0$ and $m\ (t) \quad 2(d+1)\,8t$, which will be explicitly defined in the sequel. Therefore, an efficient description of the set $M(t)$ in terms of linear matrix inequalities enables efficient convex optimization methods in polynomial time.

2.1 A Direct Derivation of SDP Formulations of the Neural Spectrahedron

In this section, we show how to obtain SDP based formulations of the neural network problem using a direct approach. In Section 3, we will show that this direct formulation is identical to the one obtained through Lagrangian duality, mirroring the relaxations of non-convex quadratic programs shown in Section 1.4.

First, we separate positive and negative components of the parameters $f\ _j g_{j=1}^m$, and parameterize $S_2^m(t)$ as follows

$$S_2^m(t) = \ ^+S_2^m(t_+) \quad ^+S_2^m(t\ ) : t_+; t \quad 0;\ t_+ + t\ = t\ ; \tag{15}$$

where denotes the Minkowski sum and we defined the one-sided Neural Spectrahedron $^+S_2^m(t_+)$ as

$$^+S_2^m(t) := \mathbf{conv}\left(\sum_{j=1} u_j u_j^T\ _j; \sum_{j=1} u_j\ _j; \sum_{j=1} \ _j : ku_j k_2 = 1; \right.$$
$$\left. _j \ 2\ \mathsf{R}_+; 8j\ 2\ [m]; \sum_{j=1} \ _j \quad t\ : \right) \tag{16}$$

We observe that $^+S_2^m(t)$ is identical to the set $^+M_1; {}^+M_2; {}^+M_4 \quad \mathsf{R}^{d^2+d+1}$ where

$$^+M(t) := t\,\mathbf{conv}\left(\sum_{j=1} \begin{matrix} u_j \\ 1 \end{matrix} \begin{matrix} u_j \\ 1 \end{matrix}^T \ _j : u_j\ 2\ \mathsf{R}^d; ku_j k_2 = 1; \right.$$
$$\left. _j \ 2\ \mathsf{R}_+; 8j\ 2\ [m]; \sum_{j=1} \ _j \quad 1\ ; \right) \tag{17}$$

which is partitioned as $^+M(t) = \begin{matrix} {}^+M_1 & {}^+M_2 \\ {}^+M_2^T & {}^+M_4 \end{matrix}$ and $^+M_1 \quad \mathsf{S}^{d\ \ d}; {}^+M_2 \quad \mathsf{R}^{d\ \ 1}$ and $^+M_4 \quad \mathsf{R}_+$.

Next, we note that as soon as the network width[2] satisfies $m \quad d+1$, we have

$$^+M(t) : = t\,\mathbf{conv}\left(\left(\begin{matrix} u \\ 1 \end{matrix} \begin{matrix} u \\ 1 \end{matrix}^T : kuk_2 = 1 \right) \quad [\ \mathbf{0}\ ; \right. \tag{18}$$

where $\mathbf{0}$ is the zero matrix, since $\sum_{j=1}^m \begin{matrix} u_j \\ 1 \end{matrix} \begin{matrix} u_j \\ 1 \end{matrix}^T \ _j \ 2\ \mathsf{S}^{(d+1)\ \ (d+1)}$ is a positive semidefinite matrix, and hence can be factorized[3] as a convex combination of at most $d+1$ rank-one matrices of the form $\begin{matrix} u \\ 1 \end{matrix} \begin{matrix} u \\ 1 \end{matrix}^T$. Note that

[2] This assumption on the width is relaxed in Theorem 1 (see Section 3).

[3] We describe the details of this factorization in the Neural Decomposition section.

the zero matrix is included to account for the inequality $\sum_{j=1}^{m} {}_j \leq 1$ in (17). This important observation enables us to represent the convex hull of the non-convex Neural Cone (an example is shown in Figure 2), via the simple convex body ${}^{+}M(t)$ given in (18).

Most importantly, the positive Neural Spectrahedron set ${}^{+}M(t)$ provides a representation of the non-convex Neural Cone C_2^m via its extreme points. ${}^{+}M(t)$ has a simple description as a linear matrix inequality provided in the following lemma (the proof can be found in the appendix).

Lemma 1 *For $m \geq d+1$, it holds that*

$$ {}^{+}M(t) = \left\{ Z : Z = \begin{bmatrix} Z_1 & Z_2 \\ Z_2^T & Z_4 \end{bmatrix} \succeq 0;\ \mathrm{tr}(Z_1) = Z_4 \leq t \right\} \tag{19} $$

Therefore the positive Neural Spectrahedron can be represented as the intersection of the positive semide nite cone and linear inequalities. Moreover, every element of ${}^{+}M(t)$ can be factorized as $\sum_{j=1}^{m} \left[u_j u_j^T \ {}_j u_j \ {}_j \right]$ for some $\|u_j\|_2 = 1;\ {}_j \geq 0; \forall j \in [m];\ \sum_{j=1}^{m} {}_j \leq t$, which can be identi ed as an element of the non-convex Neural Cone C_2^m and a neural network in the lifted parameter space as shown in (11).

The assumption $m \geq d+1$ is only used here for the simplicity of the illustration. We improve this condition on m, the width of the network, for the same SDP formulation in Section 3. In the following sections, we only require $m \geq m^*$, where m^* can be determined via the convex SDP. Furthermore, the regularization parameter directly controls the number of neurons m^*. We illustrate the effect of the regularization parameter on m^* in the numerical experiments section, and show that m^* can be made arbitrarily small.

3 Convex Optimization and Duality for Degree-Two Polynomial Activation Networks

We consider the non-convex training of a two-layer fully connected neural network with degree-two polynomial activations $f(x) = \sum_{j=1}^{m} (x^T u_j)_j$ and derive a convex dual optimization problem. Here, is the degree-two polynomial $(u) = au^2 + bu + c$. This neural network has m neurons with the first layer weights $u_j \in \mathbb{R}^d$ and second layer weights ${}_j \in \mathbb{R}$. We refer to this case where $f : \mathbb{R}^d \to \mathbb{R}$ as the scalar output case. The results are extended to the vector output case in the supplementary material [4].

It is relatively easy to obtain a weak dual that provides a lower-bound via Lagrangian duality. However, in non-convex problems, a duality gap may exist since strong duality does not hold in general. Remarkably, we show that strong duality holds as soon as the network width exceeds a threshold which can be easily determined.

We will assume ℓ_1 norm regularization on the second layer weights as regularization and include constraints that the first layer weights are unit norm.

We note that ℓ_1 norm regularization on the second layer weights results in a special dual problem and hence is crucial in the derivations. We show in Square Activation Networks section that this formulation is equivalent to cubic regularization when the activation function is the square activation. For the squared ℓ_2 norm regularization, i.e., weight decay, we will in fact show that the problem is NP-hard.

The training of a network under this setting requires solving the non-convex optimization problem given by

$$(P) \quad \min_{u_j \in \mathbb{R}^d,\ \alpha_j \in \mathbb{R}, \forall j \in [m]} \ell\left(\sum_{j=1}^{m} (X u_j)_+ \alpha_j, y\right) + \beta \sum_{j=1}^{m} |\alpha_j|$$
$$\text{s.t.} \quad \|u_j\|_2 = 1, \forall j \in [m]. \tag{20}$$

Here, $\ell(\hat{y}, y)$ is a convex loss function, e.g., squared loss or logistic loss. We will use the notation $p^* := \min(P)$, i.e., the minimum value of the optimization problem (P). Theorem 1 states the main result for degree-two polynomial activation neural networks that the non-convex optimization problem in (20) can be solved globally optimally via a convex problem.

