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

Burak Bartan
Department of Electrical Engineering
Stanford University
bbartan@stanford.edu

Mert Pilanci
Department of Electrical Engineering
Stanford University
pilanci@stanford.edu

January 15, 2021

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 semidefinite programming. Remarkably, we show that semidefinite lifting is always exact and therefore 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 global optimal solution set as their non-convex counterparts. More specifically, the globally optimal two-layer neural network with 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. We extend the results beyond the fully connected architecture to different neural network architectures including networks with vector outputs and convolutional architectures with pooling. 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 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 concerning the computational complexity of optimal training of neural network models. Moreover, the convex optimization perspective provides a more concise parameterization of neural network models that enables further analysis of their interesting properties.

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

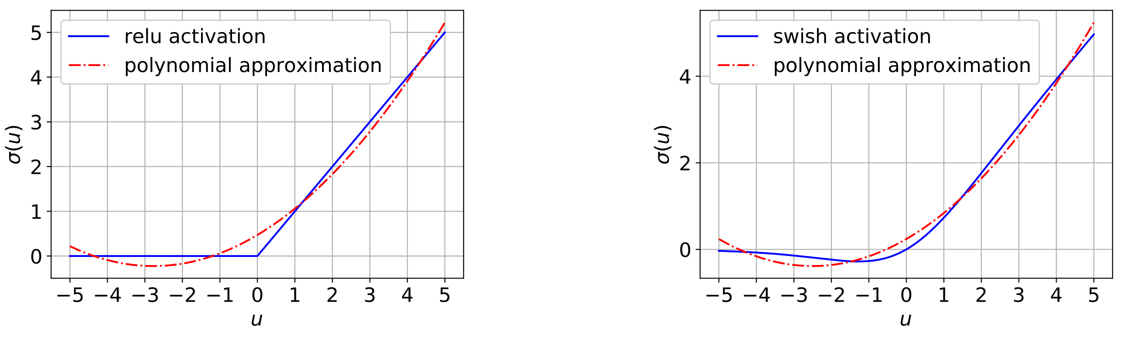


Figure 1: ReLU (left) and swish (right) activation functions and their second degree polynomial approximations. ReLU activation: $\sigma(u) = \max(0, u)$ and its polynomial approximation: $\sigma(u) = 0.09u^2 + 0.5u + 0.47$. Swish activation: $\sigma(u) = u(1 + e^{-u})^{-1}$ and its polynomial approximation: $\sigma(u) = 0.1u^2 + 0.5u + 0.24$.

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 applied to the convex problem to obtain a globally optimal model.

Various types of activation functions were proposed in the literature as nonlinearities in neural network layers. Among the most widely adopted ones is the ReLU (rectified linear unit) activation given by $\sigma(u) = \max(0, u)$. A recently proposed alternative is the swish activation $\sigma(u) = u(1 + e^{-u})^{-1}$, which performs comparably well [45]. 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., $\sigma(u) = au^2 + bu + c$. Although polynomial coefficients a, b, c can be regarded as hyperparameters, it is often sufficient to choose the coefficients in order 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 polynomial activations leverages convex duality and the S-procedure, and can be stated as a simple semidefinite program (SDP). We refer the reader to [44] for a survey of the S-procedure and applications in SDPs. In addition, another commonly used activation function in the literature, quadratic activation, is a special case of polynomial activations ($b = c = 0$) and we devote a separate section to this case (Section 5). The corresponding convex program is an SDP and takes a simpler form.

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 section 1.2) are the following: Our results (1) provide global optimal solutions in fully polynomial time (polynomial in all problem parameters), (2) uncover an important role of the regularizer in computational tractability, (3) hold for arbitrary loss function and other network architectures such as vector output, convolutional and pooling, (4) are independent of the choice of the numerical optimizer and its parameters.

We summarize the types of neural network architectures considered in this work and the corresponding convex problems that we have derived to train them to global optimality 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 $m \geq m^*$. The fifth column, named "construction algorithm", refers to the method for obtaining the optimal neural network weights from the solution of the associated convex program. The last column contains the references to the theorems for each result.

1.1 Overview of Our Contributions

- We show that the standard optimization formulation for training neural networks $f_\theta(x) = \sum_{j=1}^m \sigma(x^T u_j)\alpha_j$ with trainable parameters $\theta = (u_1, \ldots, u_m, \alpha_1, \ldots, \alpha_m)$ and degree two polynomial activations $\sigma(u) = au^2 + bu + c$, training data $X = [x_1, \ldots, x_n]^T \in \mathbb{R}^{n \times d}$, $y \in \mathbb{R}^n$, and ℓ_2^2 regularization on the parameters given by

$$\min_\theta \; \ell(f_\theta(X), y) + \beta \sum_{j=1}^m (\|u_j\|_2^2 + \|\alpha_j\|_2^2) \tag{1}$$

  is computationally intractable via a reduction to the NP-hard subset sum problem, for any value of m.

- Surprisingly, for quadratic activation networks, $\sigma(u) = u^2$, we show that modifying the quadratic weight decay regularization to *cubic regularization*

$$\min_\theta \; \ell(f_\theta(X), y) + \beta \sum_{j=1}^m (\|u_j\|_2^3 + \|\alpha_j\|_2^3) \tag{2}$$

  enables global optimization in fully polynomial time via convex semidefinite programming. The computational complexity is polynomial in all problem parameters (n, d, m).

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

$$\min_{\theta \text{ s.t. } \|u_j\|_2=1, \forall j \in [m]} \; \ell(f_\theta(X), y) + \beta \|\alpha\|_1 \tag{3}$$

  can be equivalently stated as a convex semidefinite problem and globally solved in fully 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 $\beta \to 0$.

- 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 the classical overparameterized 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 convex optimization, 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.

	Non-convex objective	Tractable convex formulation	Upper bound on critical width m^*	Construction algorithm	Thms
Poly (scalar)	$\ell\left(\sum_{j=1}^m \sigma(Xu_j)\alpha_j\,,\, y\right) + \beta \sum_{j=1}^m \lvert\alpha_j\rvert$ s.t. $\lVert u_j\rVert = 1$	Eq (21)	$2(d+1)$	Neural decomp	Thm 3.1
Poly (vector)	$\ell\left(\sum_{j=1}^m \sigma(Xu_j)\alpha_j^T\,,\, Y\right) + \beta \sum_{j=1}^m \lVert\alpha_j\rVert_1$ s.t. $\lVert u_j\rVert = 1$	Eq (72)	$2(d+1)C$	Neural decomp	Thm 7.1
Convolutional	$\ell\left(\sum_{j=1}^m \sum_{k=1}^K \sigma(X_k u_j)\alpha_{jk},\, y\right) + \beta \sum_{j=1}^m \lVert\alpha_j\rVert_1$ s.t. $\lVert u_j\rVert = 1$	Eq (85)	$2(f+1)K^2$	Neural decomp	Thm 8.1
Pooling	$\ell\left(\sum_{j=1}^m \sum_{k=1}^{K/P} \frac{1}{P} \sum_{l=1}^P \sigma(X_{(k-1)P+l} u_j)\alpha_{jk},\, y\right) + \beta \sum_{j=1}^m \lVert\alpha_j\rVert_1$ s.t. $\lVert u_j\rVert = 1$	Eq (95)	$2(f+1)\frac{K^2}{P^2}$	Neural decomp	Thm 9.1
Quad (scalar, cubic reg)	$\ell\left(\sum_{j=1}^m \sigma(Xu_j)\alpha_j,\, y\right) + \beta \sum_{j=1}^m \lvert\alpha_j\rvert$ s.t. $\lVert u_j\rVert = 1$, or $\ell\left(\sum_{j=1}^m \sigma(Xu_j)\alpha_j,\, y\right) + \frac{\beta}{c} \sum_{j=1}^m (\lvert\alpha_j\rvert^3 + \lVert u_j\rVert_2^3)$	Eq (45)	d	Eigen-decomposition	Thm 5.1
Quad (scalar, quad reg)	$\ell\left(\sum_{j=1}^m \sigma(Xu_j)\alpha_j,\, y\right) + \beta \sum_{j=1}^m \lvert\alpha_j\rvert^{2/3}$ s.t. $\lVert u_j\rVert = 1$, or $\ell\left(\sum_{j=1}^m \sigma(Xu_j)\alpha_j,\, y\right) + \frac{\beta}{c} \sum_{j=1}^m (\lvert\alpha_j\rvert^2 + \lVert u_j\rVert_2^2)$	NP-hard (intractable)	-	-	Thm 6.1

Table 1: List of the neural network architectures that we have studied in this work and the corresponding convex programs. Abbreviations are as follows. Poly (scalar): Polynomial activation scalar output, Poly (vector): Polynomial activation vector output, Convolutional: CNN with polynomial activation, Pooling: CNN with polynomial activation and average pooling, Quad (scalar, cubic reg): Quadratic activation scalar output with cubic regularization, Quad (scalar, quad reg): Quadratic 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. $\sigma(u)$ is defined as u^2 for quadratic activation, and $au^2 + bu + c$ for polynomial activation.

- In addition to the fully connected neural network architecture, we derive the equivalent convex programs for various other architectures such as convolutional, pooling and vector output architectures.
- We provide extensive numerical simulations showing that the standard backpropagation approach with or without regularization 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, 49] consider the optimization landscape of a restricted class of neural networks with quadratic activation and quadratic regularization where the second layer weights are *fixed*. They show that when the neural network is overparameterized, i.e., $m \geq d$, 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 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 [35] similarly studies two-layer neural networks with quadratic activation function and squared loss and states results on both optimization and generalization properties. The authors in [19] focus on quadratic 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 [30, 33] it was shown that sufficiently wide ReLU networks have a benign landscape when each layer is sufficiently wide, satisfying $m \geq n+1$.

Another recent work analyzing the training of neural networks with quadratic-like activations for deeper architectures is [2]. Authors in [2] consider 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 $\sigma(u) = u + 0.1u^2$, which the authors prefer over the standard quadratic activation $\sigma(u) = u^2$ to make the neural network training stable. Moreover, the performance of layerwise learning with such activation functions is considerably high, although there is some gap between end-to-end trained models. In addition, neural networks with polynomial activations have immediate applications in encrypted computing [24, 20, 36, 39]. 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 [20], degree two polynomial approximations were shown to be effective for accurate neural network predictions with encryption. These results demonstrate that degree two polynomial activations are quite promising and worth studying from both theoretical and practical perspectives.

In a recent series of papers, the authors derived convex formulations for training ReLU neural networks to global optimality [43, 15, 16, 14, 46, 47]. 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 polynomial activations. Hence, the mathematical techniques involved in deriving the convex programs and the resulting convex programs are substantially different. The convex program derived for ReLU activation in [43] has polynomial time trainability for fixed rank data matrices, whereas the convex programs developed in this work are all polynomial-time trainable with respect to all problem dimensions. More specifically, their convex program is given by

$$\begin{aligned} \min_{\{v_i,w_i\}_{i=1}^P} \ & \frac{1}{2}\left\|\sum_{i=1}^P D_i X(v_i - w_i) - y\right\|_2^2 + \beta \sum_{i=1}^P (\|v_i\|_2 + \|w_i\|_2) \\ \text{s.t.} \ \ & (2D_i - I_n)Xv_i \geq 0,\ (2D_i - I_n)Xw_i \geq 0, \forall i \in [P]\,, \end{aligned} \tag{4}$$

where the neural network weights are constructed from $v_i \in \mathbb{R}^d$ and $w_i \in \mathbb{R}^d$, $i = 1, \ldots, P$. The matrices D_i are diagonal matrices whose diagonal entries consist of $1_{x_1^T u \geq 0}, 1_{x_2^T u \geq 0}, \ldots, 1_{x_n^T u \geq 0}$ for all possible $u \in \mathbb{R}^d$. The number of distinct D_i matrices, denoted by P is the number of hyperplane arrangements corresponding to the data matrix X. It is known that P is bounded by $2r\left(\frac{e(n-1)}{r}\right)^r$ where $r = \text{rank}(X)$ (see [43] 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, 21, 5].

The work in [6] 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 [7]. Polynomial networks are equivalent to quadratic activation networks with an addition of a linear neuron. In [7], the authors consider learning an infinitely wide quadratic activation layer by a greedy algorithm. However, this algorithm does not provide optimal finite width networks even in the quadratic activation case. Furthermore, [34] presents a greedy algorithm for training polynomial networks. The algorithm provided in [34] is based on gradually adding neurons to the neural network to reduce the loss. More recently, [48] considers applying lifting for quadratic 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 quadratic activation and $\sigma(u) = au^2 + bu + c$ where $a, b, c \in \mathbb{R}$ as 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. $\text{vec}(\cdot)$ 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 $1, \ldots, m$ interchangeably.

We use $\ell(\hat{y}, y)$ for convex loss functions throughout the text for both scalar and vector outputs. $\ell^*(v) = \sup_z (v^T z - \ell(z, y))$ denotes the Fenchel conjugate of the function $\ell(\cdot, y)$. Furthermore, we assume $\ell^{**} = \ell$ which holds when ℓ is a convex and closed function [8].

We use $Z \succeq 0$ for positive semidefinite matrices (PSD). $\mathbb{S}$ refers to the set of symmetric matrices. tr refers to matrix trace. $\otimes$ 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 with polynomial activations to Section 2. 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 the following quadratic objective problem with a single quadratic constraint:

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

where Q_1, Q_2 are indefinite, i.e., not assumed to be positive semidefinite. 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

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

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

$$\begin{aligned}\min_{U,u}\quad & \mathrm{tr}(Q_1U)+b_1^Tu+c_1\\ \text{s.t.}\quad & \mathrm{tr}(Q_2U)+b_2^Tu+c_2\leq 0\\ & \begin{bmatrix} U & u\\ u^T & 1\end{bmatrix}\succeq 0\,.\end{aligned} \tag{7}$$

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 Appendix B in [8]). 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 [8, 25].

We also note that the lifting approach $U=uu^T$ and the subsequent relaxation $U\succeq uu^T$ for quadratic programs with more than two quadratic constraints is not tight in general [38, 9]. A notable case with multiple constraints is the NP-hard Max-Cut problem and its SDP relaxation [22]

$$\max_{u_i^2=1,\forall i} u^TQu=\max_{u_i^2=1,\forall i}\mathrm{tr}(Quu^T)\leq \max_{U\succeq 0,\,U_{ii}=1,\forall i}\mathrm{tr}(QU). \tag{8}$$

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

$$\mathbf{conv}\left\{uu^T\ :u_i\in\{-1,+1\}\,\forall i\right\}$$

and other non-integral extreme points [31]. Nevertheless, an approximation ratio of 0.878 can be obtained via the Goemans-Williamson randomized rounding procedure [22]. It is conjectured that this is the best approximation ratio for Max-Cut [27], whereas it can be formally proven to be NP-hard to approximate within a factor of $\frac{16}{17}$ [23, 50]. 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 polynomial activation network $f(x)=\sum_{j=1}^m\sigma(x^Tu_j)\alpha_j$ where $\sigma(u)=u^2+u$, and $\{u_j,\alpha_j\}_{j=1}^m$ are trainable parameters. The corresponding training problem for a given loss function $\ell(\cdot,y)$ and its SDP relaxation are as follows

$$\min_{\{u_j,\alpha_j\}_{j=1}^m}\sum_{x\in\mathcal{X}}\ell\Big(\sum_{j=1}^m((x^Tu_j)^2+x^Tu_j)\alpha_j,\,y\Big)\geq \min_{\{U_j\succeq u_ju_j^T,\alpha_j\}_{j=1}^m}\sum_{x\in\mathcal{X}}\ell\Big(\sum_{j=1}^m x^TU_jx\alpha_j+x^Tu_j\alpha_j,\,y\Big). \tag{9}$$

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 $U_j\succeq u_ju_j^T$ 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 required to address all these concerns.

Although SDP relaxations are extensively studied for various non-convex problems (see e.g. [51] 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 lifting and 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 polynomial activation neural networks can be represented via a class of simple linear matrix inequalities, dubbed *neural spectrahedra* (see Figure 2 for an example), and enables global optimization in fully polynomial time and elucidates their parameterization in convex analytic terms.

1.5 Paper Organization

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 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 polynomial activation networks to global optimality. Quadratic activation neural networks and the hardness result are studied in Section 5 and 6. Vector output and convolutional neural network architectures are studied in Section 7 and 8, respectively and convolutional networks with average pooling is in Section 9. We discuss the implementation details for solving the convex programs and give experimental results in Section 10.

2 Lifted Representations of Networks with Polynomial Activations

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

$$\begin{aligned} f(x) = \sum_{j=1}^m \left(a(x^T u_j)^2 + bx^T u_j + c\right)\alpha_j &= \sum_{j=1}^m \left(\langle axx^T, u_j u_j^T\rangle + \langle bx, u_j\rangle + c\right)\alpha_j \\ &= \left\langle \begin{bmatrix} axx^T \\ bx \\ c \end{bmatrix}, \begin{bmatrix} \sum_{j=1}^m u_j u_j^T \alpha_j \\ \sum_{j=1}^m u_j\alpha_j \\ \sum_{j=1}^m \alpha_j \end{bmatrix} \right\rangle \\ &= \langle \phi(x), \psi(\{u_j, \alpha_j\}_{j=1}^m)\rangle \,, \end{aligned} \tag{10}$$

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

$$\phi(x) := \left(axx^T, bx, c\right) \in \mathbb{R}^{d^2+d+1}.$$

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

$$\psi(\{u_j, \alpha_j\}_{j=1}^m) := \Big(\sum_{j=1}^m u_j u_j^T \alpha_j, \sum_{j=1}^m u_j\alpha_j, \sum_{j=1}^m \alpha_j\Big) \in \mathbb{R}^{d^2+d+1}.$$

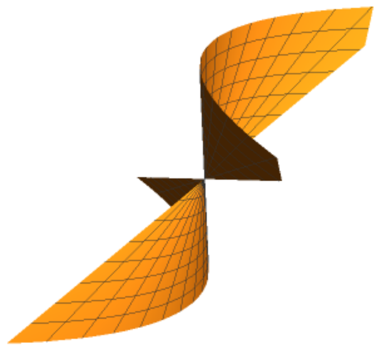
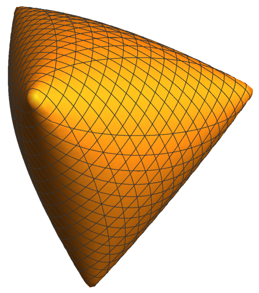



Figure 2: (Left) The Neural Cone $\mathcal{C}_2^1$ described by $(u^2\alpha, u\alpha, \alpha) \in \mathbb{R}^3$ where $u, \alpha \in \mathbb{R}, |u| \leq 1$. (Right) Neural Spectrahedron $\mathcal{M}(1)$ described by $(Z_{11}, Z_{12}, Z_{22}) \in \mathbb{R}^3$ where $Z = \begin{bmatrix} Z_{11} & Z_{12} & Z_{13} \\ Z_{12} & Z_{22} & Z_{23} \\ Z_{13} & Z_{23} & Z_{33} \end{bmatrix} \succeq 0$, $Z_{11} + Z_{22} = Z_{33} \leq 1$ (constrained to the slice $Z_{22} = Z_{11}$ and $Z' = 0$ in (14)).