Theorem 1 (Globally optimal convex program for degree-2 polynomial activation networks) *The solution of the convex problem*

$$(SDRP) \quad \min_{Z, Z^0 \in \mathbb{S}^{(d+1)} \times (d+1)} \ell(\hat{y}, y) + \beta (Z_4 + Z_4^0)$$
$$\text{s.t.} \quad \hat{y}_i = a x_i^T (Z_1 - Z_1^0) x_i + b x_i^T (Z_2 - Z_2^0) + c(Z_4 - Z_4^0), \quad i = 1, \ldots, n$$
$$\operatorname{tr}(Z_1) = Z_4, \ \operatorname{tr}(Z_1^0) = Z_4^0$$
$$Z = \begin{bmatrix} Z_1 & Z_2 \\ Z_2^T & Z_4 \end{bmatrix} \succeq 0, \ Z^0 = \begin{bmatrix} Z_1^0 & Z_2^0 \\ Z_2^{0T} & Z_4^0 \end{bmatrix} \succeq 0 \tag{21}$$

provides a global optimal solution for the non-convex problem (P) in (20) *when the number of neurons satisfies* $m \geq m^*$ *where*

$$m^* = \operatorname{rank}(Z^*) + \operatorname{rank}(Z^{0*}). \tag{22}$$

Z^* *and* Z^{0*} *denote the optimal solution of the convex program in* (21)*. The optimal network weights can be extracted from* Z^* *and* Z^{0*} *using the Neural Decomposition procedure. It follows that the optimal number of neurons is upper bounded by* $m^* \leq 2(d+1)$*.*

To the best of our knowledge, Theorem 1 provides the first method that globally optimizes a standard non-trivial neural network architecture with computational complexity polynomial in all problem dimensions.

The convex SDP in (21) and the other SDP formulations presented in this paper can be solved using interior point algorithms. Since the Slater's condition holds for these SDPs, the interior point algorithm for these problems

are polynomial time in the real number model, where the representations of the numbers are ignored and only the number of iterations is bounded by a polynomial ([27]).

The proof of Theorem 1 consists of two parts. In the first part, we leverage convex duality to find a lower-bounding convex SDP in Section 3.1. In the second part, we show a matching upper bound via the proposed Neural Decomposition algorithm in Section 4.

3.1 Lower Bound Proof for Theorem 1

In this section we show that the solution of the convex program (21) provides a lower bound for the solution of the non-convex problem (20). In the Neural Decomposition section, we prove, via the method of neural decomposition, that the solution of the convex problem provides also an upper bound, which concludes the proof of Theorem 1.

In proving the lower bound, we leverage duality. Minimizing over first α_j's and then u_j's, we can restate the problem (P) given in (20) as

$$p^* = \min_{u_j \in \mathcal{S}^d; \forall j \in [m]} \; \min_{\alpha_j \in \mathbb{R}; \forall j \in [m], \hat{y} \in \mathbb{R}^n} \ell(\hat{y}, y) + \sum_{j=1}^{m} |\alpha_j| \quad \text{s.t.} \quad \hat{y} = \sum_{j=1}^{m} (X u_j)_+ \alpha_j, \tag{23}$$

where the set $\mathcal{S}^d$ is defined as $\mathcal{S}^d := \{u \in \mathbb{R}^d : \|u\|_2 = 1\}$. Here, we use the notation $p^* := \min(P)$. The dual problem for the inner minimization problem is given by

$$\max_{v \in \mathbb{R}^n} \; -\ell^*(v) \quad \text{s.t.} \quad |v^T (X u_j)_+| \leq \beta, \; \forall j \in [m]. \tag{24}$$

Next, let us call the optimal solution of the following problem d^*

$$d^* = \min_{u_j \in \mathcal{S}^d; \forall j \in [m]} \; \max_{v \in \mathbb{R}^n} \; -\ell^*(v) \quad \text{s.t.} \quad |v^T (X u_j)_+| \leq \beta, \; \forall j \in [m]. \tag{25}$$

By changing the order of the minimization and maximization operations, we obtain the following bound

$$d^* \geq \max_{v \in \mathbb{R}^n} \; -\ell^*(v) \quad \text{s.t.} \quad |v^T (X u)_+| \leq \beta, \; \forall u \in \mathcal{S}^d. \tag{26}$$

We note that the inequality $|v^T (Xu)| \leq$ can equivalently be written as two quadratic (in u) inequalities

$$u^T \left(\sum_{i=1}^{n} x_i x_i^T v_i \right) u + b v^T X u + c v^T \mathbf{1} \geq \quad ;$$
$$u^T \left(\sum_{i=1}^{n} x_i x_i^T v_i \right) u - b v^T X u - c v^T \mathbf{1} \geq \quad . \tag{27}$$

Next, we use the S-procedure given in Corollary 1 to reformulate the problem in (26) as a problem with linear matrix inequality constraints. Corollary 1 is based on Lemma 2 which characterizes the solvability of a quadratic system. The proof of Corollary 1 is given in the supplementary material [4].

Lemma 2 (Proposition 3.1 from [39]) *Let f_1 and f_2 be quadratic functions where f_2 is strictly concave (or strictly convex) and assume that f_2 takes both positive and negative values. Then, the following two statements are equivalent:*

1. *$f_1(u) < 0; f_2(u) = 0$ is not solvable.*
2. *There exists $\in \mathbb{R}$ such that $f_1(u) + f_2(u) \geq 0$, $\forall u$.*

Corollary 1 (S-procedure with equality) $\max_{\|u\|_2=1} u^T Q u + b^T u \geq$ *if and only if there exists $\in \mathbb{R}$ such that*

$$\begin{bmatrix} I & Q & \frac{1}{2} b \\ \frac{1}{2} b^T & & \end{bmatrix} \succeq 0 .$$

Corollary 1 allows us to write the maximization problem in (26) as the equivalent problem given by

$$\begin{aligned} \max_{v \in \mathbb{R}^n, \ _1, \ _2 \in \mathbb{R}} \quad & \ell(\quad v) \\ \text{s.t.} \quad & \begin{bmatrix} _1 I - \sum_{i=1}^{n} x_i x_i^T v_i & \frac{1}{2} b X^T v \\ \frac{1}{2} b v^T X & c \mathbf{1}^T v - _1 \end{bmatrix} \succeq 0 \\ & \begin{bmatrix} _2 I + \sum_{i=1}^{n} x_i x_i^T v_i & \frac{1}{2} b X^T v \\ \frac{1}{2} b v^T X & + c \mathbf{1}^T v - _2 \end{bmatrix} \succeq 0 ; \end{aligned} \tag{28}$$

where we note the two additional variables $_1, \ _2 \in \mathbb{R}$ are introduced. Next, we will find the dual of the problem in (28). Let us first define the following Lagrange multipliers

$$Z = \begin{bmatrix} Z_1 & Z_2 \\ Z_3 & Z_4 \end{bmatrix} \quad ; \quad Z' = \begin{bmatrix} Z_1' & Z_2' \\ Z_3' & Z_4' \end{bmatrix} \quad ; \tag{29}$$

where $Z; Z' \in \mathbb{S}^{(d+1)(d+1)}$ are symmetric matrices, and the dimensions for each block matrix are $Z_1; Z_1' \in \mathbb{S}^{d \ d}$, $Z_2; Z_2' \in \mathbb{R}^{d \ 1}$, $Z_3; Z_3' \in \mathbb{R}^{1 \ d}$, $Z_4; Z_4' \in$

$\mathbb{R}^{1}$ ^{1}. We note that because of the symmetry of Z and Z^0, we have $Z_2^T = Z_3$ and $Z_2{}^{0T} = Z_3^0$. The Lagrangian for the problem in (28) is

$$L(v;\ _1;\ _2; Z; Z^0) = \quad `\ (\quad v) + \ _1 \operatorname{tr}(Z_1) + \ _2 \operatorname{tr}(Z_1^0) \quad a \sum_{i=1} v_i x_i^T (Z_1 \quad Z_1^0) x_i$$

$$bv^T X (Z_2 \quad Z_2^0) + (\qquad _1) Z_4 + (\qquad _2) Z_4^0 \quad c \sum_{i=1} v_i (Z_4 \quad Z_4^0): \quad (30)$$

Maximizing the Lagrangian with respect to v, $_1$, $_2$, we obtain the problem (SDRP) given in (21), which concludes the lower bound part of the proof. In Neural Decomposition section, we introduce a method for decomposing the solution of this convex program (i.e. Z and Z^0) into feasible neural network weights to prove the upper bound.

4 Neural Decomposition

A lower bound on the optimal value of the non-convex problem (P) in (20) is obtained via the solution of the convex program (SDRP) in (21) that we have derived using Lagrangian duality. Now we show that this lower bound is in fact identical to the optimal value of the non-convex problem, thus proving strong duality. Our approach is based on proving an upper bound by constructing neural network weights from the solution of the convex problem such that the convex objective achieves the same objective as the non-convex objective. Suppose that (Z ; Z^0) is a solution to (21). Let us denote the rank of Z by r and the rank of Z^0 by r^0. We will discuss the decomposition for Z and then complete the picture by considering the same decomposition for Z^0 . We begin by noting that Z satisfies the constraints of (21), i.e.,

$$Z \qquad 0;\ \operatorname{tr}(Z_1) = Z_4;\ \text{or}\ \operatorname{tr}\ Z \underbrace{\begin{bmatrix} I_d & 0 \\ 0 & 1 \end{bmatrix}}_{G} = 0: \quad (31)$$

Suppose that we have a decomposition of Z as a sum of rank-1 matrices such that $Z = \sum_{j=1}^{r} p_j p_j^T$ where $p_j \ 2 \ \mathbb{R}^{d+1}$ and $\operatorname{tr}(p_j p_j^T G) = p_j^T G p_j = 0$ for $j = 1; \dots; r$. We show how this can always be done in the following subsection by introducing a new matrix decomposition method, dubbed the *neural decomposition* procedure.

Letting $p_j := \begin{bmatrix} c_j^T & d_j \end{bmatrix}^T$ with $c_j \ 2 \ \mathbb{R}^d$ and $d_j \ 2 \ \mathbb{R}$, we note that $p_j^T G p_j = 0$ implies $\|c_j\|_2^2 = d_j^2$. We may assume $p_j \ 6= 0; \ 8j$ in the decomposition (otherwise we can simply remove zero components), implying $\|c_j\|_2^2 > 0; \ 8j$. Furthermore, this expression for p_j's allows us to establish that

$$\sum_{j=1} p_j p_j^T = \sum_{j=1} \begin{bmatrix} c_j \\ d_j \end{bmatrix} \begin{bmatrix} c_j^T & d_j \end{bmatrix} = \sum_{j=1} \begin{bmatrix} c_j c_j^T & c_j d_j \\ d_j c_j^T & d_j^2 \end{bmatrix} = \begin{bmatrix} Z_1 & Z_2 \\ Z_3 & Z_4 \end{bmatrix} : \quad (32)$$

As a result, we have the following decompositions:

$$Z_1 = \sum_{j=1} c_j c_j^T = \sum_{j=1} u_j u_j^T \|c_j\|_2^2 = \sum_{j=1} u_j u_j^T d_j^2 \tag{33}$$

$$Z_2 = \sum_{j=1} c_j d_j = \sum_{j=1} u_j d_j \|c_j\|_2 = \sum_{j=1} u_j d_j |d_j| \tag{34}$$

$$Z_4 = \sum_{j=1} d_j^2 ; \tag{35}$$

where we have introduced the normalized weights $u_j = \frac{c_j}{\|c_j\|_2}$, $j = 1, \ldots, r$. If $d_j \neq 0$ for some j, we redefine the corresponding p_j as $p_j \leftarrow -p_j$, which does not modify the decomposition $\sum_j p_j p_j^T$ and the equality $p_j^T G p_j = 0$. Hence, without loss of generality, we can assume that $d_j \geq 0$ for all $j = 1, \ldots, r$, which leads to

$$Z_1 = \sum_{j=1} u_j u_j^T d_j^2, \quad Z_2 = \sum_{j=1} u_j d_j^2, \quad Z_4 = \sum_{j=1} d_j^2. \tag{36}$$

Similarly for Z^0 , we will form the following decompositions:

$$Z_1^0 = \sum_{j=1} u_j^0 u_j^{0T} d_j^{0\,2}, \quad Z_2^0 = \sum_{j=1} u_j^0 d_j^{0\,2}, \quad Z_4^0 = \sum_{j=1} d_j^{0\,2}. \tag{37}$$

Considering the decompositions for both Z and Z^0 , finally we obtain a neural network with first layer weights as $\{u_1, \ldots, u_r, u_1^0, \ldots, u_{r^0}^0\}$, and second layer weights as $\{d_1^2, \ldots, d_r^2, d_1^{0\,2}, \ldots, d_{r^0}^{0\,2}\}$. We note that this corresponds to a neural network with $r + r^0$ neurons. If both Z and Z^0 are full rank, then we will have $2(d+1)$ neurons, which is the maximum.

To see why we can use the decompositions of Z and Z^0 to construct neural network weights, we plug the expressions (36) and (37) in the objective of the convex program in (21):

$$\ell(\hat{y}, y) + \sum_{j=1} |d_j^2| + \sum_{j=1} |d_j^{0\,2}| \;, \quad \text{where}$$

$$\hat{y}_i = a x_i^T \sum_{j=1} u_j u_j^T d_j^2 + \sum_{j=1} u_j^0 u_j^{0T} (d_j^{0\,2}) \, x_i + b x_i^T \sum_{j=1} u_j d_j^2 + \sum_{j=1} u_j^0 (d_j^{0\,2})$$
$$+ c \sum_{j=1} d_j^2 + \sum_{j=1} (d_j^{0\,2}) \;, \quad i = 1, \ldots, n. \tag{38}$$

We note that this expression exactly matches the optimal value of the non-convex objective in (20) for a neural network with $r + r^0$ neurons. Also, the unit norm constraints on the first layer weights are satisfied (hence feasible) since

u_j's and u_j^0's are normalized. This establishes that the neural network weights obtained from the solution of the convex program provide an upper bound for the minimum value of the original non-convex problem. Consequently, we have shown that the optimal solution of the convex problem (SDRP) in (21) provides a global optimal solution to the non-convex problem (P) in (20) and this concludes the proof of Theorem 1.

4.1 Neural Decomposition Procedure

Here we describe the procedure for computing the decomposition given by $Z = \sum_{j=1}^{r} p_j p_j^T$ 0 such that $p_j^T G p_j = 0$, $j = 1;\dots;r$ where $G = \begin{smallmatrix} I_d & 0 \\ 0 & 1 \end{smallmatrix}$. The computational complexity of this procedure is at most $O(d^3)$, which is dominated by the cost of one Eigenvalue Decomposition of Z in the initial step. This algorithm is inspired by the constructive proof of the S-procedure given in Lemma 2.4 of [39] with modifications to account for the equalities $p_j^T G p_j = 0$.