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., $\sum_{j=1}^m \|u_j\|_2^2$ 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 $\mathcal{C}_2^m \subseteq \mathbb{R}^{d^2+d+1}$ as*

$$\mathcal{C}_2^m := \left\{ \Big(\sum_{j=1}^m u_j u_j^T \alpha_j, \sum_{j=1}^m u_j \alpha_j, \sum_{j=1}^m \alpha_j \Big) : u_j \in \mathbb{R}^d, \|u_j\|_2 = 1, \, \alpha_j \in \mathbb{R} \, \forall j \in [m] \right\}. \tag{11}$$

See Figure 2 (left) for a depiction of $\mathcal{C}_2^1 \subseteq \mathbb{R}^3$ 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 $f(x) = \sum_{j=1}^m \sigma(x^T u_j)\alpha_j$ through a special matrix decomposition procedure which we introduce in Section 4. 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 fully 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 [43] for a convex optimization formulation of networks with ReLU activation, where the worst case computational complexity is $\mathcal{O}((\frac{n}{r})^r)$ with $r = \text{rank}(X)$.

It is equally important that our results characterize neural networks as constrained linear learning methods $\langle \phi(x), \psi \rangle$ in the lifted feature space $\phi(x)$, where the constraints on the lifted parameters ψ 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. We also investigate interesting regularization properties of this convex set, and draw similarities to ℓ_1 norm and nuclear norm. In contrast, Reproducing Kernel Hilbert Space methods and Neural Tangent Kernel approximations [26, 10] are linear learning methods over lifted feature maps where the corresponding parameter constraints are ellipsoids. These approximations fall short of explaining the extraordinary power of finite width neural networks employed in practical applications.

We extend the definition of the Neural Cone to degree k activations as follows.

Definition 2 (**Neural Cone of degree** k). *We define the non-convex cone* $\mathcal{C}_k^m \subseteq \mathbb{R}^{\sum_{i=0}^k d^i}$ *as follows*

$$\mathcal{C}_k^m := \left\{ \Big(\sum_{j=1}^m u_j^{\otimes k} \alpha_j, \cdots, \sum_{j=1}^m u_j \otimes u_j \alpha_j, \sum_{j=1}^m u_j \alpha_j, \sum_{j=1}^m \alpha_j \Big) \,:\, u_j \in \mathbb{R}^d, \|u_j\|_2 = 1,\, \alpha_j \in \mathbb{R}\, \forall j \in [m] \right\} \tag{12}$$

where we use the notation $u^{\otimes k} := \underbrace{u \otimes \cdots \otimes u}_{k\ \textit{times}}$.

It is easy to see that two-layer neural networks with degree k polynomial activations can be represented linearly using the lifted parameter space $\mathcal{C}_k$ and corresponding lifted features. Taking the closure of the union $\{\mathcal{C}\}_{k=0}^{\infty}$, any analytic activation function can be represented in this fashion. In this paper we limit the analysis to the degree 2 case.

Next, we describe a compact set that we call *neural spectrahedron* which describes the lifted parameter space of networks with a constraint on the ℓ_1 norm of output layer weights.

Definition 3. *A neural spectrahedron* $\mathcal{S}_2^m(t) \subseteq \mathbb{R}^{d^2+d+1}$ *is defined as the compact convex set*

$$\mathcal{S}_2^m(t) := \textbf{\textit{conv}} \left\{ \Big(\sum_{j=1}^m u_j u_j^T \alpha_j, \sum_{j=1}^m u_j \alpha_j, \sum_{j=1}^m \alpha_j \Big) : \|u_j\|_2 = 1, \alpha_j \in \mathbb{R}, \forall j = 1, \ldots, m, \sum_{j=1}^m |\alpha_j| \le t \right\} \tag{13}$$

We will show that a neural spectrahedron can be equivalently described as a linear matrix inequality via defining $S_2^m(t) = \big(\mathcal{M}_{11}(t), \mathcal{M}_{12}(t), \mathcal{M}_{22}(t)\big)$ for all $m \ge m^*$ where

$$\mathcal{M}(t) = \left\{ Z - Z' : Z = \begin{bmatrix} Z_1 & Z_2 \\ Z_2^T & Z_4 \end{bmatrix} \succeq 0,\, Z' = \begin{bmatrix} Z_1' & Z_2' \\ Z_2'^T & Z_4' \end{bmatrix} \succeq 0, \mathrm{tr}(Z_1) = Z_4, \mathrm{tr}(Z_1') = Z_4',\, Z_4 + Z_4' \le t \right\}, \tag{14}$$

$Z, Z' \in \mathbb{S}^{(d+1)\times(d+1)}$, $Z_1, Z_1' \in \mathbb{S}^{d\times d}$, $Z_2, Z_2' \in \mathbb{R}^{d\times 1}$ and $Z_4, Z_4' \in \mathbb{R}_+$, and $m^* = m^*(t)$ is a critical number of neurons that satisfies $m^*(0) = 0$ and $m^*(t) \le 2(d+1)\, \forall t$, which will be explicitly defined in the sequel. Therefore, an efficient description of the set $\mathcal{M}(t)$ in terms of linear matrix inequalities enables efficient convex optimization methods in polynomial time. Moreover, it should be noted that in non-convex optimization, the choice of the optimization algorithm and its internal hyperparameters, such as initialization, mini-batching and step sizes have a substantial contribution to the quality of the learned neural network model. This is in stark contrast to convex optimization problems, where optimizer hyperparameters have no effect, and solutions can be obtained in a very robust, efficient and reproducible manner.

2.1 A geometric description of the Neural Spectrahedron for the special case of nonnegative output layer weights

Here we describe a simpler case with the restriction $\alpha_j \geq 0 \, \forall j \in [m]$ in the Neural Cone $\mathcal{C}_2^m$ and we will suppose that $m \geq d+1$. In this special case, let us define the one-sided positive Neural Spectrahedron as

$$ {}^{+}\mathcal{S}_2^m(t) := \mathbf{conv}\left\{ \Big(\sum_{j=1}^m u_j u_j^T \alpha_j, \sum_{j=1}^m u_j \alpha_j, \sum_{j=1}^m \alpha_j \Big) : \|u_j\|_2 = 1, \alpha_j \in \mathbb{R}_+, \forall j = 1, \ldots, m, \sum_{j=1}^m \alpha_j \leq t \right\}. \tag{15} $$

We observe that ${}^{+}\mathcal{S}_2^m(t)$ is identical to the set $\big({}^{+}\mathcal{M}_{11}, {}^{+}\mathcal{M}_{12}, {}^{+}\mathcal{M}_{22} \big) \subseteq \mathbb{R}^{d^2+d+1}$ where

$$ {}^{+}\mathcal{M}(t) : = t\,\mathbf{conv}\left\{ \sum_{j=1}^m \begin{bmatrix} u_j \\ 1 \end{bmatrix} \begin{bmatrix} u_j \\ 1 \end{bmatrix}^T \alpha_j \, : u_j \in \mathbb{R}^d, \|u_j\|_2 = 1, \alpha_j \in \mathbb{R}_+, \sum_{j=1}^m \alpha_j \leq 1 \right\}, \tag{16} $$

which is partitioned as ${}^{+}\mathcal{M}(t) = \begin{bmatrix} {}^{+}\mathcal{M}_{11} & {}^{+}\mathcal{M}_{12} \\ {}^{+}\mathcal{M}_{12}^T & {}^{+}\mathcal{M}_{22} \end{bmatrix}$ where ${}^{+}\mathcal{M}_{11} \subseteq \mathbb{S}^{d\times d}$, ${}^{+}\mathcal{M}_{12} \subseteq \mathbb{R}^{d\times 1}$ and ${}^{+}\mathcal{M}_{22} \subseteq \mathbb{R}_+$.

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

$$ {}^{+}\mathcal{M}(t) : = t\,\mathbf{conv}\left\{ \left\{ \begin{bmatrix} u \\ 1 \end{bmatrix} \begin{bmatrix} u \\ 1 \end{bmatrix}^T : \|u\|_2 = 1 \right\} \cup \mathbf{0} \right\}, \tag{17} $$

where $\mathbf{0}$ is the zero matrix, since $\sum_{j=1}^m \begin{bmatrix} u_j \\ 1 \end{bmatrix} \begin{bmatrix} u_j \\ 1 \end{bmatrix}^T \alpha_j \in \mathbb{S}^{(d+1)\times(d+1)}$ is a positive semidefinite matrix, and hence can be factorized[2] as a convex combination of at most $d+1$ rank-one matrices of the form $\begin{bmatrix} u \\ 1 \end{bmatrix} \begin{bmatrix} u \\ 1 \end{bmatrix}^T$. Note that the zero matrix is included to account for the inequality $\sum_{j=1}^m \alpha_j \leq 1$ in (16). 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 ${}^{+}\mathcal{M}(t)$ given in (17).

Most importantly, the positive Neural Spectrahedron set ${}^{+}\mathcal{M}(t)$ provides a representation of the non-convex Neural Cone $\mathcal{C}_2^m$ via its extreme points. Furthermore, ${}^{+}\mathcal{M}(t)$ has a simple description as a linear matrix inequality provided in the following lemma (the proof can be found in the appendix).

Lemma 2.1. *For $m \geq d+1$, it holds that*

$$ {}^{+}\mathcal{M}(t) = \left\{ Z : \, Z = \begin{bmatrix} Z_1 & Z_2 \\ Z_2^T & Z_4 \end{bmatrix} \succeq 0, \, \mathrm{tr}(Z_1) = Z_4 \leq t \right\}. \tag{18} $$

Therefore the positive Neural Spectrahedron can be represented as the intersection of the positive semidefinite cone and linear inequalities. Moreover, every element of ${}^{+}\mathcal{M}(t)$ can be factorized as $\sum_{j=1}^m \begin{bmatrix} u_j u_j^T \alpha_j & u_j \alpha_j \\ u_j^T \alpha_j & \alpha_j \end{bmatrix}$ for some $\|u_j\|_2 = 1, \alpha_j \geq 0$, $\forall j \in [m]$, $\sum_{j=1}^m \alpha_j \leq t$, which can be identified as an element of the non-convex Neural Cone $\mathcal{C}_2^m$ and a neural network in the lifted parameter space as shown in (10).

[1]This assumption is not required in our later analysis.

[2]We describe the details of this factorization in Section 4.

The assumption $m \geq d+1$ is not required and only used here to illustrate this simpler special case. In the more general case of arbitrary output layer weights $\alpha_j \in \mathbb{R}$, $\forall j \in [m]$, we have the more general linear matrix inequality representation in (14), which is in terms of two positive semidefinite cones and three linear inequalities. In general, such a restriction on the number of neurons m in terms of the dimension d is not necessary. In the next sections, we only require $m \geq m^*$, where m^* can be determined via a convex program. 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 Duality for Polynomial Activation Networks

We consider the non-convex training of a two-layer fully connected neural network with polynomial activation and derive a convex dual optimization problem. The input-output relation for this architecture is

$$f(x) = \sum_{j=1}^{m} \sigma(x^T u_j)\alpha_j \,, \tag{19}$$

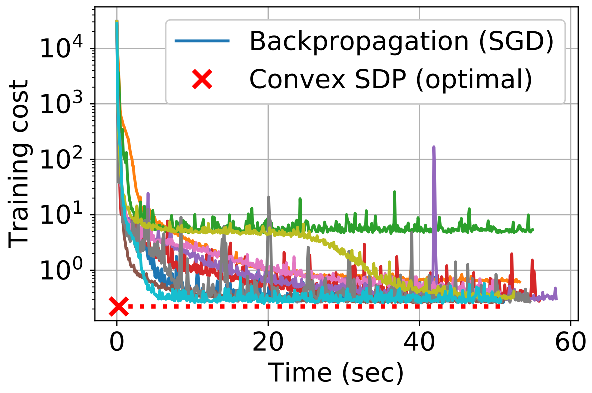
where σ is the degree two polynomial $\sigma(u) = au^2 + bu + c$. This neural network has m neurons with the first layer weights $u_j \in \mathbb{R}^d$ and second layer weights $\alpha_j \in \mathbb{R}$. We refer to this case where $f : \mathbb{R}^d \to \mathbb{R}$ as the scalar output case. Section 7 extends the results to the vector output case.

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 critical 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 Section 5 that this formulation is equivalent to cubic regularization when the activation is quadratic. For the standard ℓ_2^2, i.e., weight decay regularization, we will in fact show that the problem is NP-hard (see Section 6). The training of a network under this setting requires solving the non-convex optimization problem given by

$$p^* = \min_{\{\alpha_j, u_j\}_{j=1}^m, \text{s.t. } \|u_j\|_2 = 1, \forall j} \ell\left(\sum_{j=1}^{m} \sigma(Xu_j)\alpha_j \,,\, y\right) + \beta \sum_{j=1}^{m} |\alpha_j| \,. \tag{20}$$

Theorem 3.1 states the main result for polynomial activation neural networks that the non-convex optimization problem in (20) can be solved globally optimally via a convex problem. Before we state Theorem 3.1, we briefly describe the numerical examples shown in Figure 3 and 4 which compare the solution of the non-convex problem via backpropagation and the solution of the corresponding convex program via a convex solver (see Section 10 for details on the solver). Figure 3 shows the training and test costs on a regression task with randomly generated data for the two-layer quadratic activation neural network. We observe that convex SDP takes a much shorter time to optimize and obtains a globally optimal solution while the SGD algorithm converges to local minima in some of the trials where the initialization is different. Furthermore, Figure 4 compares the classification accuracies for the two-layer vector output polynomial activation network on a multiclass classification problem with real data. The exact statement of the vector output



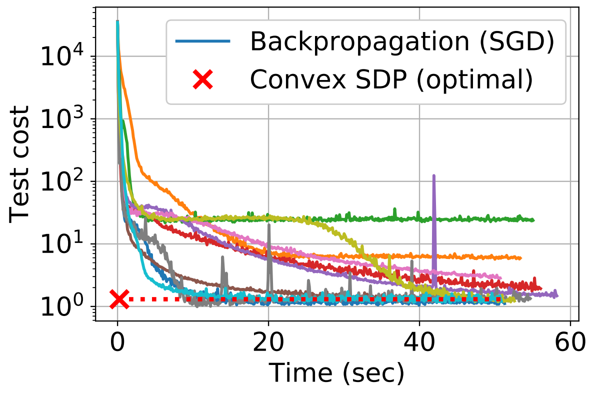
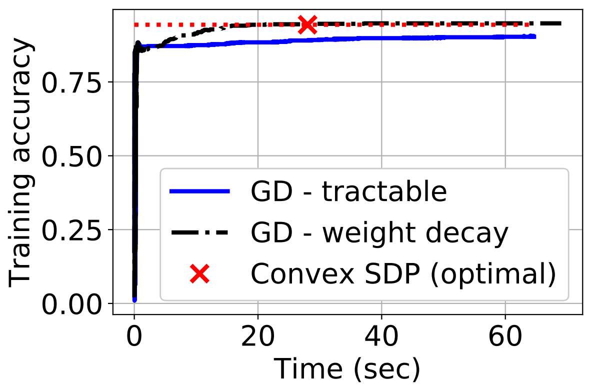


Figure 3: Cost against wall-clock time on the training (left) and test (right) sets for stochastic gradient descent (SGD) and the convex SDP for quadratic activation networks. The solid lines show the training curve of the non-convex model with SGD (with learning rate tuned optimally via extensive grid search) and each different 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 coefficient is $\beta = 10^{-6}$ and the batch size for SGD is 10.



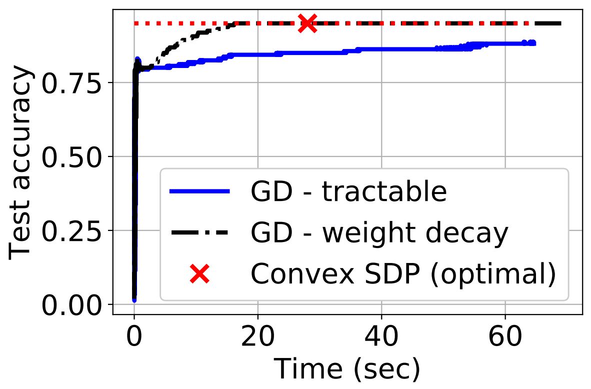


Figure 4: Classification accuracy results on the UCI dataset ”annealing” $(n = 638, d = 31)$ for polynomial activation networks. This is a multiclass classification dataset with $C = 5$ classes. Both training (left) and test (right) set accuracies are shown for the gradient descent (GD) and the convex SDP methods. Legend labels are as follows. *GD - tractable*: The non-convex problem in (65) solved via gradient descent, *GD - weight decay*: Non-convex problem with quadratic regularization on all weights solved via gradient descent, *Convex SDP (optimal)*: The convex problem in (72). Degree two polynomial activation with coefficients $a = 0.09$, $b = 0.5$, $c = 0.47$ is used. The regularization coefficient is $\beta = 1$. The learning rate for GD is optimized offline and only the best performing learning rate is shown. The resulting number of neurons from the convex program is 172.

extension of the main result is provided in Section 7. In Section 10, we present additional numerical results verifying all of the theoretical results on various datasets.

Figure 5 compares the accuracy of the non-convex polynomial activation model when it is trained with different optimizers (SGD and Adam [28]) for a range of step sizes. Figure 5 shows 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 Section 8 and 9.

Theorem 3.1 (Globally optimal convex program for polynomial activation networks)**.** *The solution of the convex problem*

$$\begin{aligned}\min_{Z,Z'} \quad & \ell(\hat{y}, y) + \beta(Z_4 + Z_4') \\ s.t. \quad & \hat{y}_i = a x_i^T (Z_1 - Z_1')x_i + b x_i^T (Z_2 - Z_2') + c(Z_4 - Z_4'), i \in [n] \\ & \operatorname{tr}(Z_1) = Z_4,\ \operatorname{tr}(Z_1') = Z_4' \\ & Z = \begin{bmatrix} Z_1 & Z_2 \\ Z_2^T & Z_4 \end{bmatrix} \succeq 0,\ Z' = \begin{bmatrix} Z_1' & Z_2' \\ Z_2'^T & Z_4' \end{bmatrix} \succeq 0 \end{aligned} \tag{21}$$

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

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

Here Z^* and Z'^* denote the solution of (21)*. The optimal network weights can be extracted from Z^* and Z'^* using the Neural Decomposition procedure given in Section 4. It follows that the optimal number of neurons is upper bounded by $m^* \leq 2(d+1)$.*

The proof of Theorem 3.1 is established in this section and the next. 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 next 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 3.1.