Neural Decomposition for PSD Matrices:

0. **Compute a rank-1 decomposition $Z = \sum_{j=1}^{r} p_j p_j^T$.**
   This can be done with the Cholesky decomposition, or the eigenvalue decomposition $Z = \sum_{j=1}^{r} q_j q_j^T$ $_j$. Since Z 0, we have $_j > 0$, for $j = 1;\dots;r$. Then we can obtain the desired rank-1 decomposition $Z = \sum_{j=1}^{r} p_j p_j^T$ by defining $p_j = \sqrt{}_j q_j$, $j = 1;\dots;r$.
1. **If $p_1^T G p_1 = 0$, return $y = p_1$. If not, find a j 2 f2;:::;rg such that $(p_1^T G p_1)(p_j^T G p_j) < 0$.**
   We know such j exists since $\mathrm{tr}(Z\ G) = \sum_{j=1}^{r} p_j^T G p_j = 0$ (this is true since it is one of the constraints of the convex program), and $p_1^T G p_1$ 6= 0. Hence, for at least one j 2 f2;:::;rg, $p_j^T G p_j$ must have the opposite sign as $p_1^T G p_1$.
2. **Return $y = \frac{p_1 + p_j}{\sqrt{1+ ^2}}$ where 2 R satisfies $(p_1 + p_j)^T G(p_1 + p_j) = 0$.**
   We know that such exists since the quadratic equation

$$(p_1 + p_j)^T G(p_1 + p_j) = {}^2 p_j^T G p_j + 2\ p_1^T p_j + p_1^T G p_1 = 0 \tag{39}$$

   has real solutions as the discriminant $4(p_1^T p_j)^2$ $4(p_1^T G p_1)(p_j^T G p_j)$ is positive due to step 1 where we picked j such that $(p_1^T G p_1)(p_j^T G p_j) < 0$. To find , we simply solve the quadratic equation for .
3. **Update r r 1, and then the vectors $p_1;\dots;p_r$ as follows:**
   Remove p_1 and p_j and insert $u = \frac{p_j \sqrt{} p_1}{\sqrt{1+ ^2}}$. Consequently, we will be dealing with the updated matrix Z Z yy^T in the next iteration, which is of rank r 1:

$$Z \quad yy^T = uu^T + \sum_{i=2;i6=j} p_i p_i^T : \tag{40}$$

Fig. 3 Illustration of the neural decomposition procedure for $d = 2$ (i.e. $Z \in \mathbb{R}^{3 \times 3}$). The dashed red arrows correspond to the eigenvectors of Z (q_1, q_2, q_3) and the solid blue arrows show the decomposed vectors p_1 and p_2. In this example, the rank of Z is 2 where q_1 and q_2 are its two principal eigenvectors. The eigenvalue corresponding to the eigenvector q_1 is zero. The light blue colored surface shows the Lorentz cones $z = \sqrt{x^2 + y^2}$ and $z = -\sqrt{x^2 + y^2}$. We observe that the decomposed vectors p_1 and p_2 lie on the boundary of Lorentz cones.

Note that Step 0 is carried out only once and then steps 1 through 3 are repeated $r - 1$ times. At the end of $r - 1$ iterations, we are left with the rank-1 matrix $p_1 p_1^T$ which satisfies $p_1^T G p_1 = 0$ since initial Z satisfies $\mathrm{tr}(Z G) = 0$ and the following $r - 1$ updates are of the form yy^T which satisfies $y^T G y = 0$. If we denote the returned y vectors as y_i for the iteration i and y_r is the last one we are left with, then y_i's satisfy the desired decomposition that $Z = \sum_{i=1}^{r} y_i y_i^T$ and $y_i^T G y_i = 0,\ i = 1, \ldots, r$.

Figure 3 is an illustration of the neural decomposition procedure for a toy example with $d = 2$ where the eigenvectors of Z and the vectors p_j are plotted together. Due to the constraints $p_j^T G p_j = 0,\ j = 1, 2$, the vectors p_j have to lie on the boundary of Lorentz cones[4] $z = \sqrt{x^2 + y^2}$ and $z = -\sqrt{x^2 + y^2}$. Decomposing the solution of the convex problem Z and Z' onto these cones, i.e., neural decomposition, enables the construction of neural network weights from Z and Z'.

5 Standard Weight Decay Formulation is NP-Hard

In the previous sections, we have studied the training of two-layer neural networks and derived a convex program whose solution globally optimizes the non-convex problem. In contrast, Theorem 2 states that if we have quadratic regularization (i.e. weight decay), the resulting optimization problem given in

[4] In special relativity, Lorentz cones describe the path that a flash of light, emanating from a single event traveling in all directions takes through spacetime (see Figure 1.3.1 in [32]).

(41) is an NP-hard problem:

$$(P_{WD}) \quad \min_{\substack{u_j \in \mathbb{R}^d, \\ \beta_j \in \mathbb{R}, \forall j \in [m]}} \ell\left(\sum_{j=1}^m (Xu_j)_+ \beta_j, y\right) + \beta \sum_{j=1}^m \left(|\beta_j|^2 + \|u_j\|_2^2\right). \tag{41}$$

Theorem 2 The two-layer neural network optimization problem with square activation and standard squared ℓ_2 norm regularization, i.e., weight decay, in (41) is NP-hard for any value of m as $\beta \to 0$.

The proof of Theorem 2 is deferred to the appendix. The proof relies on the polynomial time reduction of the problem P_{WD} from the NP-hard problem of phase retrieval. This result illustrates the type of regularization function plays a significant role in the computational tractability of the optimal training problem.

Remark 1 It follows from Theorem 2 that the two-layer neural network training problem with degree-two polynomial activation and unit norm first layer weights and $\sum_j |\beta_j|^p$ as the regularization term with $p < 1$ is also NP-hard for $\beta \to 0$ since it reduces to the square activation case for the polynomial coefficients $a = 1, b = 0, c = 0$.

6 Numerical Results

In this section, we present numerical results that verify the presented theory of the convex formulations along with experiments comparing the test set performance of the derived formulations. All experiments have been run on a MacBook Pro with 16GB RAM.

We have used CVXPY [11,1] for solving the convex SDPs. In particular, we have used the open source solver SCS (splitting conic solver) [35,36] in CVXPY, which is a scalable first order solver for convex cone problems.

Furthermore, we have solved the non-convex problems via backpropagation for which we have used PyTorch [37]. We have used the SGD algorithm for the non-convex models. For all the experiments involving SGD in this section, we show only the results corresponding to the best learning rate that we select via an offline hyperparameter search. The momentum parameter is 0.9. In the plots, the non-convex models are either labeled as `Backpropagation (GD)' or `Backpropagation (SGD)'. The first one, short for gradient descent, means that the batch size is equal to the number of samples n, and the second one, short for stochastic gradient descent, means that the batch size is not n and the exact batch size is explicitly stated in the figure captions.

Figure 4 compares the solution of the non-convex problem via backpropagation and the solution of the corresponding convex program via a convex solver. The training and test costs are shown for a regression task with randomly generated data for the two-layer square activation neural network. We observe that convex SDP takes a much shorter time to optimize and obtains a

Fig. 4 Cost against wall-clock time on the training (left) and test (right) sets for stochastic gradient descent (SGD) and the convex SDP for square activation networks. The solid lines show the training curve of the non-convex model with SGD (with learning rate tuned optimally via extensive grid search) and each di erent colored solid curve corresponds to an independent trial. The dotted horizontal curve shows the cost for the convex SDP and the cross indicates the time that it takes to solve the convex SDP. The dataset X is synthetically generated by sampling from the i.i.d. Gaussian distribution and has dimensions $n = 100; d = 10$. Labels y are generated by a teacher network with 10 planted neurons. The regularization coe cient is $= 10^{6}$ and the batch size for SGD is 10.

globally optimal solution while the SGD algorithm converges to local minima in some of the trials where the initialization is di erent.

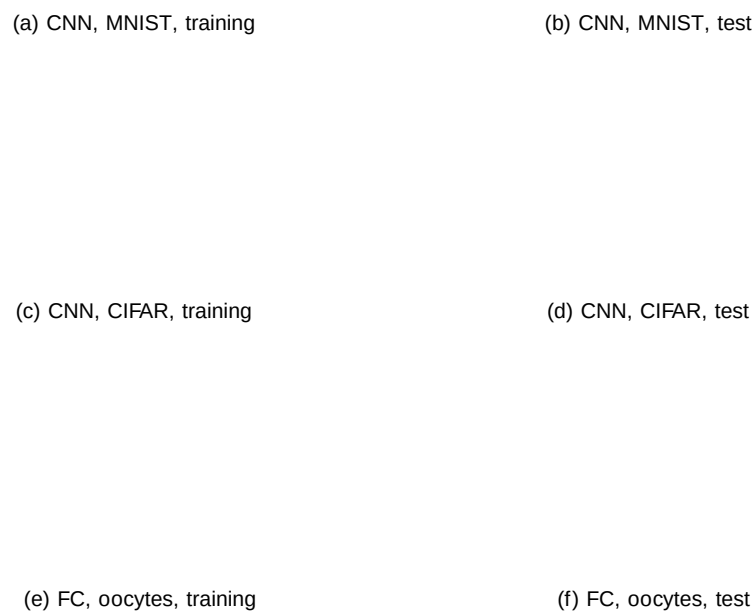
Figure 5 compares the accuracy of the non-convex degree-two polynomial activation model when it is trained with di erent optimizers (SGD and Adam [24]) for a range of step sizes. The plots in Figure 5 show results for the convolutional neural network model. We observe that the convex formulations outperform the non-convex solution via SGD and Adam. The extension of the main result to convolutional neural networks is discussed in the supplementary material [4].

6.1 Comparison with ReLU Networks

We compare the classi cation accuracies for degree-two polynomial activation and ReLU activation in Figure 6 on three di erent binary classi cation UCI datasets. The regularization coe cient has been picked separately for degree-two polynomial activation and ReLU activation networks to maximize the accuracy. Figure 6 demonstrates that the convex SDP shows competitive accuracy performance and faster run times compared to ReLU networks.

6.2 Regularization Parameter

Figure 7 shows how the accuracy changes as a function of the regularization coe cient for the convex problem for two-layer degree-two polynomial activation networks. Figure 7 highlights that the choice of the regularization coe cient is critical in the accuracy performance. In plot a, we see that the value of that maximizes the test set accuracy is $= 10$ for which the optimal number of neuronsm is near 20. We note that for the dataset in plot a, the



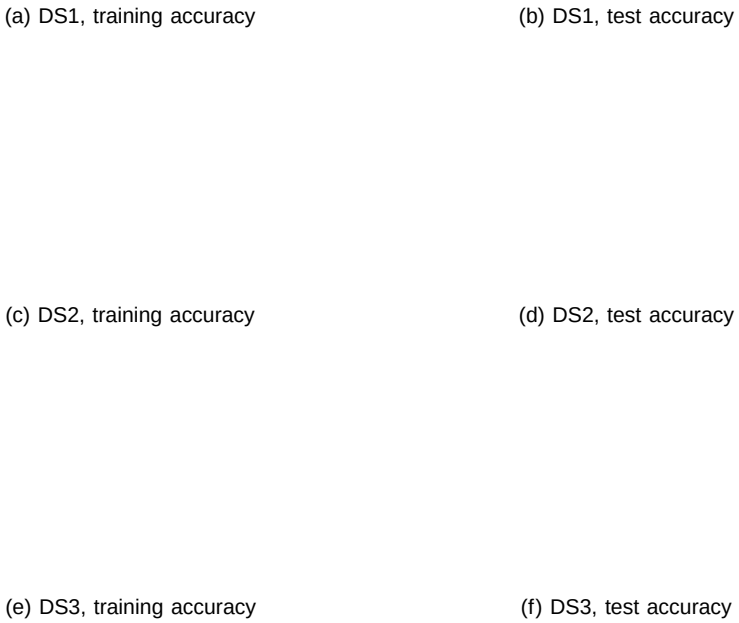
(a) CNN, MNIST, training (b) CNN, MNIST, test

(c) CNN, CIFAR, training (d) CNN, CIFAR, test

(e) FC, oocytes, training (f) FC, oocytes, test

Fig. 5 Classi cation accuracy for various learning rates and optimizers are plotted on the same gure. SGD and Adam are used in solving the non-convex optimization problem. The solid blue lines each correspond to a di erent learning rate for SGD and each dashed green curve corresponds to a di erent learning rate for the Adam algorithm. Plots a, b: CNN with degree-two polynomial activations and global average pooling for binary classi cation on the rst two classes of the MNIST dataset (12000 training samples). Plots c, d: The same architecture as plots a, b and the dataset is the rst two classes of the CIFAR-10 dataset (10000 training samples). Plots e, f: Fully connected architecture for binary classi cation on the dataset oocytes-merluccius-nucleus-4d.

optimal number of neurons is upper bounded by $m \quad 2(d+1) = 32$. Similarly for plot b, the best choice for the regularization coe cient is $= 1$ and the optimal number of neurons for $= 1$ is near 40. Furthermore, we observe that a higher value for tends to translate to a lower optimal number of neurons m (plotted on the right vertical axis). Even though the convex optimization problem in (21) has a xed number of variables (in this case, $2(d+1)^2$) for a given dataset, a low number of neurons is still preferable for many reasons such as inference speed. We observe that the number of neurons can be controlled via the regularization coe cient .



(a) DS1, training accuracy (b) DS1, test accuracy

(c) DS2, training accuracy (d) DS2, test accuracy

(e) DS3, training accuracy (f) DS3, test accuracy

Fig. 6 Comparison of classi cation accuracies for neural networks with ReLU activation, degree-two polynomial activation ($a = 0:09; b = 0:5; c = 0:47$), and the convex SDP. DS1: dataset 1 is the oocytes-merluccius-nucleus-4d ($n = 817; d = 41$), DS2: dataset 2 is the credit approval dataset ($n = 552; d = 15$), DS3: dataset 3 is the breast cancer dataset ($n = 228; d = 9$).

(a) credit approval (b) ionosphere

Fig. 7 Accuracy (left vertical axis) and optimal number of neurons (right vertical axis) against the regularization coe cient on binary classi cation datasets. These results have been obtained using the convex program in (21). The dimensions of the datasets are $n = 552; d = 15$ for plot a and $n = 280; d = 33$ for plot b.

7 Discussion

In this paper, we have studied the optimization of two-layer neural networks with degree-two polynomial activations. We have shown that regularization plays an important role in the tractability of the problems associated with neural network training. We have developed convex programs for the cases where the regularization leads to tractable formulations. Convex formulations are useful since they have many well-known advantages over non-convex optimization such as having to optimize fewer hyperparameters and no risk of getting stuck at local minima.