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

$$p^* = \min_{\{u_j\}_{j=1}^m \text{ s.t. } \|u_j\|_2=1, \forall j} \ \min_{\{\alpha_j\}_{j=1}^m, \hat{y}} \ell(\hat{y}, y) + \beta \sum_{j=1}^m |\alpha_j| \quad \text{s.t.} \quad \hat{y} = \sum_{j=1}^m \sigma(Xu_j)\alpha_j\,. \tag{23}$$

The dual problem for the inner minimization problem is given by

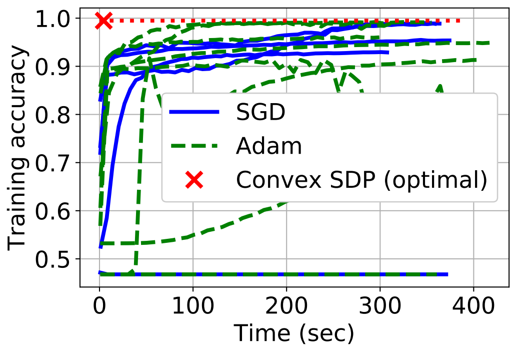
$$\max_v -\ell^*(-v) \quad \text{s.t.} \quad |v^T \sigma(Xu_j)| \leq \beta,\, \forall j\,. \tag{24}$$

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

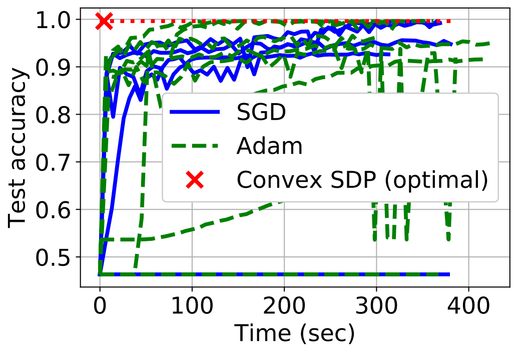
$$d^* = \min_{\{u_j\}_{j=1}^m \text{ s.t. } \|u_j\|_2=1, \forall j} \ \max_{|v^T\sigma(Xu_j)|\leq\beta, \forall j} -\ell^*(-v). \tag{25}$$

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

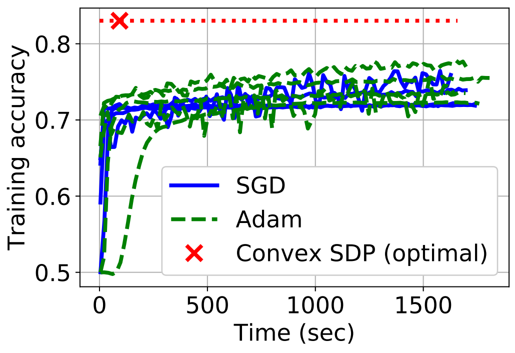
$$d^* \geq \max_{|v^T\sigma(Xu_j)|\leq\beta, \|u_j\|_2=1, \forall j} -\ell^*(-v). \tag{26}$$



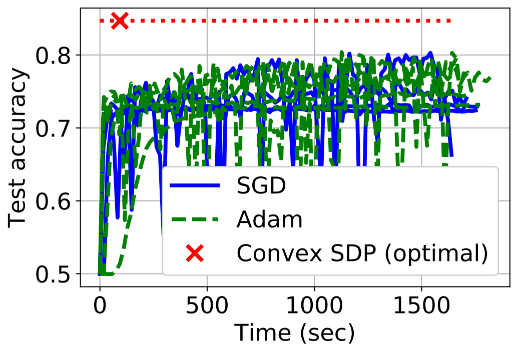
(a) CNN, MNIST, training accuracy



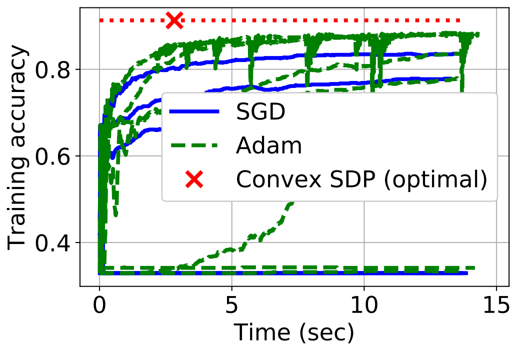
(b) CNN, MNIST, test accuracy



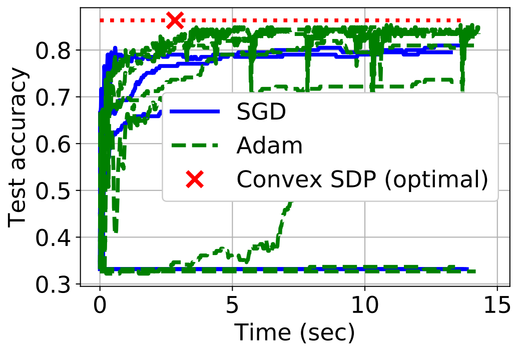
(c) CNN, CIFAR, training accuracy



(d) CNN, CIFAR, test accuracy



(e) Fully connected, oocytes, training accuracy



(f) Fully connected, oocytes, test accuracy

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

We note that the constraints $|v^T\sigma(Xu_j)| \le \beta$ can equivalently be written as two quadratic (in u_j) inequalities for each $j = 1, \dots, m$,

$$u_j^T\left(a\sum_{i=1}^n x_ix_i^Tv_i\right)u_j + bv^TXu_j + cv^T\,\bar{1} \le \beta,\; -u_j^T\left(a\sum_{i=1}^n x_ix_i^Tv_i\right)u_j - bv^TXu_j - cv^T\,\bar{1} \le \beta. \tag{27}$$

Next, we use the S-procedure given in Corollary 3.3 to reformulate the quadratic inequality constraints as linear matrix inequality constraints. Corollary 3.3 is based on Lemma 3.2 which characterizes the solvability of a quadratic system. The proof of Corollary 3.3 is given in the appendix.

Lemma 3.2 (Proposition 3.1 from [44])**.** *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 $\lambda \in \mathbb{R}$ such that $f_1(u) + \lambda f_2(u) \ge 0$, $\forall u$.*

Corollary 3.3 (S-procedure with equality)**.** $\max_{\|u\|_2=1} u^TQu + b^Tu \le \beta$ *if and only if there exists $\lambda \in \mathbb{R}$ such that*

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

Corollary 3.3 allows us to write the maximization problem in (26) as the equivalent problem given by

$$\begin{aligned} &\max -\ell^*(-v) \\ &\text{s.t.}\; \begin{bmatrix} \rho_1 I - a\sum_{i=1}^n x_ix_i^Tv_i & -\frac{1}{2}bX^Tv \\ -\frac{1}{2}bv^TX & \beta - c\,\bar{1}^T\,v - \rho_1 \end{bmatrix} \succeq 0 \\ &\quad\;\; \begin{bmatrix} \rho_2 I + a\sum_{i=1}^n x_ix_i^Tv_i & \frac{1}{2}bX^Tv \\ \frac{1}{2}bv^TX & \beta + c\,\bar{1}^T\,v - \rho_2 \end{bmatrix} \succeq 0\,, \end{aligned} \tag{28}$$

where we note the two additional variables $\rho_1, \rho_2 \in \mathbb{R}$ are introduced. Next, we will find the dual of the problem in (28). Let us first define the following Lagrange multipliers

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

where $Z, Z' \in \mathbb{S}^{(d+1)\times(d+1)}$ are symmetric matrices, and the dimensions for each block matrix are $Z_1, Z_1' \in \mathbb{S}^{d\times d}$, $Z_2, Z_2' \in \mathbb{R}^{d\times 1}$, $Z_3, Z_3' \in \mathbb{R}^{1\times d}$, $Z_4, Z_4' \in \mathbb{R}^{1\times 1}$. We note that because of the symmetry of Z and Z', we have $Z_2^T = Z_3$ and ${Z_2}'^T = Z_3'$. The Lagrangian for the problem in (28) is

$$\begin{aligned} L(v, \rho_1, \rho_2, Z, Z') = -\ell^*(-v) + \rho_1\operatorname{tr}(Z_1) + \rho_2\operatorname{tr}(Z_1') - a\sum_{i=1}^n v_ix_i^T(Z_1 - Z_1')x_i - bv^TX(Z_2 - Z_2') + \\ + (\beta - \rho_1)Z_4 + (\beta - \rho_2)Z_4' - c\sum_{i=1}^n v_i(Z_4 - Z_4'). \end{aligned} \tag{30}$$

Maximizing the Lagrangian with respect to v, ρ_1, ρ_2, we obtain the problem in (21), which concludes the lower bound part of the proof. In the next section, we introduce a method for decomposing the solution of this convex program (i.e. Z^* and Z'^*) into feasible neural network weights to prove the upper bound.

4 Neural Decomposition

We have shown that a lower bound on the optimal value of the non-convex problem in (20) is obtained via the solution of the convex program 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'^*) is a solution to (21). Let us denote the rank of Z^* by r and the rank of Z'^* by r'. We will discuss the decomposition for Z^* and then complete the picture by considering the same decomposition for Z'^*. We begin by noting that Z^* satisfies the constraints of (21), i.e.,

$$Z^* \succeq 0 \text{ and } \operatorname{tr}(Z_1^*) = Z_4^*, \text{ or equivalently } \operatorname{tr}\Big(Z^* \underbrace{\begin{bmatrix} I_d & 0 \\ 0 & -1 \end{bmatrix}}_{G}\Big) = 0. \tag{31}$$

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 $p_j \in \mathbb{R}^{d+1}$ and $\operatorname{tr}(p_j p_j^T G) = p_j^T G p_j = 0$ for $j = 1, \ldots, r$. We show how this can always be done in subsection 4.1 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 $c_j \in \mathbb{R}^d$ and $d_j \in \mathbb{R}$, we note that $p_j^T G p_j = 0$ implies $\|c_j\|_2^2 = d_j^2$. We may assume $p_j \neq 0$, $\forall j$ in the decomposition (otherwise we can simply remove zero components), implying $\|c_j\|_2^2 > 0$, $\forall j$. Furthermore, this expression for p_j's allows us to establish that

$$\sum_{j=1}^r p_j p_j^T = \sum_{j=1}^r \begin{bmatrix} c_j \\ d_j \end{bmatrix} \begin{bmatrix} c_j^T & d_j \end{bmatrix} = \sum_{j=1}^r \begin{bmatrix} c_j c_j^T & c_j d_j \\ d_j c_j^T & d_j^2 \end{bmatrix} = \begin{bmatrix} Z_1^* & Z_2^* \\ Z_3^* & Z_4^* \end{bmatrix}. \tag{32}$$

As a result, we have the following decompositions:

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

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

$$Z_4^* = \sum_{j=1}^r d_j^2\,, \tag{35}$$

where we have introduced the normalized weights $u_j = \frac{c_j}{\|c_j\|_2}$, $j = 1, \ldots, r$. If $d_j \leq 0$ 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

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

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

$$Z_1'^* = \sum_{j=1}^{r'} u_j' u_j'^T d_j'^2, \quad Z_2'^* = \sum_{j=1}^{r'} u_j' d_j'^2, \quad Z_4'^* = \sum_{j=1}^{r'} d_j'^2. \tag{37}$$

Considering the decompositions for both Z^* and Z'^*, finally we obtain a neural network with first layer weights as $\{u_1, \ldots, u_r, u'_1, \ldots, u'_{r'}\}$, and second layer weights as $\{d_1^2, \ldots, d_r^2, -{d'_1}^2, \ldots, -{d'_{r'}}^2\}$. We note that this corresponds to a neural network with $r + r'$ neurons. If both Z^* and Z'^* 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'^* to construct neural network weights, we plug-in the expressions (36) and (37) in the objective of the convex program in (21):

$$\ell(\hat{y}, y) + \beta\Bigg(\sum_{j=1}^{r} |d_j^2| + \sum_{j=1}^{r'} |-{d'_j}^2|\Bigg), \quad \text{where} \quad \hat{y}_i = a x_i^T \Bigg(\sum_{j=1}^{r} u_j u_j^T d_j^2 + \sum_{j=1}^{r'} u'_j {u'_j}^T (-{d'_j}^2)\Bigg) x_i +$$
$$+ b x_i^T \Bigg(\sum_{j=1}^{r} u_j d_j^2 + \sum_{j=1}^{r'} u'_j (-{d'_j}^2)\Bigg) + c\Bigg(\sum_{j=1}^{r} d_j^2 + \sum_{j=1}^{r'} (-{d'_j}^2)\Bigg), \quad i = 1, \ldots, n\,. \tag{38}$$

We note that this expression exactly matches the optimal value of the non-convex objective in (20) for a neural network with $r + r'$ neurons. Also, the unit norm constraints on the first layer weights are satisfied (hence feasible) since u_j's and u'_j'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 (21) provides a global optimal solution to the non-convex problem (20) and this concludes the proof of Theorem 3.1.

4.1 Neural Decomposition Procedure

Here we describe the procedure for computing the decomposition $Z^* = \sum_{j=1}^{r} p_j p_j^T \succeq 0$ such that $p_j^T G p_j = 0$, $j = 1, \ldots, r$. The computational complexity of this procedure is at most $O(nd^2)$, which is dominated by Eigenvalue Decomposition of Z^*. This algorithm is inspired by the constructive proof of the S-procedure given in Lemma 2.4 of [44] 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 eigenvalue decomposition $Z^* = \sum_{j=1}^r q_j q_j^T \lambda_j$. Since $Z^* \succeq 0$, we have $\lambda_j > 0$, 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 $p_j = \sqrt{\lambda_j} q_j$, $j = 1, \dots, r$.

1. **If $p_1^T G p_1 = 0$, return $y = p_1$. If not, find a $j \in \{2, \dots, r\}$ 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 $p_1^T G p_1 \neq 0$. Hence, for at least one $j \in \{2, \dots, r\}$, $p_j^T G p_j$ must have the opposite sign as $p_1^T G p_1$.

2. **Return $y = \frac{p_1 + \alpha p_j}{\sqrt{1+\alpha^2}}$ where $\alpha \in \mathbb{R}$ satisfies $(p_1 + \alpha p_j)^T G (p_1 + \alpha p_j) = 0$.**

   We know that such α exists since the quadratic equation

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

   has real solutions since 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 α, we simply solve the quadratic equation for α.

3. **Update $r \leftarrow r - 1$, and then the vectors $p_1, \dots, p_r$ as follows:**

   Remove p_1 and p_j and insert $u = \frac{p_j - \alpha p_1}{\sqrt{1+\alpha^2}}$. Consequently, we will be dealing with the updated matrix $Z^* \leftarrow Z^* - yy^T$ in the next iteration, which is of rank $r - 1$:

   $$Z^* - yy^T = uu^T + \sum_{i=2, i\neq j}^{r} p_i p_i^T. \tag{40}$$

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 $y_i^T G y_i = 0$, $i = 1, \dots, r$.

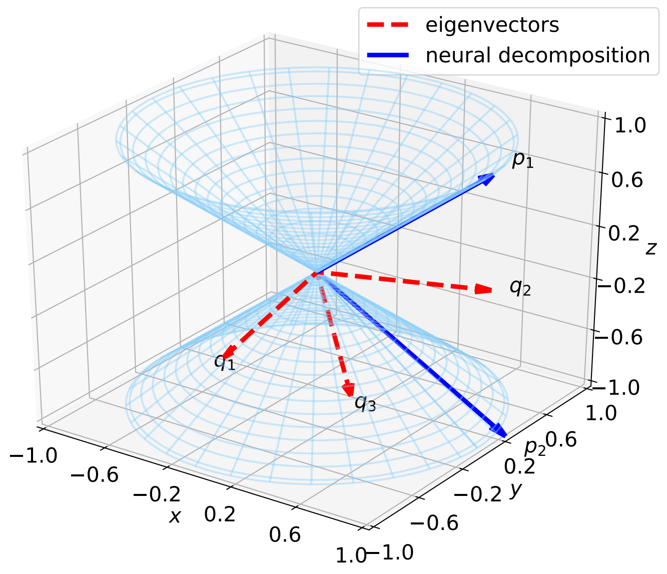


Figure 6: Illustration of the neural decomposition procedure for $d = 2$ (i.e. $Z^* \in \mathbb{R}^{3\times 3}$). The dashed red arrows correspond to the eigenvectors of Z^* (q_1, q_2, q_3) and the solid blue arrows show the decomposed vectors p_1 and p_2. In this example, the rank of Z^* is 2 where 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.

Figure 6 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[3] $z = \sqrt{x^2 + y^2}$ 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 Quadratic Activation Networks

In this section, we derive the corresponding convex program when the activation function is quadratic, i.e., $\sigma(u) = u^2$. The resulting convex problem takes a simpler form than the polynomial activation case. We start by noting that the bound in (26) holds for any activation function. The inequalities $|v^T \sigma(Xu_j)| \le \beta$ however lead to different constraints than the polynomial activation case. Note that $|v^T (Xu_j)^2| \le \beta$ is equivalent to the inequalities

$$u_j^T \left(\sum_{i=1}^n x_i x_i^T v_i \right) u_j \le \beta \quad \text{and} \quad u_j^T \left(-\sum_{i=1}^n x_i x_i^T v_i \right) u_j \le \beta \,. \tag{41}$$

The constraint $\max_{u_j : \|u_j\|_2 = 1} |v^T (Xu_j)^2| \le \beta$ can be expressed as largest eigenvalue inequalities

$$\lambda_{\max} \left(\sum_{i=1}^n x_i x_i^T v_i \right) \le \beta \quad \text{and} \quad \lambda_{\max} \left(-\sum_{i=1}^n x_i x_i^T v_i \right) \le \beta \,, \tag{42}$$

[3]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 [37]).

where $\lambda_{\max}$ denotes the maximum eigenvalue. Next, representing the largest eigenvalue constraints as linear matrix inequality constraints, we arrive at the following maximization problem

$$\begin{aligned} \max_{v} \quad & -\ell^*(-v) \\ \text{s.t.} \quad & \sum_{i=1}^{n} x_i x_i^T v_i - \beta I_d \preceq 0, \quad -\sum_{i=1}^{n} x_i x_i^T v_i - \beta I_d \preceq 0. \end{aligned} \tag{43}$$

Writing the Lagrangian for (43) as $L(v, Z_1, Z_2) = -\ell^*(-v) - \sum_{i=1}^n v_i x_i^T (Z_1 - Z_2) x_i + \beta \operatorname{tr}(Z_1 + Z_2)$ with $Z_1, Z_2 \in \mathbb{S}^{d\times d}$ and maximizing with respect to v, we obtain the following convex problem

$$\min_{Z_1, Z_2 \succeq 0} \ell\left(\begin{bmatrix} x_1^T (Z_1 - Z_2) x_1 & \dots & x_n^T (Z_1 - Z_2) x_n \end{bmatrix}^T, y\right) + \beta \operatorname{tr}(Z_1 + Z_2). \tag{44}$$

Replacing $Z = Z_1 - Z_2$, where $Z_1 \succeq 0, Z_2 \succeq 0$, we recall that any matrix Z can be uniquely decomposed in this form thanks to the Moreau decomposition onto the cone of positive definite matrices and its polar dual, which is the set of negative semidefinite matrices. In particular, suppose that the eigenvalue decomposition of Z is $Z = \sum_j \lambda_j z_j z_j^T$. Then, Z_1 and Z_2 are uniquely determined by $Z_1 = \sum_{j:\lambda_j>0} \lambda_j z_j z_j^T$ and $Z_2 = -\sum_{j:\lambda_j<0} \lambda_j z_j z_j^T$. Note that $\operatorname{tr}(Z_1 + Z_2) = \sum_{j:\lambda_j>0} \lambda_j + \sum_{j:\lambda_j<0} (-\lambda_j) = \sum_j |\lambda_j| = \|Z\|_*$ is the sum of the absolute values of the eigenvalues of Z, which is equivalent to the nuclear norm for symmetric matrices. Consequently, this leads to the following simplified problem with nuclear norm regularization:

$$\begin{aligned} \min_{Z=Z^T} \quad & \ell(\hat{y}, y) + \beta \|Z\|_* \\ \text{s.t.} \quad & \hat{y}_i = x_i^T Z x_i, \quad i = 1, \dots, n. \end{aligned} \tag{45}$$

Theorem 5.1 states the main result for the global optimization of quadratic activation neural networks. The rest of this section is devoted to the proof and interpretation of Theorem 5.1.