The methods presented in this work optimize the neural network parameters in a higher dimensional space in which the problem becomes convex. For fully connected neural networks with square activation, the standard non-convex problem requires optimizingm neurons (i.e. ad-dimensional rst layer weight and a 1-dimensional second layer weight per neuron). The convex program for this neural network nds the optimal network parameters in the lifted space$S^{d\ d}$. For degree-two polynomial activations, convex optimization takes place for$Z$ andZ^0 in$S^{(d+1)\ (d+1)}$. We note that the dimensions of the convex programs are polynomial with respect to all problem dimensions. In contrast, the convex program of [38] has2^H variables where$H$ grows exponentially with respect to the rank of the data matrix.

We have used the SCS solver with CVXPY for solving the convex problems in the numerical experiments. It is important to note that there is room for future work in terms of which solvers to use. Solvers speci cally designed for the presented convex programs could enjoy faster run times.

The scope of this work is limited to two-layer neural networks. We note that it is a promising direction to consider the use of our convex programs for two-layer neural networks as building blocks in learning deep neural networks. Many recent works such as [2] and [5] investigate layerwise learning algorithms for deep neural networks. The training of individual layers in layerwise learning could be improved by the presented convex programs since the convex programs can be e ciently solved and eliminate much of the hyperparameter tuning involved in standard neural network training.

Declarations

Funding This work was partially supported by the National Science Foundation under grants IIS-1838179, ECCS-2037304, Facebook Research and the Army Research O ce.

Con icts of interest/Competing interests The authors have no con icts of interest to declare that are relevant to the content of this article.

Availability of data and material (data transparency) The data used in this work are publicly available.

Code availability (software application or custom code) Code will be made available at an online repository.

Ethics approval Not applicable.
Consent to participate No participants, not applicable.
Consent for publication No participants, not applicable.

References

1. Agrawal, A., Verschueren, R., Diamond, S., Boyd, S.: A rewriting system for convex optimization problems. Journal of Control and Decision 5(1), 42{60 (2018)
2. Allen-Zhu, Z., Li, Y.: Backward feature correction: How deep learning performs deep learning. arXiv preprint arXiv:2001.04413 (2020)
3. Arora, R., Basu, A., Mianjy, P., Mukherjee, A.: Understanding deep neural networks with recti ed linear units. In: 6th International Conference on Learning Representations, ICLR 2018 (2018)
4. Bartan, B., Pilanci, M.: Neural spectrahedra and semide nite lifts: Global convex optimization of polynomial activation neural networks in fully polynomial-time. CoRR abs/2101.02429 (2021). URL https://arxiv.org/abs/2101.02429
5. Belilovsky, E., Eickenberg, M., Oyallon, E.: Greedy layerwise learning can scale to imagenet. CoRR abs/1812.11446 (2018). URL http://arxiv.org/abs/1812.11446
6. Bienstock, D., Muñoz, G., Pokutta, S.: Principled deep neural network training through linear programming (2018)
7. Blondel, M., Fujino, A., Ueda, N.: Convex factorization machines. European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD) (2015)
8. Blondel, M., Niculae, V., Otsuka, T., Ueda, N.: Multi-output polynomial networks and factorization machines. In: Proceedings of the 31st International Conference on Neural Information Processing Systems, NIPS'17, p. 3351{3361. Curran Associates Inc., Red Hook, NY, USA (2017)
9. Boyd, S., Vandenberghe, L.: Convex optimization. Cambridge university press (2004)
10. Burer, S.: Copositive programming. In: Handbook on semide nite, conic and polynomial optimization, pp. 201{218. Springer (2012)
11. Diamond, S., Boyd, S.: CVXPY: A Python-embedded modeling language for convex optimization. Journal of Machine Learning Research 17(83), 1{5 (2016)
12. Du, S., Lee, J.: On the power of over-parametrization in neural networks with quadratic activation. In: J. Dy, A. Krause (eds.) Proceedings of the 35th International Conference on Machine Learning, Proceedings of Machine Learning Research , vol. 80, pp. 1329{1338. PMLR, Stockholmsmässan, Stockholm Sweden (2018). URL http://proceedings.mlr.press/v80/du18a.html
13. Ergen, T., Pilanci, M.: Convex geometry of two-layer relu networks: Implicit autoencoding and interpretable models. In: International Conference on Arti cial Intelligence and Statistics, pp. 4024{4033. PMLR (2020)
14. Ergen, T., Pilanci, M.: Implicit convex regularizers of cnn architectures: Convex optimization of two- and three-layer networks in polynomial time. arXiv preprint arXiv:2006.14798 (2020)
15. Ergen, T., Pilanci, M.: Revealing the structure of deep neural networks via convex duality. arXiv preprint arXiv:2002.09773 (2020)
16. Fickus, M., Mixon, D.G., Nelson, A.A., Wang, Y.: Phase retrieval from very few measurements. arXiv preprint arXiv:1307.7176 (2013)
17. Gamarnik, D., K z ldağ, E.C., Zadik, I.: Stationary points of shallow neural networks with quadratic activation function. arXiv preprint arXiv:1912.01599 (2020)
18. Gilad-Bachrach, R., Dowlin, N., Laine, K., Lauter, K., Naehrig, M., Wernsing, J.: Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In: International Conference on Machine Learning, pp. 201{210 (2016)
19. Goel, S., Kanade, V., Klivans, A., Thaler, J.: Reliably learning the relu in polynomial time. In: S. Kale, O. Shamir (eds.) Proceedings of the 2017 Conference on Learning Theory, Proceedings of Machine Learning Research , vol. 65, pp. 1004{1042. PMLR, Amsterdam, Netherlands (2017)