Theorem 5.1 (Globally optimal convex program for quadratic activation cubic regularization networks)**.** *The solution of the convex problem in* (45) *provides a global optimal solution to the non-convex problem for quadratic activation and cubic regularization given in* (47) *when the number of neurons satisfies $m \geq m^*$ where*

$$m^* = \operatorname{rank}(Z^*). \tag{46}$$

The optimal neural network weights are determined from the solution of the convex problem via eigenvalue decomposition of Z^* and the rescaling given in (51)*. The optimal number of neurons is upper bounded by $m^* \leq d$ since* $\operatorname{rank}(Z^*) \leq d$*.*

5.1 Strong Duality for Quadratic Activation

We have shown that a lower bound on the non-convex problem for quadratic activation is given by the nuclear norm regularized convex objective. Now we show that this lower bound is in fact identical to the non-convex problem. Suppose that Z^* is a solution to (45). Let us decompose Z^* via eigenvalue decomposition as $Z^* = \sum_j \lambda_j z_j z_j^T$. We can generate an upper bound on the non-convex problem by constructing neural network parameters as $\alpha_j = \lambda_j$, and $u_j = z_j$ with objective value $\ell\left(\sum_j (X z_j)^2 \lambda_j, y\right) + \beta \sum_j |\lambda_j|$. Noting that this value exactly matches the optimal value of the convex objective in (45), we conclude that the optimal solution of (45) provides a global optimal solution to the non-convex problem.

5.2 Equivalent Non-convex Problem: Quadratic Activation with Cubic Regularization

We now show that the non-convex problem with unit norm first layer weights and the ℓ_1 norm regularized second layer weights is in fact equivalent to the non-convex problem with cubic regularization on all the weights. Let us consider the *unconstrained* problem with cubic regularization:

$$p^* := \min_{\{\alpha_j, u_j\}_{j=1}^m} \ell\left(\sum_{j=1}^m (Xu_j)^2 \alpha_j,\, y\right) + \frac{\beta}{c}\sum_{j=1}^m (|\alpha_j|^3 + \|u_j\|_2^3)\,, \tag{47}$$

where $c = 2^{\frac{1}{3}} + 2^{-\frac{2}{3}} \approx 1.88988$. Rescaling the variables $u_j \leftarrow u_j t_j^{1/2}$ and $\alpha_j \leftarrow \alpha_j / t_j\,, \forall j$ for $t_j > 0$, $j = 1, \ldots, m$ yields

$$p^* = \min_{\{\alpha_j, u_j\}_{j=1}^m} \ell\left(\sum_{j=1}^m (Xu_j)^2 \alpha_j,\, y\right) + \frac{\beta}{c}\sum_{j=1}^m (|\alpha_j|^3/t_j^3 + \|u_j\|_2^3 t_j^{3/2})\,. \tag{48}$$

Noting the regularization term is convex in t_j for $t_j > 0$ and optimizing it with respect to t_j, we obtain $t_j = 2^{2/9}\left(\frac{|\alpha_j|}{\|u_j\|_2}\right)^{2/3}$. Plugging the expression for t_j in yields

$$p^* = \min_{\{\alpha_j, u_j\}_{j=1}^m} \ell\left(\sum_{j=1}^m (Xu_j)^2 \alpha_j,\, y\right) + \beta\sum_{j=1}^m |\alpha_j|\|u_j\|_2^2\,. \tag{49}$$

Now we define the scaled second layer weights $\alpha_j' = \alpha_j \|u_j\|_2^2$. Noting that $(Xu_j)^2\alpha_j = (X\frac{u_j}{\|u_j\|_2})^2\alpha_j'$ and defining $u_j' = u_j/\|u_j\|_2$, we obtain the equivalent problem with the ℓ_1 norm of the second layer weights as the regularization term

$$p^* = \min_{\{\alpha_j', u_j'\}_{j=1}^m, \text{s.t.} \|u_j'\|_2 = 1, \forall j} \ell\left(\sum_{j=1}^m (Xu_j')^2 \alpha_j',\, y\right) + \beta\sum_{j=1}^m |\alpha_j'|\,. \tag{50}$$

5.2.1 Rescaling

We note that the weights α_j and u_j that the eigenvalue decomposition of the solution of (45) gives are scaled versions of the weights of the problem with cubic regularization in (47). The solution to the problem in (47) can be constructed by rescaling the weights as

$$u_j \leftarrow u_j\sqrt{t_j'}, \quad \alpha_j \leftarrow \frac{\alpha_j}{t_j'} \quad \text{, where} \quad t_j' = 2^{2/9}|\alpha_j|^{2/3} \quad j = 1, \ldots, m. \tag{51}$$

This concludes the proof of Theorem 5.1.

5.3 Comparison with Polynomial Activation Networks

In this subsection, we list the important differences between the results for quadratic activation and polynomial activation neural networks. The convex program for the quadratic activation network does not have the equality constraints that appear in the convex program for the polynomial activation. In addition, for the quadratic activation, the upper bound on the critical width m^* is d while it is $2(d+1)$ for the polynomial activation case.

We note that in the case of quadratic activation, the optimal neural network weights are determined from eigenvalue decomposition of Z^*. This results in the first layer weights to be orthonormal because they can be chosen as the eigenvectors of the real and symmetric matrix Z^*. In contrast, we do not have this property for polynomial activations as the associated optimal weights are determined via neural decomposition. In this case, the resulting hidden neurons are not necessarily orthogonal, which shows that the Neural Decomposition is a type of non-orthogonal matrix decomposition. This can also be seen in Figure 6.

5.4 Constructing Multiple Globally Optimal Solutions in the Neural Network Parameter Space

Once we find an optimal Z^* using the SDP in (45), we can transform it to the neural network parameter space with at most d neurons using the eigenvalue decomposition of Z^* as $Z^* = \sum_{j=1}^{d} u_j u_j^T \alpha_j$. However, we can also generate a neural network with an arbitrary number of neurons, which is also optimal. We now describe this construction below for an arbitrary number of neurons $m \geq 2d$. Let us pick an arbitrary $m/2 \times d$ matrix H with orthonormal columns, i.e.,

$$I_d = H^T H = \sum_{j=1}^{m/2} h_j h_j^T \,, \tag{52}$$

where $h_1, \ldots, h_{m/2}$ are the rows of H and we assume $m/2 \geq d$. One can generate such matrices using randomized Haar ensemble, or partial Hadamard matrices. Then, we can represent Z^* using

$$\begin{aligned} Z^* &= Z^* H^T H \\ &= \sum_{j=1}^{m/2} Z^* h_j h_j^T \,. \end{aligned}$$

Since Z^* is a symmetric matrix, $\sum_{j=1}^{m/2} Z^* h_j h_j^T$ is also symmetric, and we can write

$$Z^* = \frac{1}{2} \sum_{j=1}^{m/2} (Z^* h_j h_j^T + h_j h_j^T Z^*) \,.$$

Finally, for each term in the above summation, we employ the symmetrization identity

$$xy^T + yx^T = \frac{1}{2} \left((x+y)(x+y)^T - (x-y)(x-y)^T \right) \,,$$

valid for any $x, y \in \mathbb{R}^d$. We arrive at the representation

$$\begin{aligned} Z^* &= \frac{1}{4} \sum_{j=1}^{m/2} ((Z^* h_j + h_j)(Z^* h_j + h_j)^T - (Z^* h_j - h_j)(Z^* h_j - h_j)^T) && (53) \\ &= \sum_{j=1}^{m} u_j u_j^T \alpha_j \,, && (54) \end{aligned}$$

where $u_j = Z^* h_j + h_j$, $\alpha_j = 1/4$ for $j = 1, \ldots, m/2$ and $u_j = Z^* h_j - h_j$, $\alpha_j = -1/4$ for $j = m/2 + 1, \ldots, m$.

Since the matrix H is arbitrary, one can map an optimal Z^* matrix from the convex semidefinite program to infinitely many optimal solutions in the neural network parameterization space.

6 Standard Weight Decay Formulation is NP-Hard

In Section 5, we have studied two-layer neural networks with quadratic activation and cubic regularization and derive a convex program whose solution globally optimizes the non-convex problem. In this section, we show that if, instead of cubic regularization, we have quadratic regularization (i.e. weight decay), the resulting optimization problem is an NP-hard problem.

Theorem 6.1. *The two-layer neural network optimization problem with quadratic activation and standard ℓ_2-squared regularization, i.e., weight decay, in* (55) *is NP-hard for any value of m as $\beta \to 0$.*

The remainder of this section breaks down the proof of Theorem 6.1. At the core of the proof is the polynomial-time reduction of the problem to the NP-hard problem of phase retrieval.

6.1 Reduction to an Equivalent Problem

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

$$p^* := \min_{\{\alpha_j, u_j\}_{j=1}^m} \ell\left(\sum_{j=1}^m (Xu_j)^2 \alpha_j,\, y\right) + \frac{\beta}{c}\sum_{j=1}^m (|\alpha_j|^2 + \|u_j\|_2^2)\,, \tag{55}$$

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

$$p^* = \min_{\{\alpha_j, u_j\}_{j=1}^m} \ell\left(\sum_{j=1}^m (Xu_j)^2 \alpha_j,\, y\right) + \frac{\beta}{c}\sum_{j=1}^m (|\alpha_j|^2/t_j^2 + \|u_j\|_2^2 t_j)\,. \tag{56}$$

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

$$p^* = \min_{\{\alpha_j, u_j\}_{j=1}^m} \ell\left(\sum_{j=1}^m (Xu_j)^2 \alpha_j,\, y\right) + \beta \sum_{j=1}^m |\alpha_j|^{2/3} \|u_j\|_2^{4/3}\,. \tag{57}$$

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

$$p^* = \min_{\{\alpha_j', u_j'\}_{j=1}^m \text{ s.t.} \|u_j'\|_2 = 1, \forall j} \ell\left(\sum_{j=1}^m (Xu_j')^2 \alpha_j',\, y\right) + \beta \sum_{j=1}^m |\alpha_j'|^{2/3}\,. \tag{58}$$

This shows that solving the standard weight decay formulation is equivalent to solving a 2/3-norm penalized problem with unit norm first layer weights.

6.2 Hardness Result

We design a data matrix such that the solution coincides with solving the phase retrieval problem which is NP-hard (see [18]). We consider the equality constrained version of (58), i.e., $\beta \to 0$, which is given by

$$\min_{\{\alpha_j,u_j\}_{j=1}^m \text{ s.t. } \|u_j\|_2=1,\forall j} \sum_{j=1}^m |\alpha_j|^{2/3}$$
$$\text{s.t.} \quad \sum_{j=1}^m (Xu_j)^2\alpha_j = y\,. \tag{59}$$

6.2.1 Addition of a Simplex Constraint

Let the first d rows of the data matrix X be $e_1^T, \dots, e_d^T$ and let the first d entries of y be $1/d$. Then, the constraint $\sum_{j=1}^m (Xu_j)^2 = y$ implies

$$\sum_{j=1}^m u_{jk}^2\alpha_j = 1/d \text{ for } k = 1,\dots,d\,. \tag{60}$$

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

6.2.2 Reduction to 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

$$\min_{\{\alpha_j,u_j\}_{j=1}^m \text{ s.t. } \|u_j\|_2=1,\forall j} \sum_{j=1}^m |\alpha_j|^{2/3}$$
$$\text{s.t.} \quad \sum_{j=1}^m (\tilde{X}u_j)^2\alpha_j = \tilde{y}$$
$$\sum_{j=1}^m u_{jk}^2\alpha_j = 1/d, \quad k = 1,\dots,d$$
$$\sum_{j=1}^m \alpha_j = 1\,. \tag{61}$$

Suppose that there exists a feasible solution $\{\alpha_j^*, u_j^*\}_{j=1}^m$, which satisfies $\|\alpha^*\|_0 = 1$, where $\alpha_1^* = 1$ and $u_1^{*T}u_1^* = 1$ with only one nonzero neuron. Then, it follows from Lemma 6.2 that this solution is strictly optimal. Consequently, the problem in (61) is equivalent to

$$\begin{aligned} \text{find} \quad & u_1 \\ \text{s.t.} \quad & (\tilde{x}_i^T u_1)^2 = \tilde{y}_i, \quad i = 1,\dots,(n-d) \\ & u_{1k}^2 = 1/d, \quad k = 1,\dots,d\,. \end{aligned} \tag{62}$$

Lemma 6.2 (ℓ_p minimization recovers 1-sparse solutions when $0 < p < 1$)**.**
Consider the optimization problem

$$\min_{\alpha_1,\dots,\alpha_m} \sum_{i=1}^m |\alpha_i|^p$$
$$\textit{s.t.} \quad \sum_{i=1}^m \alpha_i = 1,\, \alpha \in \mathcal{C}\,, \tag{63}$$

where $\mathcal{C}$ is a convex set and $p \in (0,1)$. Suppose that there exists a feasible solution $\alpha^* \in \mathcal{C}$ and $\sum_i \alpha_i^* = 1$ such that $\|\alpha^*\|_0 = 1$. 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.

6.2.3 NP-hardness Proof

Subset sum problem given in Definition 4 is a decision problem known to be NP-complete (e.g. [18]). The decision version of the problem in (62) 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 to the subset sum problem.

Definition 4 (Subset sum problem)**.** *Given a set of integers $\mathcal{A}$ and an integer z, does there exist a subset $\mathcal{A}_S$ whose elements sum to z?*

Lemma 6.3 establishes the reduction of the decision version of (62) to the subset sum problem. The proof is provided in the appendix and follows the same approach used in the proof for the NP-hardness of phase retrieval in [18], with the main difference being the additional constraints $u_{1k}^2 = 1/d$, $k = 1, \ldots, d$ in (62). Finally, Lemma 6.3 concludes the proof of Theorem 6.1.

Lemma 6.3. *Consider the problem in (62). 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 $\tilde{x}_{d+1} = \sqrt{d}\begin{bmatrix} a_1 & \ldots & a_d \end{bmatrix}^T$. Then, the decision version of the resulting problem returns 'yes' if and only if the answer for the subset sum problem with $\mathcal{A} = \{a_1, \ldots, a_d\}$ is 'yes'.*

Remark 6.1. *It follows from Theorem 6.1 that the two-layer neural network training problem with polynomial activation and unit norm first layer weights and $\sum_j |\alpha_j|^p$ as the regularization term with $p < 1$ is also NP-hard for $\beta \to 0$ since it reduces to the quadratic activation case for the polynomial coefficients $a = 1, b = 0, c = 0$.*

7 Vector Output Networks

The derivations until this point have been for neural network architectures with scalar outputs, i.e., $y_i \in \mathbb{R}$. In this section, we turn to the vector output case $y_i \in \mathbb{R}^C$ where C is the output dimension, and derive a convex problem that has the same optimal value as the non-convex neural network optimization problem. We exploit the same techniques described in the scalar output case except for the part for constructing the *vector* second layer weights from the solution of the convex program. In the scalar output case, the convex problem is over the symmetric matrices Z, Z' and in the vector output case, the optimization is over C such matrix pairs Z_k, Z'_k, $k = 1, \ldots, C$.

We begin our treatment of the vector output case by considering the neural network defined by

$$f(x) = \sum_{j=1}^{m} \sigma(x^T u_j)\alpha_j^T\,, \tag{64}$$

where $\alpha_j \in \mathbb{R}^C$, $j = 1, \ldots, m$ are the *vector* second layer weights. Note that in the scalar output case, the second layer weights α_j were scalars. Taking the regularization to be the ℓ_1 norm of the second layer weights, the neural network training requires solving the following non-convex optimization problem

$$p^* = \min_{\{u_j, \alpha_j\}_{j=1}^m, \text{s.t. } \|u_j\|_2 = 1, \forall j} \ell\left(\sum_{j=1}^{m} \sigma(Xu_j)\alpha_j^T\,,\, Y\right) + \beta \sum_{j=1}^{m} \|\alpha_j\|_1\,, \tag{65}$$

where $Y \in \mathbb{R}^{n\times C}$ is the output matrix. Equivalently,

$$p^* = \min_{\{u_j\}_{j=1}^m \text{ s.t. } \|u_j\|_2=1, \forall j} \; \min_{\{\alpha_j\}_{j=1}^m, \hat{Y}} \ell\left(\hat{Y}, Y\right) + \beta \sum_{j=1}^m \|\alpha_j\|_1 \quad \text{s.t.} \quad \hat{Y} = \sum_{j=1}^m \sigma(Xu_j)\alpha_j^T \,. \tag{66}$$

The dual problem for the inner minimization problem is given by

$$\max_v -\ell^*(-v) \quad \text{s.t.} \quad |v_k^T \sigma(Xu_j)| \le \beta \,, \forall j, k \,, \tag{67}$$

where $v \in \mathbb{R}^{n\times C}$ is the dual variable and $v_k \in \mathbb{R}^n$ is the k'th column of v.

Theorem 7.1 gives the main result of this section.

Theorem 7.1 (Globally optimal convex program for polynomial activation vector output networks)**.** *The solution of the convex problem in* (72) *provides a global optimal solution for the vector output non-convex problem in* (65) *when the number of neurons satisfies $m \ge m^*$ where*

$$m^* = \sum_{k=1}^C (\text{rank}(Z_k^*) + \text{rank}({Z'_k}^*)). \tag{68}$$

The optimal neural network weights are determined from the solution of the convex problem via the neural decomposition procedure for each Z_k^* and ${Z'_k}^*$ and the construction given in (75)*. The optimal number of neurons is upper bounded by $m^* \le 2(d+1)C$.*

Proof of Theorem 7.1. Applying the S-procedure for the constraints in the dual problem (67), we obtain the following maximization problem

$$\begin{aligned}
&\max -\ell^*(-v) \\
&\text{s.t.} \begin{bmatrix} \rho_{k,1} I - a\sum_{i=1}^n x_i x_i^T v_{i,k} & -\frac{1}{2} b X^T v_k \\ -\frac{1}{2} b v_k^T X & \beta - c\,\bar{1}^T v_k - \rho_{k,1} \end{bmatrix} \succeq 0, \quad k = 1, \dots, C \\
&\quad\;\; \begin{bmatrix} \rho_{k,2} I + a\sum_{i=1}^n x_i x_i^T v_{i,k} & \frac{1}{2} b X^T v_k \\ \frac{1}{2} b v_k^T X & \beta + c\,\bar{1}^T v_k - \rho_{k,2} \end{bmatrix} \succeq 0, \quad k = 1, \dots, C \,.
\end{aligned} \tag{69}$$

Next, let us introduce the following Lagrange multipliers

$$Z_k = \begin{bmatrix} Z_{k,1} & Z_{k,2} \\ Z_{k,3} & Z_{k,4} \end{bmatrix} \in \mathbb{S}^{(d+1)\times(d+1)}, \quad Z'_k = \begin{bmatrix} Z'_{k,1} & Z'_{k,2} \\ Z'_{k,3} & Z'_{k,4} \end{bmatrix} \in \mathbb{S}^{(d+1)\times(d+1)}, \quad k = 1, \dots, C. \tag{70}$$

Then, the Lagrangian is

$$\begin{aligned}
&L\left(v, \{\rho_{k,1}, \rho_{k,2}, Z_k, Z'_k\}_{k=1}^C\right) = \\
&= -\ell^*(-v) + \sum_{k=1}^C \left(\rho_{k,1}\,\text{tr}(Z_{k,1}) + \rho_{k,2}\,\text{tr}(Z'_{k,1})\right) - a\sum_{k=1}^C \sum_{i=1}^n v_{i,k} x_i^T (Z_{k,1} - Z'_{k,1}) x_i - b\sum_{k=1}^C v_k^T X (Z_{k,2} - Z'_{k,2}) + \\
&+ \sum_{k=1}^C \left((\beta - \rho_{k,1}) Z_{k,4} + (\beta - \rho_{k,2}) Z'_{k,4}\right) - c\sum_{k=1}^C \sum_{i=1}^n v_{k,i} (Z_{k,4} - Z'_{k,4}) \,.
\end{aligned} \tag{71}$$

Finally maximizing the Lagrangian leads to the following convex SDP:

$$\begin{aligned}
\min_{\{Z_k = Z_k^T, Z'_k = {Z'_k}^T\}_{k=1}^C} \; & \ell(\hat{Y}, Y) + \beta \sum_{k=1}^C (Z_{k,4} + Z'_{k,4}) \\
\text{s.t.} \quad & \hat{Y}_{ik} = a x_i^T (Z_{k,1} - Z'_{k,1}) x_i + b x_i^T (Z_{k,2} - Z'_{k,2}) + c(Z_{k,4} - Z'_{k,4}), \quad i \in [n], \, k \in [C] \\
& \text{tr}(Z_{k,1}) = Z_{k,4}, \;\; \text{tr}(Z'_{k,1}) = Z'_{k,4}, \quad k = 1, \dots, C \\
& Z_k \succeq 0, \;\; Z'_k \succeq 0, \quad k = 1, \dots, C \,.
\end{aligned} \tag{72}$$

We construct the neural network weights from the optimal solution of the convex program as follows. We follow the neural decomposition procedure from Section 4 for extracting neurons from each of the matrices Z_k^* and ${Z'_k}^*$, $k = 1, \ldots, C$. The decompositions for Z_k^* will be of the form

$$Z_{k,1}^* = \sum_{j=1}^{r_k} u_{k,j} u_{k,j}^T d_{k,j}^2, \quad Z_{k,2}^* = \sum_{j=1}^{r_k} u_{k,j} d_{k,j}^2, \quad Z_{k,4}^* = \sum_{j=1}^{r_k} d_{k,j}^2. \tag{73}$$

Then, the weights due to Z_k^*, $k = 1, \ldots, C$ are determined as follows:

$$\begin{aligned} \text{First layer weights:} &\quad \{u_{1,1}, u_{1,2}, \ldots, u_{1,r_1}\}, \ldots, \{u_{C,1}, u_{C,2}, \ldots, u_{C,r_C}\} \\ \text{Second layer weights:} &\quad \{d_{1,1}^2 e_1^T, d_{1,2}^2 e_1^T, \ldots, d_{1,r_1}^2 e_1^T\}, \ldots, \{d_{C,1}^2 e_C^T, d_{C,2}^2 e_C^T, \ldots, d_{C,r_C}^2 e_C^T\}, \end{aligned} \tag{74}$$

where e_k denotes the k'th C-dimensional unit vector, and r_k is the rank of the matrix Z_k^*. In short, the matrix Z_k^* with rank r_k leads to the first layer weights $\{u_{k,1}, u_{k,2}, \ldots, u_{k,r_k}\}$ and the second layer weights $\{d_{k,1}^2 e_k^T, d_{k,2}^2 e_k^T, \ldots, d_{k,r_k}^2 e_k^T\}$. The weights due to ${Z'_k}^*$, $k = 1, \ldots, C$ are determined the same way. Then, we reach the following neural network construction:

$$f(X) = \sum_{k=1}^{C} \sum_{j=1}^{r_k} \sigma(X u_{k,c}) d_{k,j}^2 e_k^T + \sum_{k=1}^{C} \sum_{j=1}^{r'_k} \sigma(X u'_{k,c}) {d'_{k,j}}^2 e_k^T. \tag{75}$$

Finally, the total number of neurons that the convex problem finds is $\sum_{k=1}^{C}(r_k + r'_k)$. The maximum number of neurons occurs if all Z_k^* and ${Z'_k}^*$ are full rank, and this corresponds to a maximum total of $2(d+1)C$ neurons.

We plug the decomposition expressions given in (73) in the convex program in (72) to conclude that the optimal value of the convex program is an upper bound for the non-convex optimization problem (65). The k'th entry of the estimate for the i'th training sample is

$$\begin{aligned} \hat{Y}_{ik} &= a x_i^T \left(\sum_{j=1}^{r_k} u_{k,j} u_{k,j}^T d_{k,j}^2 + \sum_{j=1}^{r'_k} u'_{k,j} {u'_{k,j}}^T (-{d'_{k,j}}^2) \right) x_i + b x_i^T \left(\sum_{j=1}^{r_k} u_{k,j} d_{k,j}^2 + \sum_{j=1}^{r'_k} u'_{k,j} (-{d'_{k,j}}^2) \right) + \\ &\quad + c \left(\sum_{j=1}^{r_k} d_{k,j}^2 + \sum_{j=1}^{r'_k} (-{d'_{k,j}}^2) \right) \\ &= \sum_{j=1}^{r_k} \sigma(x_i^T u_{k,j}) d_{k,j}^2 + \sum_{j=1}^{r'_k} \sigma(x_i^T u'_{k,j}) (-{d'_{k,j}}^2). \end{aligned} \tag{76}$$

It follows that the output vector for the i'th sample is

$$\hat{y}_i = \sum_{k=1}^{C} \sum_{j=1}^{r_k} \sigma(x_i^T u_{k,j}) d_{k,j}^2 e_k^T + \sum_{k=1}^{C} \sum_{j=1}^{r'_k} \sigma(x_i^T u'_{k,j}) (-{d'_{k,j}}^2) e_k^T. \tag{77}$$

We note that this output is of the same form as the non-convex case (66). We also need to check that the regularization term is equivalent to the sum of ℓ_1 norms of the second layer weights:

$$\begin{aligned} \beta \sum_{k=1}^{C} (Z_{k,4} + Z'_{k,4}) &= \beta \sum_{k=1}^{C} \sum_{j=1}^{r_k} d_{k,j}^2 + \beta \sum_{k=1}^{C} \sum_{j=1}^{r'_k} {d'_{k,j}}^2 \\ &= \beta \sum_{k=1}^{C} \sum_{j=1}^{r_k} \|d_{k,j}^2 e_k^T\|_1 + \beta \sum_{k=1}^{C} \sum_{j=1}^{r'_k} \| - {d'_{k,j}}^2 e_k^T\|_1, \end{aligned} \tag{78}$$

which is of the form $\beta \sum_{j=1}^{m} \|\alpha_j\|_1$. Hence, the neural network weights that we obtain via the neural decomposition procedure lead to an upper bound for the original non-convex optimization problem. This concludes the proof that the optimal solution of the convex problem (72) provides a global optimal solution to the non-convex problem (65). □

8 Convolutional Neural Networks

In this section, we consider two-layer convolutional networks with a convolutional first layer and a fully connected second layer. We will denote the filter size by f. Let us denote the patches of a data sample x by $x_1, \ldots, x_K$ where the patches have the same dimension as the filters, i.e., $x_k \in \mathbb{R}^f$. The stride and padding do not affect the below derivations as they can be readily handled when forming the patches. The output of this network is expressed as:

$$f(x) = \sum_{j=1}^{m} \sum_{k=1}^{K} \sigma(x_k^T u_j)\alpha_{jk}\,, \tag{79}$$

where $u_j \in \mathbb{R}^f$ denotes the j'th filter. We will take the regularization to be the ℓ_1 norm of the second layer weights $\alpha_j = \begin{bmatrix}\alpha_{j1} & \ldots & \alpha_{jK}\end{bmatrix}^T \in \mathbb{R}^K$, $j = 1, \ldots, m$:

$$p^* = \min_{\{u_j\}_{j=1}^m \text{ s.t. } \|u_j\|_2=1, \forall j} \; \min_{\{\alpha_j\}_{j=1}^m, \hat{y}} \ell(\hat{y}, y) + \beta \sum_{j=1}^{m} \|\alpha_j\|_1 \quad \text{s.t.} \quad \hat{y} = \sum_{j=1}^{m} \sum_{k=1}^{K} \sigma(X_k^T u_j)\alpha_{jk} \tag{80}$$

where we use $X_k \in \mathbb{R}^{n\times f}$ to denote the matrix with the k'th patch of all the data samples. The dual for the inner minimization problem is given by

$$\max_v -\ell^*(-v) \quad \text{s.t.} \quad |v^T \sigma(X_k u_j)| \le \beta\,, \forall j, k\,. \tag{81}$$

We state the main result of this section in Theorem 8.1.

Theorem 8.1 (Globally optimal convex program for polynomial activation convolutional neural networks)**.** *The solution of the convex problem in* (85) *provides a global optimal solution for the non-convex convolutional neural network problem in* (79) *when the number of filters is at least* $(\text{rank}(Z_k^*) + \text{rank}(Z_k'^*))$ *and equivalently, the number of neurons satisfies* $m \ge m^*$ *where*

$$m^* = K \sum_{k=1}^{K} (\text{rank}(Z_k^*) + \text{rank}(Z_k'^*)). \tag{82}$$

The optimal neural network weights are determined from the solution of the convex problem via the neural decomposition procedure for each Z_k^* *and* $Z_k'^*$. *The optimal number of filters is upper bounded by* $2(f+1)K$ *and the optimal number of neurons is upper bounded by* $m^* \le 2(f+1)K^2$.

Proof of Theorem 8.1. We apply the S-procedure to replace the constraints of (81) with equivalent LMI constraints and this yields

$$\begin{aligned}
&\max -\ell^*(-v)\\
&\text{s.t.} \begin{bmatrix} \rho_{k,1} I - a\sum_{i=1}^n x_{i,k}x_{i,k}^T v_i & -\frac{1}{2} b X_k^T v \\ -\frac{1}{2} b v^T X_k & \beta - c\bar{1}^T v - \rho_{k,1} \end{bmatrix} \succeq 0, \quad k = 1, \ldots, K\\
&\quad\;\; \begin{bmatrix} \rho_{k,2} I + a\sum_{i=1}^n x_{i,k}x_{i,k}^T v_i & \frac{1}{2} b X_k^T v \\ \frac{1}{2} b v^T X_k & \beta + c\bar{1}^T v - \rho_{k,2} \end{bmatrix} \succeq 0, \quad k = 1, \ldots, K\,,
\end{aligned} \tag{83}$$

where $x_{i,k} \in \mathbb{R}^f$ denotes the k'th patch of the i'th data sample. The Lagrangian is as follows

$$L\left(v, \{\rho_{k,1}, \rho_{k,2}, Z_k, Z'_k\}_{k=1}^K\right) =$$
$$= -\ell^*(-v) + \sum_{k=1}^K \left(\rho_{k,1} \operatorname{tr}(Z_{k,1}) + \rho_{k,2} \operatorname{tr}(Z'_{k,1})\right) - a \sum_{k=1}^K \sum_{i=1}^n v_i x_{i,k}^T (Z_{k,1} - Z'_{k,1}) x_{i,k} - b \sum_{k=1}^K v^T X_k (Z_{k,2} - Z'_{k,2}) +$$
$$+ \sum_{k=1}^K \left((\beta - \rho_{k,1}) Z_{k,4} + (\beta - \rho_{k,2}) Z'_{k,4}\right) - c \sum_{k=1}^K \sum_{i=1}^n v_i (Z_{k,4} - Z'_{k,4}), \tag{84}$$

where Z_k, Z'_k are $(f+1) \times (f+1)$ dimensional symmetric matrices. Maximizing the Lagrangian with respect to v, $\rho_{k,1}$, $\rho_{k,2}$, $k = 1, \ldots, K$ yields the convex SDP

$$\begin{aligned} \min_{\{Z_k = Z_k^T, Z'_k = Z'^T_k\}_{k=1}^K} \quad & \ell(\hat{y}, y) + \beta \sum_{k=1}^K (Z_{k,4} + Z'_{k,4}) \\ \text{s.t.} \quad & \hat{y}_i = a \sum_{k=1}^K x_{i,k}^T (Z_{k,1} - Z'_{k,1}) x_{i,k} + b \sum_{k=1}^K x_{i,k}^T (Z_{k,2} - Z'_{k,2}) + c \sum_{k=1}^K (Z_{k,4} - Z'_{k,4}), \quad i \in [n] \\ & \operatorname{tr}(Z_{k,1}) = Z_{k,4}, \; \operatorname{tr}(Z'_{k,1}) = Z'_{k,4}, \quad k = 1, \ldots, K \\ & Z_k \succeq 0, \; Z'_k \succeq 0, \quad k = 1, \ldots, K. \end{aligned} \tag{85}$$

We now show that the convex program in (85) provides an upper bound for the non-convex problem via the same strategy that we have used for the vector output case in Section 7. We construct the neural network weights from each of the matrices Z_k^* and Z'^*_k, $k = 1, \ldots, K$ via neural decomposition:

$$Z_{k,1}^* = \sum_{j=1}^{r_k} u_{k,j} u_{k,j}^T d_{k,j}^2, \quad Z_{k,2}^* = \sum_{j=1}^{r_k} u_{k,j} d_{k,j}^2, \quad Z_{k,4}^* = \sum_{j=1}^{r_k} d_{k,j}^2, \tag{86}$$

and the weights due to each Z_k^* are

$$\begin{aligned} \text{First layer filters:} \quad & u_{k,1}, u_{k,2}, \ldots, u_{k,r_k} \\ \text{Second layer weights:} \quad & \{d_{k,1}^2, 0, 0, \ldots, 0\}, \{0, d_{k,2}^2, 0, \ldots, 0\}, \ldots, \{0, 0, 0, \ldots, d_{k,r_k}^2\}. \end{aligned} \tag{87}$$

To clarify, for each filter $u_{k,j}$, we have K (scalar) weights in the second layer because we apply the same filter to K different patches and the resulting K numbers (after being input to the activation function) each are multiplied by a different second layer weight. The second layer weights associated with the filter $u_{k,j}$ will be these K numbers: $\{0, \ldots, 0, d_{k,j}^2, 0 \ldots, 0\}$, where the only nonzero entry is the j'th one. Consequently, each Z_k^* matrix produces $\operatorname{rank}(Z_k^*)$ filters and $K \operatorname{rank}(Z_k^*)$ neurons. Including the weights due to Z'^*_k as well, we will have $\sum_{k=1}^K (r_k + r'_k)$ filters and $K \sum_{k=1}^K (r_k + r'_k)$ neurons in total. The optimal number of filters is upper bounded by $2(f+1)K$ and the optimal number of neurons is upper bounded by $2(f+1)K^2$.

We omit the details of plugging the weights into the convex objective to show that it becomes equivalent to the non-convex objective. The details are similar to the vector output case. □

9 Average Pooling

In this section we will consider convolutional neural networks with average pooling. We will denote the pool size by P. Let us consider a two-layer neural network where the first layer is a convolutional layer with filter size f. The convolutional layer is followed by the polynomial activation, average pooling, and a fully connected layer. We will denote the number of patches per sample by K. The output of this architecture can be expressed as

$$f(x) = \sum_{j=1}^{m} \sum_{k=1}^{K/P} \left(\frac{1}{P} \sum_{l=1}^{P} \sigma(x_{(k-1)P+l}^T u_j) \right) \alpha_{jk} \,. \tag{88}$$

We note that the number of parameters in the second layer (i.e. α_{jk}'s) is equal to $m\frac{K}{P}$. The optimization problem for this architecture can be written as

$$p^* = \min_{\{u_j\}_{j=1}^m \text{ s.t. } \|u_j\|_2=1, \forall j} \; \min_{\{\alpha_j\}_{j=1}^m, \hat{y}} \ell(\hat{y}, y) + \beta \sum_{j=1}^{m} \|\alpha_j\|_1 \quad \text{s.t.} \quad \hat{y} = \sum_{j=1}^{m} \sum_{k=1}^{K/P} \left(\frac{1}{P} \sum_{l=1}^{P} \sigma(X_{(k-1)P+l} u_j) \right) \alpha_{jk} \,, \tag{89}$$

where $\alpha_j = \begin{bmatrix} \alpha_{j1} & \dots & \alpha_{j,K/P} \end{bmatrix}^T$, $j = 1, \dots, m$. The dual of the inner minimization problem is given by

$$\max_v -\ell^*(-v) \quad \text{s.t.} \quad \left| v^T \left(\frac{1}{P} \sum_{l=1}^{P} \sigma(X_{(k-1)P+l} u_j) \right) \right| \leq \beta \,, \forall j, k \,. \tag{90}$$

Theorem 9.1 states our result for CNN with average pooling.

Theorem 9.1 (Globally optimal convex program for polynomial activation convolutional neural networks with average pooling)**.** *The solution of the convex problem in* (95) *provides a global optimal solution for the non-convex problem for the convolutional neural network with average pooling in* (89) *when the number of neurons satisfies* $m \geq m^*$ *where*

$$m^* = \frac{K}{P} \sum_{k=1}^{K/P} (\operatorname{rank}(Z_k^*) + \operatorname{rank}({Z'_k}^*)). \tag{91}$$

The optimal neural network weights are determined from the solution of the convex problem via the neural decomposition procedure for each Z_k^* *and* ${Z'_k}^*$. *The optimal number of neurons is upper bounded by* $m^* \leq 2(f+1)\frac{K^2}{P^2}$.