20. Goemans, M.X., Williamson, D.P.: Improved approximation algorithms for maximum cut and satis ability problems using semide nite programming. Journal of the ACM 42, 1115{1145 (1995)
21. Hastad, J.: Some optimal inapproximability results. Journal of the ACM (JACM) 48(4), 798{859 (2001)
22. Hesamifard, E., Takabi, H., Ghasemi, M.: Cryptodl: towards deep learning over encrypted data. In: Annual Computer Security Applications Conference (ACSAC 2016), Los Angeles, California, USA, vol. 11 (2016)
23. Khot, S., Kindler, G., Mossel, E., O'Donnell, R.: Optimal inapproximability results for max-cut and other 2-variable csps? SIAM Journal on Computing 37(1), 319{357 (2007)
24. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. In: 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings (2015). URL http://arxiv.org/abs/1412.6980
25. Lacotte, J., Pilanci, M.: All local minima are global for two-layer relu neural networks: The hidden convex optimization landscape. arXiv preprint arXiv:2006.05900 (2020)
26. Laurent, M., Poljak, S.: On a positive semide nite relaxation of the cut polytope. Linear Algebra and its Applications 223(224), 439{461 (1995)
27. Laurent, M., Rendl, F.: Semide nite programming and integer programming. In: K. Aardal, G. Nemhauser, R. Weismantel (eds.) Discrete Optimization, Handbooks in Operations Research and Management Science , vol. 12, pp. 393{514. Elsevier (2005). DOI https://doi.org/10.1016/S0927-0507(05)12008-8. URL https://www.sciencedirect.com/science/article/pii/S0927050705120088
28. Lederer, J.: No spurious local minima: on the optimization landscapes of wide and deep neural networks (2020)
29. Livni, R., Shalev-Shwartz, S., Shamir, O.: On the computational e ciency of training neural networks. NIPS'14 p. 855{863 (2014)
30. Mannelli, S.S., Vanden-Eijnden, E., Zdeborova, L.: Optimization and generalization of shallow neural networks with quadratic activation functions. arXiv preprint arXiv:2006.15459 (2020)
31. Mishra, P., Lehmkuhl, R., Srinivasan, A., Zheng, W., Popa, R.A.: Delphi: A cryptographic inference system for neural networks. PPMLP'20 p. 27{30 (2020). DOI 10.1145/3411501.3419418. URL https://doi.org/10.1145/3411501.3419418
32. Naber, G.L.: The geometry of Minkowski spacetime: An introduction to the mathematics of the special theory of relativity, vol. 92. Springer Science & Business Media (2012)
33. Nesterov, Y., Wolkowicz, H., Ye, Y.: Semide nite programming relaxations of nonconvex quadratic optimization. In: Handbook of semide nite programming, pp. 361{419. Springer (2000)
34. Obla, S., Gong, X., Alou , A., Hu, P., Takabi, D.: E ective activation functions for homomorphic evaluation of deep neural networks. IEEE Access 8, 153098{153112 (2020)
35. O'Donoghue, B., Chu, E., Parikh, N., Boyd, S.: Conic optimization via operator splitting and homogeneous self-dual embedding. Journal of Optimization Theory and Applications 169(3), 1042{1068 (2016). URL http://stanford.edu/~boyd/papers/scs.html
36. O'Donoghue, B., Chu, E., Parikh, N., Boyd, S.: SCS: Splitting conic solver, version 2.1.2. https://github.com/cvxgrp/scs (2019)
37. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S.: Pytorch: An imperative style, high-performance deep learning library. Advances in Neural Information Processing Systems 32 pp. 8024{8035 (2019). URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf
38. Pilanci, M., Ergen, T.: Neural networks are convex regularizers: Exact polynomial-time convex optimization formulations for two-layer networks. Proceedings of the International Conference on Machine Learning (ICML 2020) (2020)
39. Polik, I., Terlaky, T.: A survey of the s-lemma. SIAM Review 49(3), 371{418 (2007). DOI 10.1137/S003614450444614X. URL https://doi.org/10.1137/S003614450444614X

40. Ramachandran, P., Zoph, B., Le, Q.: Searching for activation functions. arXiv preprint arXiv:1710.05941 (2018). URL https://arxiv.org/pdf/1710.05941.pdf
41. Sahiner, A., Ergen, T., Pauly, J., Pilanci, M.: Vector-output relu neural network problems are copositive programs: Convex analysis of two layer networks and polynomial-time algorithms. arXiv preprint arXiv:2012.13329 (2020)
42. Sahiner, A., Mardani, M., Ozturkler, B., Pilanci, M., Pauly, J.: Convex regularization behind neural reconstruction. arXiv preprint arXiv:2012.05169 (2020)
43. Soltani, M., Hegde, C.: Fast and provable algorithms for learning two-layer polynomial neural networks. IEEE Transactions on Signal Processing **67**(13), 3361–3371 (2019). DOI 10.1109/TSP.2019.2916743
44. Soltanolkotabi, M., Javanmard, A., Lee, J.D.: Theoretical insights into the optimization landscape of over-parameterized shallow neural networks. IEEE Trans. Inf. Theor. **65**(2), 742–769 (2019). DOI 10.1109/TIT.2018.2854560. URL https://doi.org/10.1109/TIT.2018.2854560
45. Trevisan, L., Sorkin, G.B., Sudan, M., Williamson, D.P.: Gadgets, approximation, and linear programming. SIAM Journal on Computing **29**(6), 2074–2097 (2000)
46. Wolkowicz, H., Saigal, R., Vandenberghe, L.: Handbook of semidefinite programming: theory, algorithms, and applications, vol. 27. Springer Science & Business Media (2012)

Appendix

This appendix contains proofs not included in the main body of the paper.

A Proof of Theorem 2

This section breaks down the proof of Theorem 2. At the core of the proof is the polynomial time reduction of the problem from the NP-hard problem of phase retrieval.

A.1 Equivalent Problem Representation

The optimization problem for training a two-layer fully connected neural network with square activation and quadratic regularization can be stated as

$$p^* := \min_{\substack{u_j \in \mathbb{R}^d; \\ \alpha_j \in \mathbb{R}; \forall j \in [m]}} \ell\left(\sum_{j=1}^m (Xu_j)^2 \alpha_j, y\right) + \frac{\beta}{c}\sum_{j=1}^m (|\alpha_j|^2 + \|u_j\|_2^2), \tag{42}$$

where the scaling factor c is the same as before (i.e. $c = 2^{\frac{1}{3}} + 2^{-\frac{2}{3}} \approx 1.88988$). Rescaling $u_j \leftarrow u_j t_j^{1/2}$ and $\alpha_j \leftarrow \alpha_j / t_j$ for $t_j > 0$, $j = 1, \dots, m$, we obtain the following equivalent optimization problem

$$p^* = \min_{\substack{u_j \in \mathbb{R}^d; \\ \alpha_j \in \mathbb{R}; \forall j \in [m]}} \ell\left(\sum_{j=1}^m (Xu_j)^2 \alpha_j, y\right) + \frac{\beta}{c}\sum_{j=1}^m (|\alpha_j|^2/t_j^2 + \|u_j\|_2^2 t_j). \tag{43}$$

Note that the regularization term is convex in t_j for $t_j > 0$. Optimizing the regularization term with respect to t_j leads to $t_j = 2^{1/3} \left(\frac{|\alpha_j|}{\|u_j\|_2}\right)^{2/3}$ and plugging this in yields

$$p^* = \min_{\substack{u_j \in \mathbb{R}^d; \\ \alpha_j \in \mathbb{R}; \forall j \in [m]}} \ell\left(\sum_{j=1}^m (Xu_j)^2 \alpha_j, y\right) + \beta \sum_{j=1}^m |\alpha_j|^{2/3} \|u_j\|_2^{4/3}. \tag{44}$$

Defining scaled weights $\alpha'_j = \alpha_j \|u_j\|_2^2$ and $u'_j = u_j / \|u_j\|_2$, we obtain the equivalent problem

$$p^* = \min_{\substack{u'_j \in \mathbb{R}^d; \ \alpha'_j \in \mathbb{R} \\ \text{s.t. } \|u'_j\|_2 = 1; \forall j \in [m]}} \ell\left(\sum_{j=1}^m (Xu'_j)^2 \alpha'_j, y\right) + \beta \sum_{j=1}^m |\alpha'_j|^{2/3}. \tag{45}$$

This shows that solving the standard weight decay formulation is equivalent to having the power of 2/3 on the second layer weights as regularization and unit norm first layer weights as constraints.

A.2 Hardness Result

We design a data matrix such that the solution coincides with solving the phase retrieval problem which is NP-hard (see [16]). We consider the equality constrained version of (45), i.e., $\lambda \to 0$, which is given by

$$\begin{aligned} \min_{u_j \in \mathbb{R}^d,\ \alpha_j \in \mathbb{R} \atop \text{s.t. } \|u_j\|_2 = 1, \forall j \in [m]} & \sum_{j=1}^{m} |\alpha_j| \\ \text{s.t.} & \sum_{j=1}^{m} (X u_j)_+ \alpha_j = y. \end{aligned} \tag{46}$$

A.2.1 Addition of a Simplex Constraint

Let the first d rows of the data matrix X be $e_1^T, \ldots, e_d^T$ and let the first d entries of y be $1/d$. Then, the constraint $\sum_{j=1}^{m} (X u_j)_+ \alpha_j = y$ implies

$$\sum_{j=1}^{m} u_{jk}^2 \alpha_j = 1/d \text{ for } k = 1, \ldots, d. \tag{47}$$

Summing the above for all $k = 1, \ldots, d$, and noting that $\sum_{k=1}^{d} u_{jk}^2 = 1$ lead to the constraint $\sum_{j=1}^{m} \alpha_j = 1$.