Proof of Theorem 9.1. We rewrite the constraints of the dual problem (90) as follows:

$$-\beta \leq \frac{1}{P} \sum_{l=1}^{P} \left(u_j^T \left(a \sum_{i=1}^{n} x_{i,(k-1)P+l} x_{i,(k-1)P+l}^T v_i \right) u_j + b v^T X_{(k-1)P+l} u_j + c v^T \bar{1} \right) \leq \beta, \quad \forall j, k \,. \tag{92}$$

S-procedure allows us to write this problem equivalently as

$$\begin{aligned} &\max -\ell^*(-v) \\ &\text{s.t.} \begin{bmatrix} \rho_{k,1} I - a\frac{1}{P} \sum_{l=1}^{P} \sum_{i=1}^{n} x_{i,(k-1)P+l} x_{i,(k-1)P+l}^T v_i & -\frac{1}{2P} b \sum_{l=1}^{P} X_{(k-1)P+l}^T v \\ -\frac{1}{2P} b \sum_{l=1}^{P} v^T X_{(k-1)P+l} & \beta - c\,\bar{1}^T v - \rho_{k,1} \end{bmatrix} \succeq 0, \quad k = 1, \dots, K/P \\ &\quad\; \begin{bmatrix} \rho_{k,2} I + a\frac{1}{P} \sum_{l=1}^{P} \sum_{i=1}^{n} x_{i,(k-1)P+l} x_{i,(k-1)P+l}^T v_i & \frac{1}{2P} b \sum_{l=1}^{P} X_{(k-1)P+l}^T v \\ \frac{1}{2P} b \sum_{l=1}^{P} v^T X_{(k-1)P+l} & \beta + c\,\bar{1}^T v - \rho_{k,2} \end{bmatrix} \succeq 0, \quad k = 1, \dots, K/P \,. \end{aligned} \tag{93}$$

The Lagrangian is as follows

$$L\left(v, \{\rho_{k,1}, \rho_{k,2}, Z_k, Z_k'\}_{k=1}^{K/P}\right) =$$
$$= -\ell^*(-v) + \sum_{k=1}^{K/P} \left(\rho_{k,1}\operatorname{tr}(Z_{k,1}) + \rho_{k,2}\operatorname{tr}(Z_{k,1}')\right) - a\frac{1}{P}\sum_{k=1}^{K/P}\sum_{l=1}^{P}\sum_{i=1}^{n} v_i x_{i,(k-1)P+l}^T (Z_{k,1} - Z_{k,1}') x_{i,(k-1)P+l}$$
$$- b\frac{1}{P}\sum_{k=1}^{K/P}\sum_{l=1}^{P} v^T X_{(k-1)P+l}(Z_{k,2} - Z_{k,2}') + \sum_{k=1}^{K/P}\left((\beta - \rho_{k,1})Z_{k,4} + (\beta - \rho_{k,2})Z_{k,4}'\right) - c\sum_{k=1}^{K/P}\sum_{i=1}^{n} v_i (Z_{k,4} - Z_{k,4}'), \tag{94}$$

where Z_k, Z_k' are $(f+1)\times(f+1)$ dimensional symmetric matrices. Maximizing the Lagrangian with respect to v, $\rho_{k,1}$, $\rho_{k,2}$, $k = 1, \dots, K/P$ yields the following convex SDP:

$$\begin{aligned}
\min_{\{Z_k = Z_k^T, Z_k' = Z_k'^T\}_{k=1}^{K/P}} \quad & \ell(\hat{y}, y) + \beta \sum_{k=1}^{K/P} (Z_{k,4} + Z_{k,4}') \\
\text{s.t.} \quad & \hat{y}_i = a\frac{1}{P}\sum_{k=1}^{K/P}\sum_{l=1}^{P} x_{i,(k-1)P+l}^T (Z_{k,1} - Z_{k,1}') x_{i,(k-1)P+l} + b\frac{1}{P}\sum_{k=1}^{K/P}\sum_{l=1}^{P} x_{i,(k-1)P+l}^T (Z_{k,2} - Z_{k,2}') + \\
& + c\sum_{k=1}^{K/P}(Z_{k,4} - Z_{k,4}'), \quad i \in [n] \\
& \operatorname{tr}(Z_{k,1}) = Z_{k,4},\ \operatorname{tr}(Z_{k,1}') = Z_{k,4}', \quad k = 1, \dots, K/P \\
& Z_k \succeq 0,\ Z_k' \succeq 0, \quad k = 1, \dots, K/P\,.
\end{aligned} \tag{95}$$

We omit the details of constructing the neural network weights from the solution of the convex SDP $Z_k^*, Z_k'^*$, $k = 1, \dots, K/P$ which follows in a similar fashion as the proof of Theorem 8.1. □

We note that when we pick the pool size as $P = 1$, this is the same as not having average pooling, and the corresponding convex program is the same as (85), derived in Section 8. The other extreme for the pool size is when $P = K$ and this corresponds to what is known as *global average pooling* in which case the convex SDP simplifies to

$$\begin{aligned}
\min_{Z = Z^T, Z' = Z'^T} \quad & \ell(\hat{y}, y) + \beta(Z_4 + Z_4') \\
\text{s.t.} \quad & \hat{y}_i = a\frac{1}{K}\sum_{l=1}^{K} x_{i,l}^T (Z_1 - Z_1') x_{i,l} + b\frac{1}{K}\sum_{l=1}^{K} x_{i,l}^T (Z_2 - Z_2') + c(Z_4 - Z_4'), \quad i \in [n] \\
& \operatorname{tr}(Z_1) = Z_4,\ \operatorname{tr}(Z_1') = Z_4' \\
& Z \succeq 0,\ Z' \succeq 0.
\end{aligned} \tag{96}$$

We note that the problem (96) has only two variables Z and Z'. This should be contrasted with the convolutional architecture with no pooling (85) which has $2K$ variables.

10 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.

Solvers: We have used CVXPY [11, 1] for solving the convex SDPs. In particular, we have used the open source solver SCS (splitting conic solver) [40, 41] 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 [42]. 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.

Polynomial approximation of activation functions: To obtain the degree-2 polynomial approximation of a given activation function $\sigma(u)$ such as the ReLU activation, one way is to select the polynomial coefficients a, b, c that minimize the ℓ_2 norm objective $\|T \begin{bmatrix} a & b & c \end{bmatrix}^T - s\|_2$ with

$$T = \begin{bmatrix} t_1^2 & t_1 & 1 \\ & \vdots & \\ t_N^2 & t_N & 1 \end{bmatrix}, \qquad s = \begin{bmatrix} \sigma(t_1) \\ \vdots \\ \sigma(t_N) \end{bmatrix}, \tag{97}$$

where t_i's are linearly spaced in $[L, U]$. The lower and upper limits L and U specify the range in which we would like to approximate the given activation function. For instance, when $L = -5$, $U = 5$, $N = 1000$ and $\sigma(u)$ is the ReLU activation, the optimal polynomial coefficients are $a = 0.09, b = 0.5, c = 0.47$. When we change the approximation range to a slightly narrower one with $L = -4, U = 4$, the coefficients then become $a = 0.12, b = 0.5, c = 0.38$. Note that the training data can be normalized appropriately to confine the range of the input to the neurons and control the approximation error.

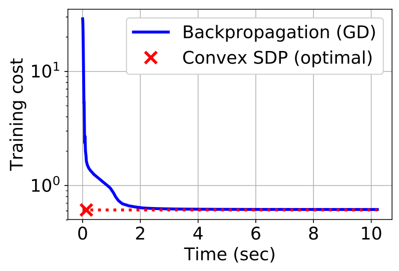
10.1 Results for Verifying the Theoretical Formulations

The first set of numerical results in Figure 7 is for verifying that the derived convex problems have the same optimal value as their non-convex counterparts. The plots in Figure 7 show the non-convex cost against time when 1) the non-convex problem is solved in PyTorch and 2) the corresponding convex problem (see Table 1) is solved using CVXPY. The number of neurons for the non-convex models in all of the plots in Figure 7 is set to the optimal number of neurons m^* found by the convex problem.

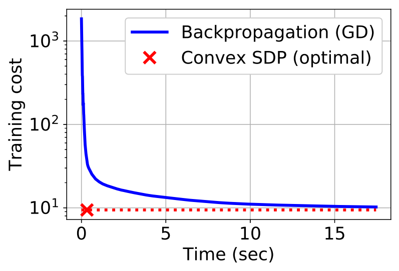
Figure 7 demonstrates that solving the convex SDP takes less time than solving the associated non-convex problem using backpropagation for all of the neural network architectures. Figure 7 also shows that the training of the non-convex models via the backpropagation algorithm does not always yield the global optimal but instead may converge to local minima. In addition, we note that the plots do not reflect the time it takes to tune the learning rate for the non-convex models, which was performed offline.

10.2 Experiments on UCI datasets

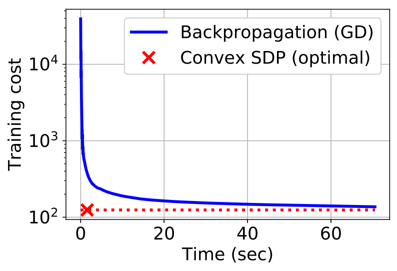
We now show how the derived convex programs perform in the context of classification datasets. The datasets used in this subsection are from the UCI machine learning repository [13]. The plots in Figure 8 show the training and test set costs and classification accuracies for *binary* classification datasets and the plots in Figure 9 are for *multiclass* classification datasets. The convex program



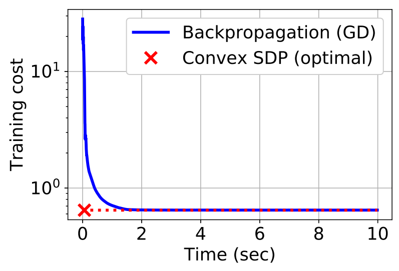
(a) Quad act $(10, 20, 9)$



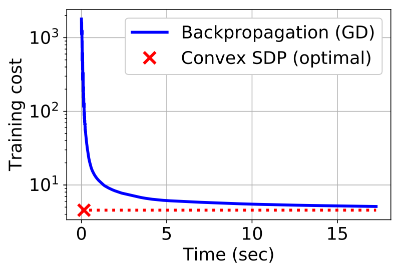
(b) Quad act $(100, 20, 20)$



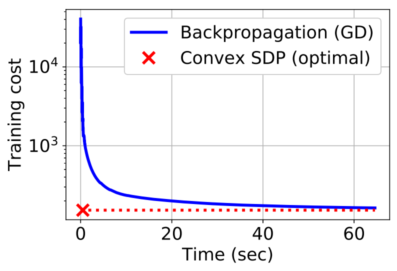
(c) Quad act $(500, 20, 20)$



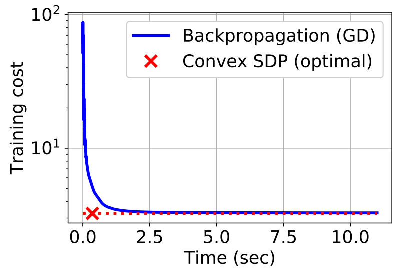
(d) Poly act $(10, 20, 10)$



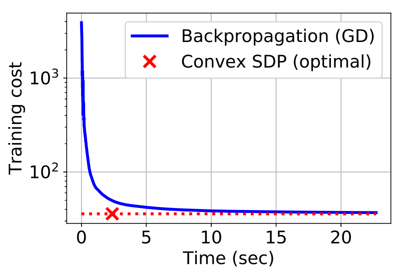
(e) Poly act $(100, 20, 23)$



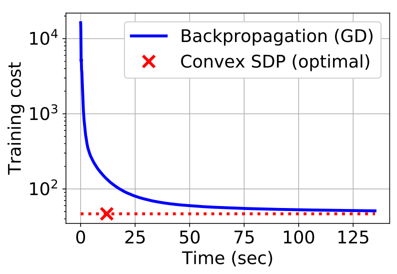
(f) Poly act $(500, 20, 35)$



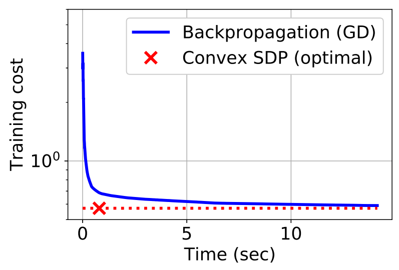
(g) Vect out $C = 3$, $(10, 20, 61)$



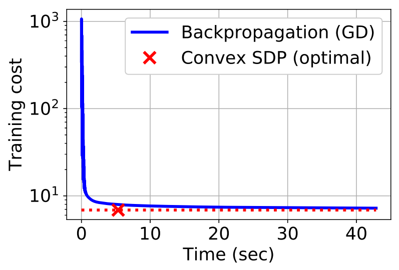
(h) Vect out $C = 3$, $(100, 20, 86)$



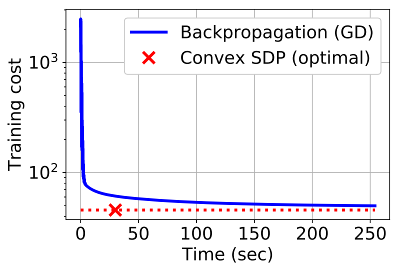
(i) Vect out $C = 3$, $(500, 20, 75)$



(j) Convol $f = 3$, $(10, 20, 11)$

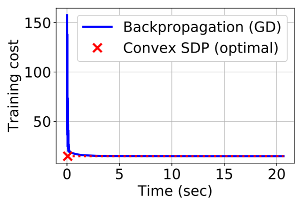


(k) Convol $f = 3$, $(100, 20, 72)$

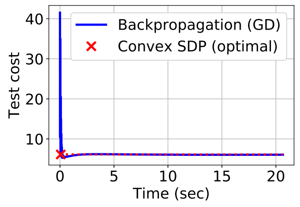


(l) Convol $f = 3$, $(500, 20, 100)$

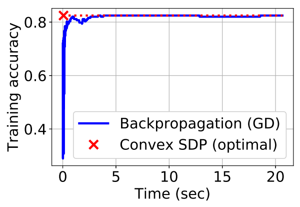
Figure 7: The numbers in the sub-captions refer to the parameters (n, d, m^*). These figures show the training cost against time for backpropagation (blue solid curves) and the convex problem (red cross shows timing of the convex solver) for the following problems: a,b,c: Quadratic activation scalar output, d,e,f: Polynomial activation scalar output, g,h,i: Polynomial activation vector output, j,k,l: Polynomial activation convolutional. The data is artificially generated with 5 planted neurons and the data matrix is the element-wise 4'th power of an i.i.d. Gaussian matrix. The regularization coefficient is $\beta = 0.1$ in all of the experiments. The polynomial coefficients for the architectures with polynomial activation are $a = 0.09$, $b = 0.5$, $c = 0.47$ (i.e. the ReLU approximation coefficients).



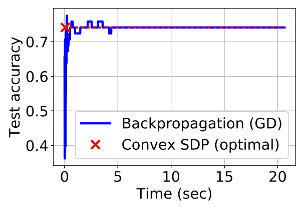
(a) DS1, training cost



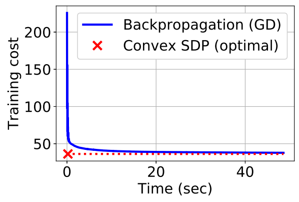
(b) DS1, test cost



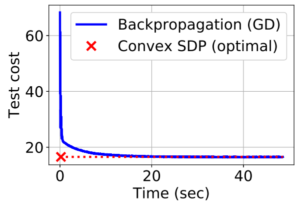
(c) DS1, training accuracy



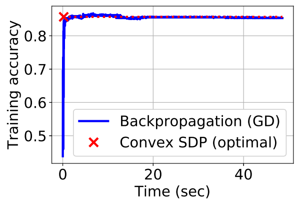
(d) DS1, test accuracy



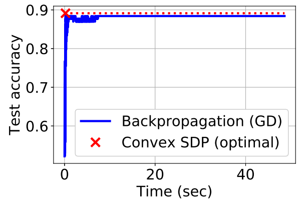
(e) DS2, training cost



(f) DS2, test cost



(g) DS2, training accuracy



(h) DS2, test accuracy

Figure 8: Results on UCI binary classification datasets. DS1: dataset 1 is the breast cancer dataset ($n = 228, d = 9$), DS2: dataset 2 is the credit approval dataset ($n = 552, d = 15$). Polynomial activation with $a = 0.09$, $b = 0.5$, $c = 0.47$ is used. Number of neurons that the convex program found is 16 and 18 for DS1 and DS2, respectively. The regularization coefficient is $\beta = 0.01$ and $\beta = 10$ for DS1 and DS2, respectively.

used for solving the binary classification problem is the scalar output polynomial activation problem given in (21) and for the multiclass problem it is the vector output version given in (72).

We note that the training cost plots of Figure 8 and 9 are consistent with the theoretical results. The accuracy plots show that the convex programs achieve the same final accuracy of the non-convex models or higher accuracies in shorter amounts of time.

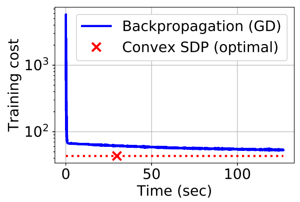
Table 2 shows the classification accuracies of various fully connected neural network architectures on binary classification UCI datasets. For each dataset, the training and validation partitions are as pre-processed in [17]. The training and validation partitions are used to select the best hyperparameters. The hyperparameter search for the non-convex models includes searching for the best regularization coefficient β and learning rate. Gradient descent has been used to optimize the non-convex models and the number of epochs is 1000. After determining the best hyperparameters, we compute the 4-fold cross validation accuracy and report it in this table. The partitions for the 4-fold cross validation are also the same as those pre-processed by [17]. Furthermore, for the results shown in Table 2, the number of neurons for all the non-convex models is set to $2(d + 1)$, which is the maximum number of neurons that the polynomial activation convex SDP could output (see Theorem 3.1). Table 2 shows that the convex SDP achieves better or similar accuracy values compared to the non-convex models on most of the datasets.

10.3 Comparison with ReLU Networks

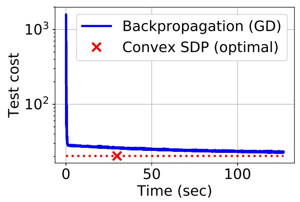
We compare the classification accuracies for polynomial activation and ReLU activation in Figure 10 on three different binary classification UCI datasets. The regularization coefficient has been picked separately for polynomial activation and ReLU activation networks to maximize the accuracy. Figure 10 demonstrates that the convex SDP shows competitive accuracy performance and faster run times compared to ReLU activation networks.

dataset	n	d	R-Q	P-C	Cvx 111	Cvx r-app	Cvx s-app	max(Cvx)
acute-inflammation	120	6	100.0	100.0	100.0	100.0	100.0	100.0
acute-nephritis	120	6	100.0	100.0	100.0	100.0	100.0	100.0
breast-cancer	286	9	69.37	**73.59**	73.59	72.89	72.89	**73.59**
breast-cancer-wisc-diag	569	30	79.05	95.95	95.42	96.13	96.13	**96.13**
breast-cancer-wisc-prog	198	33	**80.1**	79.08	77.55	79.59	77.55	79.59
congressional-voting	435	16	61.47	61.47	61.47	61.7	61.47	**61.7**
conn-bench-sonar-mines-rocks	208	60	79.81	79.33	81.73	79.81	75.0	**81.73**
cylinder-bands	512	35	75.59	75.2	75.59	76.95	76.37	**76.95**
echocardiogram	131	10	84.09	83.33	85.61	85.61	84.09	**85.61**
fertility	100	9	**89.0**	86.0	88.0	88.0	88.0	88.0
haberman-survival	306	3	73.03	**73.68**	71.38	73.36	72.04	73.36
heart-hungarian	294	12	83.56	83.9	83.22	84.25	84.25	**84.25**
hepatitis	155	19	80.13	**89.1**	80.13	77.56	80.13	80.13
horse-colic	368	25	81.67	81.0	81.67	80.33	84.0	**84.0**
ilpd-indian-liver	583	9	**73.63**	72.95	71.92	73.12	72.95	73.12
molec-biol-promoter	106	57	77.88	78.85	72.12	82.69	78.85	**82.69**
monks-1	556	6	**84.68**	70.16	75.81	81.45	81.45	81.45
parkinsons	195	22	90.82	87.24	88.27	86.73	91.33	**91.33**
pittsburg-bridges-T-OR-D	102	7	**88.0**	88.0	82.0	87.0	87.0	87.0
planning	182	12	71.67	71.11	71.67	71.11	71.11	71.67
spect	265	22	60.0	**75.0**	71.25	60.0	58.75	71.25
spectf	267	44	72.5	75.0	58.75	60.0	77.5	**77.5**
statlog-heart	270	13	82.46	**85.07**	81.72	83.58	83.21	83.58
vertebral-column-2clases	310	6	87.01	85.71	82.79	87.01	84.42	87.01

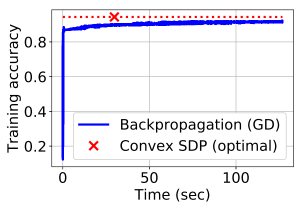
Table 2: Classification accuracies on binary classification UCI datasets. The first 3 columns are the dataset name, the number of samples n in the dataset, and the dimension d of the samples. The remaining columns show the classification accuracies (percentage) for various models. The highest accuracies for each dataset are shown in bold font. Abbreviations used in the table are as follows: R-Q: Non-convex two-layer neural network model with ReLU activation and quadratic regularization (i.e. weight decay), P-C: Non-convex two-layer neural network model with polynomial activation with coefficients $a = 0.09, b = 0.5, c = 0.47$ and normalized first layer weights and ℓ_1 norm regularization on the second layer weights, Cvx 111: Convex SDP with polynomial coefficients $a = 1, b = 1, c = 1$, Cvx r-app: Convex SDP with polynomial coefficients $a = 0.09, b = 0.5, c = 0.47$ (approximating ReLU activation), Cvx s-app: Convex SDP with polynomial coefficients $a = 0.1, b = 0.5, c = 0.24$ (approximating swish activation), max(Cvx): The highest accuracy among the convex SDPs.



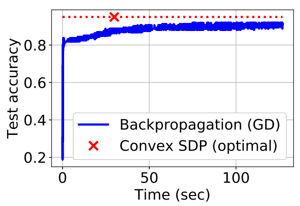
(a) DS3, training cost



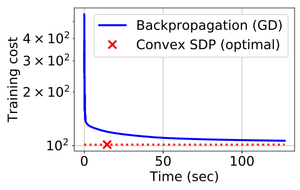
(b) DS3, test cost



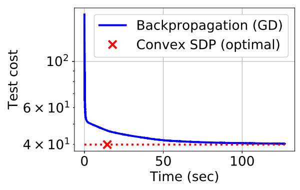
(c) DS3, training accuracy



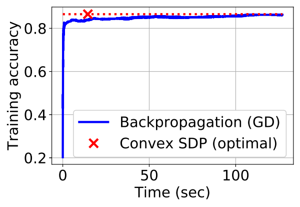
(d) DS3, test accuracy



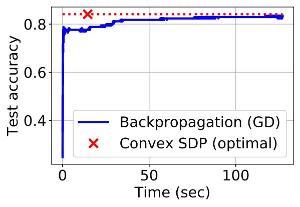
(e) DS4, training cost



(f) DS4, test cost



(g) DS4, training accuracy



(h) DS4, test accuracy

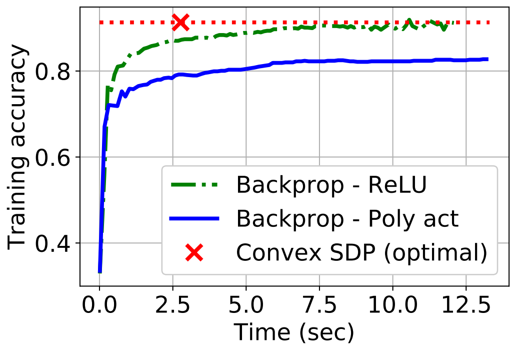
Figure 9: Results on UCI *multiclass* classification datasets. DS3: dataset 3 is the annealing dataset ($n = 638, d = 31, C = 5$), DS4: dataset 4 is the statlog vehicle dataset ($n = 676, d = 18, C = 4$). Polynomial activation with $a = 0.09$, $b = 0.5$, $c = 0.47$ is used. Number of neurons that the convex program found is 172 and 107 for DS3 and DS4, respectively. The regularization coefficient is $\beta = 1$ both for DS3 and DS4.

10.4 CNN Experiments

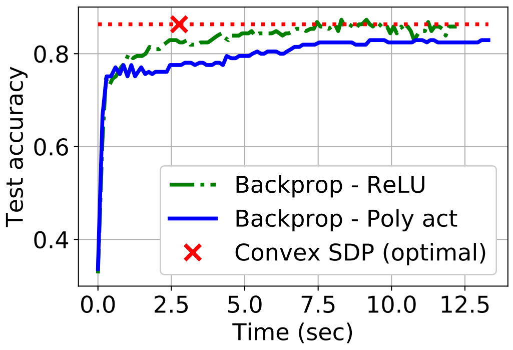
Figure 11 shows the binary classification accuracy performance of the CNN architecture with global average pooling on MNIST [32], Fashion MNIST [52], and Cifar-10 [29] datasets. Figure 11 compares the non-convex tractable problem, the corresponding convex formulation, and the non-convex weight decay formulation. By the weight decay formulation, we mean quadratic regularization on both the first layer filters and the second layer weights. We observe that the accuracy of the convex SDP is slightly better or the same as SGD while the run time for the convex SDP solution is consistently shorter than the time it takes for SGD to converge.

10.5 Regularization Parameter

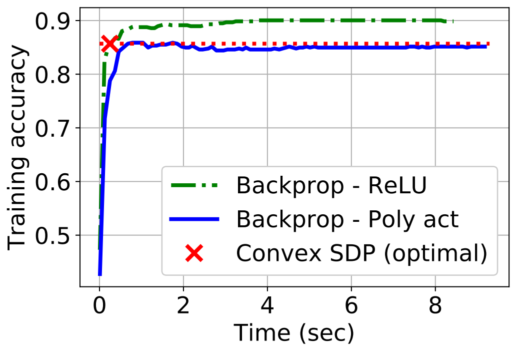
Figure 12 shows how the accuracy changes as a function of the regularization coefficient β for the convex problem for two-layer polynomial activation networks. Figure 12 highlights that the choice of the regularization coefficient is critical in the accuracy performance. In plot a, we see that the value of β that maximizes the test set accuracy is $\beta = 10$ for which the optimal number of neurons m^* is near 20. We note that for the dataset in plot a, the optimal number of neurons is upper bounded by $m^* \leq 2(d+1) = 32$. Similarly for plot b, the best choice for the regularization coefficient is $\beta = 1$ and the optimal number of neurons for $\beta = 1$ is near 40. Furthermore, we observe that a higher value for β 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 fixed 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 coefficient β.



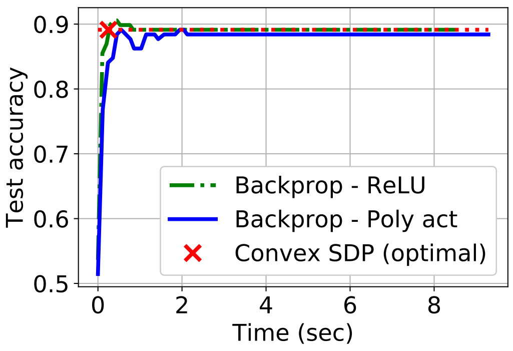
(a) DS1, training accuracy



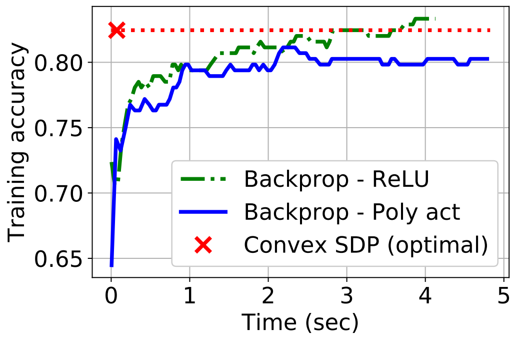
(b) DS1, test accuracy



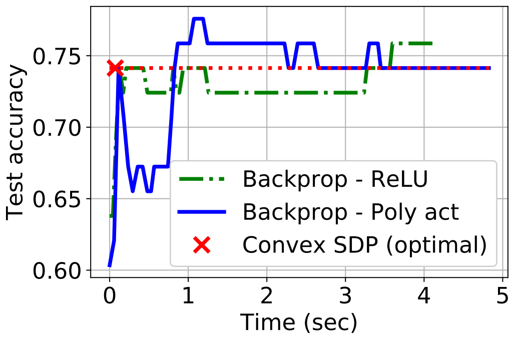
(c) DS2, training accuracy



(d) DS2, test accuracy

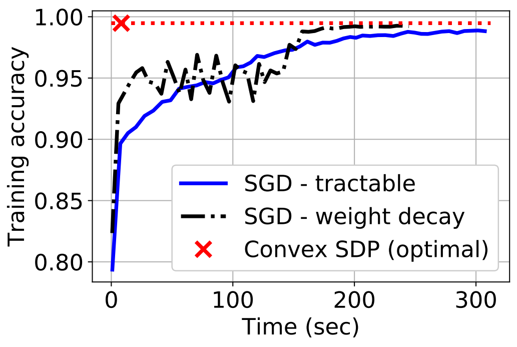


(e) DS3, training accuracy

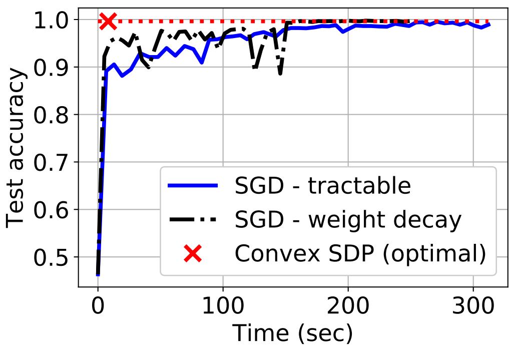


(f) DS3, test accuracy

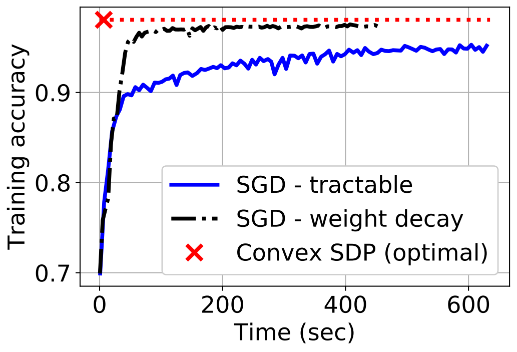
Figure 10: Comparison of classification accuracies for neural networks with ReLU activation, polynomial activation ($a = 0.09, b = 0.5, c = 0.47$), and the convex SDP. DS1: dataset 1 is the oocytes-merluccius-nucleus-4d ($n = 817, d = 41$), DS2: dataset 2 is the credit approval dataset ($n = 552, d = 15$), DS3: dataset 3 is the breast cancer dataset ($n = 228, d = 9$).



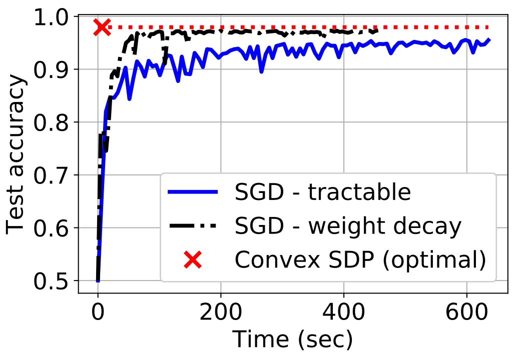
(a) MNIST, training accuracy



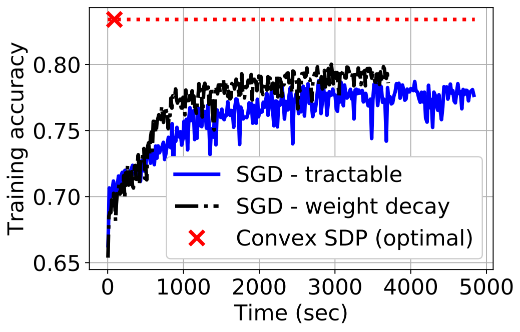
(b) MNIST, test accuracy



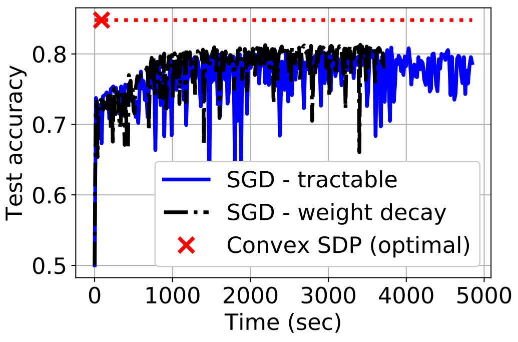
(c) Fashion-MNIST, training accuracy



(d) Fashion-MNIST, test accuracy

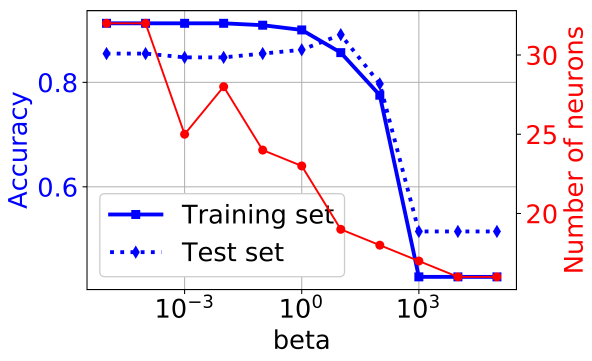


(e) Cifar, training accuracy

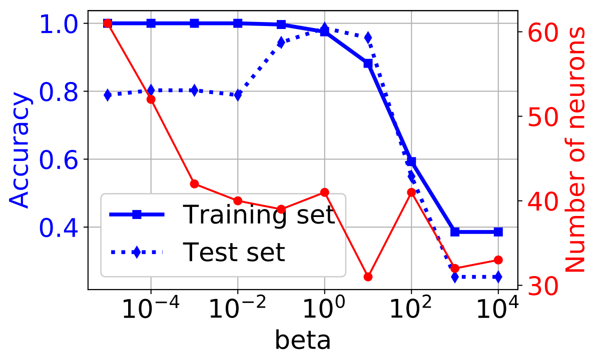


(f) Cifar, test accuracy

Figure 11: Binary classification with polynomial activation convolutional neural network with pooling and the corresponding convex SDP. Legend labels are as follows. *SGD - tractable*: The non-convex problem in (89), *SGD - weight decay*: Non-convex problem with quadratic regularization on all weights, *Convex SDP (optimal)*: The convex problem in (95). Polynomial coefficients are $a = 0.09, b = 0.5, c = 0.47$. Filter size is $f = 3$, stride is 1, and no padding is used. Batch size for SGD is 100. The regularization coefficient is $\beta = 10^{-6}$. Constrained least squares form of the convex program was used for speed (see section A.1) and the pre-computation step, not shown in the plots, takes 2 minutes. Plots a, b show the binary classification accuracy on the first two classes of the MNIST dataset where the classes are the digits 0 and 1 and there are 12600 gray-scale images of size 28×28. Plots c, d show the binary classification accuracy on the first two classes of Fashion-MNIST dataset where the classes are 'T-shirt/top' and 'Trouser' and there are 12000 gray-scale images of size 28×28. For plots e, f, the dataset is Cifar-2 (the first two classes of the Cifar-10 dataset) and has 10000 RGB images each of size $32 \times 32 \times 3$.



(a) credit approval ($n = 552, d = 15$)



(b) ionosphere ($n = 280, d = 33$)

Figure 12: Accuracy (left vertical axis) and optimal number of neurons (right vertical axis) against the regularization coefficient β on binary classification datasets. These results have been obtained using the convex program in (21).

10.6 Other Losses

We have so far evaluated the performance of the derived convex programs for squared loss, i.e. $\ell(\hat{y}, y) = \|\hat{y} - y\|_2^2$. We reiterate that the derived convex programs are general in the sense the formulations hold for any convex loss function ℓ. To verify this numerically, we now present results for additional loss functions such as Huber loss and ℓ_1 norm loss in Figure 13. More concretely, Huber loss is defined as $\ell(\hat{y}, y) = \sum_{i=1}^{n} \text{Huber}(\hat{y}_i - y_i)$ where $\text{Huber}(x) = 2|x| - 1$ for $|x| > 1$ and $\text{Huber}(x) = x^2$ for $|x| \leq 1$. The ℓ_1 norm loss is $\ell(\hat{y}, y) = \|\hat{y} - y\|_1$. We observe that in the case of ℓ_1 norm loss, backpropagation takes longer to converge.

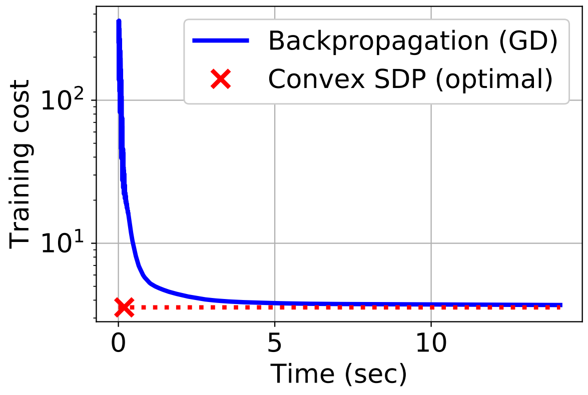
10.7 The Effect of Polynomial Coefficients

The plots in Figure 14 show the classification accuracy against the polynomial coefficients a, b, c for the polynomial activation convex problem. In each plot, we vary one of the coefficients and fix the other two coefficients as 1. We observe that the coefficient of the quadratic term a plays the most important role in the accuracy performance. The accuracy is not affected by the choice of the coefficient c.