A.2.2 Reduction from the NP-Hard Phase Retrieval and Subset Sum Problem

We let $X = [I; \tilde{X}]$ and $y = [\frac{1}{d}\bar{1}; \tilde{y}]$ to obtain the simplex constraint $\sum_{j=1}^{m} \alpha_j = 1$ as shown in the previous subsection. In this case, the optimization problem reduces to

$$\begin{aligned} \min_{u_j \in \mathbb{R}^d,\ \alpha_j \in \mathbb{R} \atop \text{s.t. } \|u_j\|_2 = 1, \forall j \in [m]} & \sum_{j=1}^{m} |\alpha_j| \\ \text{s.t.} & \sum_{j=1}^{m} (\tilde{X} u_j)_+ \alpha_j = \tilde{y} \\ & \sum_{j=1}^{m} u_{jk}^2 \alpha_j = 1/d, \quad k = 1, \ldots, d \\ & \sum_{j=1}^{m} \alpha_j = 1. \end{aligned} \tag{48}$$

Suppose that there exists a feasible solution $\{\alpha_j, u_j\}_{j=1}^{m}$, which satisfies $k \quad k_0 = 1$, where $\alpha_1 = 1$ and $u_1^T u_1 = 1$ with only one nonzero neuron. Then, it follows from Lemma 3 that this solution is strictly optimal. Consequently, the

problem in (48) is equivalent to

$$\begin{aligned} \text{find}_{u_1 \in \mathbb{R}^d} \quad & u_1 \\ \text{s.t.} \quad & (\tilde{x}_i^T u_1)^2 = \tilde{y}_i, \quad i = 1, \dots, (n - d) \\ & u_{1k}^2 = 1/d, \quad k = 1, \dots, d. \end{aligned} \tag{49}$$

Lemma 3 (ℓ_p minimization recovers 1-sparse solutions when $0 < p < 1$)
Consider the optimization problem

$$\begin{aligned} \min_{\alpha_1, \dots, \alpha_m \in \mathbb{R}} \quad & \sum_{i=1}^{m} |\alpha_i|^p \\ \text{s.t.} \quad & \sum_{i=1}^{m} \alpha_i = 1, \quad \alpha \in C, \end{aligned} \tag{50}$$

where C is a convex set and $p \in (0, 1)$. Suppose that there exists a feasible solution $\alpha \in C$ and $\sum_i \alpha_i = 1$ such that $k - k_0 = 1$. Then, α is strictly optimal with objective value 1. More precisely, any solution with cardinality strictly greater than 1 has objective value strictly larger than 1.

A.2.3 NP-hardness Proof

Subset sum problem given in Definition 3 is a decision problem known to be NP-complete (e.g. [16]). The decision version of the problem in (49) can be stated as follows: Does there exist a feasible u_1? We show that this decision problem is NP-hard via a polynomial time reduction from the subset sum problem.

Definition 3 (Subset sum problem) Given a set of integers A, does there exist a subset A_S whose elements sum to z?

Lemma 4 establishes the reduction of the decision version of (49) from the subset sum problem. The proof is given in the sequel and follows the same approach used in the proof for the NP-hardness of phase retrieval in [16], with the main difference being the additional constraints $u_{1k}^2 = 1/d$, $k = 1, \dots, d$ in (49). Finally, Lemma 4 concludes the proof of Theorem 2.

Lemma 4 *Consider the problem in* (49)*. Let the first d samples of $\tilde{X} \in \mathbb{R}^{(d+1) \times d}$, denoted $\tilde{X}_D \in \mathbb{R}^{d \times d}$, be any diagonal matrix with -1's and $+1$'s on its diagonal, and let the $(d+1)$'st sample be $\tilde{x}_{d+1} = \sqrt{d}\,[a_1 \cdots a_d]^T$. Then, the decision version of the resulting problem returns 'yes' if and only if the answer for the subset sum problem with $A = \{a_1, \dots, a_d\}$ is 'yes'.*

B Proof of Lemma 1

Proof We will denote the set in (18) as S_1 and the set in (19) as S_2 to simplify the notation. We will prove $S_1 = S_2$ by showing $S_1 \subseteq S_2$ and $S_2 \subseteq S_1$.
We first show $S_1 \subseteq S_2$. Let us take a point $S \in S_1$. This implies that S is a matrix of the form

$$t \sum_{j=1} \begin{bmatrix} u_j \\ 1 \end{bmatrix} \begin{bmatrix} u_j \\ 1 \end{bmatrix}^T {}_j = t \sum_{j=1} \begin{bmatrix} u_j u_j^T {}_j & u_j {}_j \\ u_j^T {}_j & {}_j \end{bmatrix} = \begin{bmatrix} t \sum_{j=1}^m u_j u_j^T {}_j & t \sum_{j=1}^m u_j {}_j \\ t \sum_{j=1}^m u_j^T {}_j & t \sum_{j=1}^m {}_j \end{bmatrix} \tag{51}$$

with $\sum_j {}_j = 1$ and $\|u_j\|_2 = 1$ for all j. We note that $\mathrm{tr}(t \sum_{j=1}^m u_j u_j^T {}_j) = t \sum_{j=1}^m \mathrm{tr}(u_j u_j^T {}_j) = t \sum_{j=1}^m \mathrm{tr}(u_j^T u_j) {}_j = t \sum_{j=1}^m {}_j = t$. This shows that S satisfies the equality condition in the definition (19). Now, we show that S is a PSD matrix. Note that each of the rank-1 matrices $\begin{bmatrix} u_j \\ 1 \end{bmatrix} \begin{bmatrix} u_j \\ 1 \end{bmatrix}^T$ is a PSD matrix and since the coefficients ${}_j$'s and t are nonnegative, it follows that S is PSD. This proves that $S \in S_2$.
We next show $S_2 \subseteq S_1$. Let us take a point $S \in S_2$. This implies that S is PSD and $\mathrm{tr}(S_1) = S_4 = t_0 \leq t$. We show in the Neural Decomposition section that it is possible to decompose S via the neural decomposition procedure to obtain the expressions given in (36). It follows that we can write S in the following form

$$S = \frac{t_0}{\sum_{j=1}^m d_j^2} \begin{bmatrix} \sum_{j=1}^m u_j u_j^T d_j^2 & \sum_{j=1}^m u_j d_j^2 \\ \sum_{j=1}^m u_j^T d_j^2 & \sum_{j=1}^m d_j^2 \end{bmatrix}; \tag{52}$$

where the scaling factor $\frac{t_0}{\sum_{j=1}^m d_j^2}$ is to ensure that $\mathrm{tr}(S_1) = S_4 = t_0 \leq t$. It is obvious to see that S is in S_1 when $t_0 = t$ by the definition of S_1 given in (18). When $t_0 < t$, we still have that S is in S_1 which can be seen by noting that S_1 is defined as the convex hull of rank-1 matrices and the zero matrix. We can scale all the rank-1 matrices in the convex combination with $\frac{t_0}{t}$ and change the weight of the zero matrix accordingly.