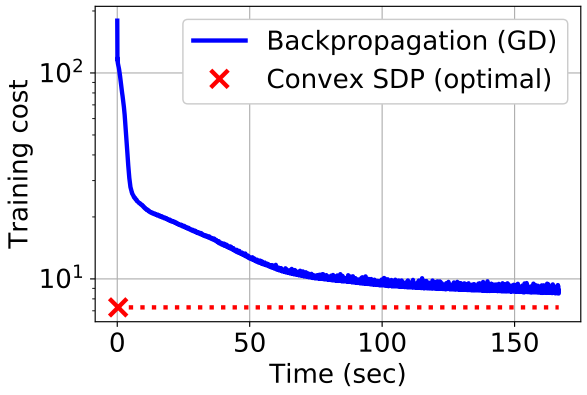
11 Discussion

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 quadratic activation, the standard non-convex problem requires optimizing m neurons (i.e. a d-dimensional first layer weight and a 1-dimensional second layer weight per neuron). The convex program for this neural network finds the optimal network parameters in the lifted space $\mathbb{S}^{d \times d}$. For

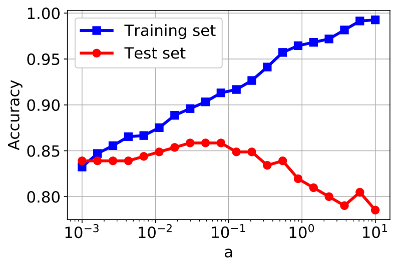


(a) Huber loss

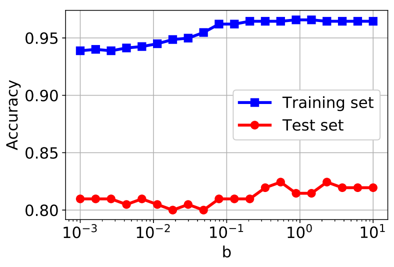


(b) ℓ_1 norm loss

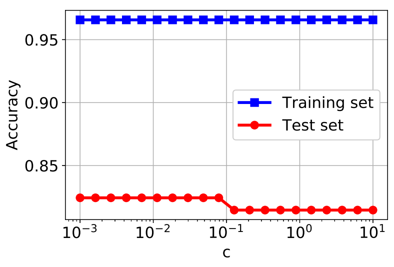
Figure 13: Verifying the theoretical results for other convex loss functions: Huber and ℓ_1 norm loss. An artificially generated dataset with dimensions $n = 100, d = 20$ is used. The regularization coefficient is $\beta = 0.1$. The number of neurons m^* is found to be 7 and 9 for plots a and b, respectively.



(a) $b = c = 1$



(b) $a = c = 1$



(c) $a = b = 1$

Figure 14: Training and test set classification accuracies against polynomial coefficients a, b, c. The regularization coefficient is $\beta = 0.1$ and the dataset is oocytes-merluccius-nucleus-4d.

polynomial activations, convex optimization takes place for Z and Z' in $\mathbb{S}^{(d+1)\times(d+1)}$. We note that the dimensions of the convex programs are polynomial with respect to all problem dimensions. In contrast, the convex program of [43] has $2dP$ variables where P 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 specifically 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 [4] 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 efficiently solved and eliminate much of the hyperparameter tuning involved in standard neural network training.

Acknowledgements

This work was partially supported by the National Science Foundation under grants IIS-1838179, ECCS-2037304, Facebook Research, Adobe Research and Stanford SystemX Alliance.

References

[1] Akshay Agrawal, Robin Verschueren, Steven Diamond, and Stephen Boyd. A rewriting system for convex optimization problems. *Journal of Control and Decision*, 5(1):42–60, 2018.

[2] Zeyuan Allen-Zhu and Yuanzhi Li. Backward feature correction: How deep learning performs deep learning. *arXiv preprint arXiv:2001.04413*, 2020.

[3] Raman Arora, Amitabh Basu, Poorya Mianjy, and Anirbit Mukherjee. Understanding deep neural networks with rectified linear units. In *6th International Conference on Learning Representations, ICLR 2018*, 2018.

[4] Eugene Belilovsky, Michael Eickenberg, and Edouard Oyallon. Greedy layerwise learning can scale to imagenet. *CoRR*, abs/1812.11446, 2018.

[5] Daniel Bienstock, Gonzalo Muñoz, and Sebastian Pokutta. Principled deep neural network training through linear programming, 2018.

[6] Mathieu Blondel, Akinori Fujino, and Naonori Ueda. Convex factorization machines. *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD)*, 2015.

[7] Mathieu Blondel, Vlad Niculae, Takuma Otsuka, and Naonori Ueda. Multi-output polynomial networks and factorization machines. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 3351–3361, Red Hook, NY, USA, 2017. Curran Associates Inc.

[8] Stephen Boyd and Lieven Vandenberghe. *Convex optimization.* Cambridge university press, 2004.

[9] Samuel Burer. Copositive programming. In *Handbook on semidefinite, conic and polynomial optimization*, pages 201–218. Springer, 2012.

[10] Lenaic Chizat, Edouard Oyallon, and Francis Bach. On lazy training in differentiable programming. In *Advances in Neural Information Processing Systems*, pages 2937–2947, 2019.

[11] Steven Diamond and Stephen Boyd. CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 17(83):1–5, 2016.

[12] Simon Du and Jason Lee. On the power of over-parametrization in neural networks with quadratic activation. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1329–1338, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

[13] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.

[14] Tolga Ergen and Mert Pilanci. Convex geometry of two-layer relu networks: Implicit autoencoding and interpretable models. In *International Conference on Artificial Intelligence and Statistics*, pages 4024–4033. PMLR, 2020.

[15] Tolga Ergen and Mert Pilanci. Implicit convex regularizers of cnn architectures: Convex optimization of two- and three-layer networks in polynomial time. *arXiv preprint arXiv:2006.14798*, 2020.

[16] Tolga Ergen and Mert Pilanci. Revealing the structure of deep neural networks via convex duality. *arXiv preprint arXiv:2002.09773*, 2020.

[17] Manuel Fernández-Delgado, Eva Cernadas, Senén Barro, and Dinani Amorim. Do we need hundreds of classifiers to solve real world classification problems? *Journal of Machine Learning Research*, 15(90):3133–3181, 2014.

[18] Matthew Fickus, Dustin G. Mixon, Aaron A. Nelson, and Yang Wang. Phase retrieval from very few measurements. *arXiv preprint arXiv:1307.7176*, 2013.

[19] David Gamarnik, Eren C. Kızıldağ, and Ilias Zadik. Stationary points of shallow neural networks with quadratic activation function. *arXiv preprint arXiv:1912.01599*, 2020.

[20] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *International Conference on Machine Learning*, pages 201–210, 2016.

[21] Surbhi Goel, Varun Kanade, Adam Klivans, and Justin Thaler. Reliably learning the relu in polynomial time. In Satyen Kale and Ohad Shamir, editors, *Proceedings of the 2017 Conference on Learning Theory*, volume 65 of *Proceedings of Machine Learning Research*, pages 1004–1042, Amsterdam, Netherlands, 07–10 Jul 2017. PMLR.

[22] M. X. Goemans and D. P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM*, 42:1115–1145, 1995.

[23] Johan Håstad. Some optimal inapproximability results. *Journal of the ACM (JACM)*, 48(4):798–859, 2001.

[24] Ehsan Hesamifard, Hassan Takabi, and Mehdi Ghasemi. Cryptodl: towards deep learning over encrypted data. In *Annual Computer Security Applications Conference (ACSAC 2016), Los Angeles, California, USA*, volume 11, 2016.

[25] Yong Hsia, Gang-Xuan Lin, Ruey-Lin Sheu, et al. A revisit to quadratic programming with one inequality quadratic constraint via matrix pencil. *PACIFIC JOURNAL OF OPTIMIZATION*, 10(3):461–481, 2014.

[26] Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In *Advances in neural information processing systems*, pages 8571–8580, 2018.

[27] Subhash Khot, Guy Kindler, Elchanan Mossel, and Ryan O'Donnell. Optimal inapproximability results for max-cut and other 2-variable csps? *SIAM Journal on Computing*, 37(1):319–357, 2007.

[28] Diederik P. Kingma and Jimmy Ba. 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.

[29] Alex Krizhevsky. Learning multiple layers of features from tiny images, 2009.

[30] Jonathan Lacotte and Mert Pilanci. All local minima are global for two-layer relu neural networks: The hidden convex optimization landscape. *arXiv preprint arXiv:2006.05900*, 2020.

[31] Monique Laurent and Svatopluk Poljak. On a positive semidefinite relaxation of the cut polytope. *Linear Algebra and its Applications*, 223(224):439–461, 1995.

[32] Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010.

[33] Johannes Lederer. No spurious local minima: on the optimization landscapes of wide and deep neural networks, 2020.

[34] Roi Livni, Shai Shalev-Shwartz, and Ohad Shamir. On the computational efficiency of training neural networks. *NIPS'14*, page 855–863, 2014.

[35] Stefano Sarao Mannelli, Eric Vanden-Eijnden, and Lenka Zdeborová. Optimization and generalization of shallow neural networks with quadratic activation functions. *arXiv preprint arXiv:2006.15459*, 2020.

[36] Pratyush Mishra, Ryan Lehmkuhl, Akshayaram Srinivasan, Wenting Zheng, and Raluca Ada Popa. Delphi: A cryptographic inference system for neural networks. *PPMLP'20*, page 27–30, 2020.

[37] Gregory L Naber. *The geometry of Minkowski spacetime: An introduction to the mathematics of the special theory of relativity*, volume 92. Springer Science & Business Media, 2012.

[38] Yuri Nesterov, Henry Wolkowicz, and Yinyu Ye. Semidefinite programming relaxations of nonconvex quadratic optimization. In *Handbook of semidefinite programming*, pages 361–419. Springer, 2000.

[39] Srinath Obla, Xinghan Gong, Asma Aloufi, Peizhao Hu, and Daniel Takabi. Effective activation functions for homomorphic evaluation of deep neural networks. *IEEE Access*, 8:153098–153112, 2020.

[40] B. O'Donoghue, E. Chu, N. Parikh, and S. Boyd. Conic optimization via operator splitting and homogeneous self-dual embedding. *Journal of Optimization Theory and Applications*, 169(3):1042–1068, June 2016.

[41] B. O'Donoghue, E. Chu, N. Parikh, and S. Boyd. SCS: Splitting conic solver, version 2.1.2. `https://github.com/cvxgrp/scs`, November 2019.

[42] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems 32*, pages 8024–8035, 2019.

[43] Mert Pilanci and Tolga Ergen. 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.

[44] Imre Pólik and Tamás Terlaky. A survey of the s-lemma. *SIAM Review*, 49(3):371–418, 2007.

[45] Prajit Ramachandran, Barret Zoph, and Quoc Le. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2018.

[46] Arda Sahiner, Tolga Ergen, John Pauly, and Mert Pilanci. 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.

[47] Arda Sahiner, Morteza Mardani, Batu Ozturkler, Mert Pilanci, and John Pauly. Convex regularization behind neural reconstruction. *arXiv preprint arXiv:2012.05169*, 2020.

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

[49] Mahdi Soltanolkotabi, Adel Javanmard, and Jason D. Lee. Theoretical insights into the optimization landscape of over-parameterized shallow neural networks. *IEEE Trans. Inf. Theor.*, 65(2):742–769, February 2019.

[50] Luca Trevisan, Gregory B Sorkin, Madhu Sudan, and David P Williamson. Gadgets, approximation, and linear programming. *SIAM Journal on Computing*, 29(6):2074–2097, 2000.

[51] Henry Wolkowicz, Romesh Saigal, and Lieven Vandenberghe. *Handbook of semidefinite programming: theory, algorithms, and applications*, volume 27. Springer Science & Business Media, 2012.

[52] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.

A Additional Discussion

A.1 Constrained Least Squares Form for the Squared Loss

Let us consider the polynomial activation scalar output case. In the case of squared loss $\ell(\hat{y}, y) = \|\hat{y} - y\|_2^2$, the convex program takes the following form:

$$\begin{aligned}\min_{Z=Z^T, Z'=Z'^T} \quad & \sum_{i=1}^{n} \left(a x_i^T (Z_1 - Z_1') x_i + b x_i^T (Z_2 - Z_2') + c(Z_4 - Z_4') - y_i \right)^2 + \beta(Z_4 + Z_4') \\ \text{s.t.} \quad & \operatorname{tr}(Z_1) = Z_4, \ \operatorname{tr}(Z_1') = Z_4' \\ & Z \succeq 0, \ Z' \succeq 0 \,. \end{aligned} \tag{98}$$

Noting that $a x_i^T (Z_1 - Z_1') x_i = \operatorname{vec}(x_i x_i^T)^T \operatorname{vec}(Z_1 - Z_1')$, we can write the squared loss term as

$$\sum_{i=1}^{n} \left(\begin{bmatrix} a \operatorname{vec}(x_i x_i^T)^T & b x_i^T & c \end{bmatrix} \begin{bmatrix} \operatorname{vec}(Z_1 - Z_1') \\ Z_2 - Z_2' \\ Z_4 - Z_4' \end{bmatrix} - y_i \right)^2 =$$

$$= \left\| \begin{bmatrix} a \operatorname{vec}(x_1 x_1^T)^T & b x_1^T & c \\ \vdots & & \\ a \operatorname{vec}(x_n x_n^T)^T & b x_n^T & c \end{bmatrix} \begin{bmatrix} \operatorname{vec}(Z_1 - Z_1') \\ Z_2 - Z_2' \\ Z_4 - Z_4' \end{bmatrix} - y \right\|_2^2 = \|X_V z - y\|_2^2$$

where we have defined $X_V \in \mathbb{R}^{n \times (d^2+d+1)}$ and $z \in \mathbb{R}^{(d^2+d+1)}$. The squared loss term is equal to $z^T X_V^T X_V z - 2 y^T X_V z + \|y\|_2^2$.

If we pre-compute $X_V^T X_V \in \mathbb{R}^{(d^2+d+1) \times (d^2+d+1)}$ and $X_V^T y \in \mathbb{R}^{(d^2+d+1)}$, then the objective no longer has dependence on the number of samples n. We note that the pre-computation of $X_V^T X_V$ and $X_V^T y$ is useful when one is performing hyperparameter tuning for the regularization coefficient β.

B Proofs

Proof of Lemma 2.1. We will denote the set in (17) as $\mathcal{S}_1$ and the set in (18) as $\mathcal{S}_2$ to simplify the notation. We will prove $\mathcal{S}_1 = \mathcal{S}_2$ by showing $\mathcal{S}_1 \subseteq \mathcal{S}_2$ and $\mathcal{S}_2 \subseteq \mathcal{S}_1$.
We first show $\mathcal{S}_1 \subseteq \mathcal{S}_2$. Let us take a point $S \in \mathcal{S}_1$. This implies that S is a matrix of the form

$$t \sum_{j=1}^{m} \begin{bmatrix} u_j \\ 1 \end{bmatrix} \begin{bmatrix} u_j \\ 1 \end{bmatrix}^T \alpha_j = t \sum_{j=1}^{m} \begin{bmatrix} u_j u_j^T \alpha_j & u_j \alpha_j \\ u_j^T \alpha_j & \alpha_j \end{bmatrix} = \begin{bmatrix} t \sum_{j=1}^m u_j u_j^T \alpha_j & t \sum_{j=1}^m u_j \alpha_j \\ t \sum_{j=1}^m u_j^T \alpha_j & t \sum_{j=1}^m \alpha_j \end{bmatrix} \tag{99}$$

with $\sum_j \alpha_j \le 1$ and $\|u_j\|_2 = 1$ for all j. We note that $\operatorname{tr}(t \sum_{j=1}^m u_j u_j^T \alpha_j) = t \sum_{j=1}^m \operatorname{tr}(u_j u_j^T \alpha_j) = t \sum_{j=1}^m \operatorname{tr}(u_j^T u_j) \alpha_j = t \sum_{j=1}^m \alpha_j \le t$. This shows that S satisfies the equality condition in the definition (18). 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 α_j's and t are nonnegative, it follows that S is PSD. This proves that $S \in \mathcal{S}_2$.
We next show $\mathcal{S}_2 \subseteq \mathcal{S}_1$. Let us take a point $S \in \mathcal{S}_2$. This implies that S is PSD and $\operatorname{tr}(S_1) = S_4 = t_0 \le t$. We show in Section 4 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

$$S = \frac{t_0}{\sum_{j=1}^m d_j^2} \begin{bmatrix} \sum_{j=1}^m u_j u_j^T d_j^2 & \sum_{j=1}^m u_j d_j^2 \\ \sum_{j=1}^m u_j^T d_j^2 & \sum_{j=1}^m d_j^2 \end{bmatrix} , \tag{100}$$

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 $\mathcal{S}_1$ when $t_0 = t$ by the definition of $\mathcal{S}_1$ given in (17). When $t_0 < t$, we still have that S is in $\mathcal{S}_1$ which can be seen by noting that $\mathcal{S}_1$ 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.

□

Proof of Lemma 6.2. Let $\alpha_1, \ldots, \alpha_m$ be any feasible point. First, note that for any $s \geq 0$ and $\alpha \in \mathbb{R}$, $\alpha \neq 0$, we have

$$\big(s\,|\alpha|\big)^p \geq s\,|\alpha|^p \,, \tag{101}$$

where equality holds if and only if $s \in \{0, 1\}$. The equality condition follows since $|\alpha| > 0$ and $s^p = s$ implies $s \in \{0, 1\}$ for $p \in (0, 1)$. Then, define $s_i := \frac{|\alpha_i|}{\sum_j |\alpha_j|}$, which satisfies $\sum_i s_i = 1$, and observe that

$$\begin{aligned}
\sum_i |\alpha_i|^p &= \sum_i \big|s_i\big(\sum_j |\alpha_j|\big)\big|^p \\
&\geq \Big(\sum_i s_i\Big)\Big(\sum_j |\alpha_j|\Big)^p \\
&= \Big(\sum_i |\alpha_i|\Big)^p \\
&\geq \Big(\sum_i \alpha_i\Big)^p \\
&= 1 \,,
\end{aligned}$$

where the first inequality holds with equality if and only if $s_i \in \{0, 1\}$, $\forall i$. Hence, in order for the equality to hold, we necessarily have $\|\alpha\|_0 \leq 1$. Since $\sum_i \alpha_i = 1$, the all-zeros vector is infeasible. This implies that $\|\alpha\|_0 = 1$. Finally, note that all feasible vectors which are 1-sparse are of the form $(1, 0, \ldots, 0)$, $(0, 1, 0, \ldots, 0), \ldots, (0, \ldots, 1)$ and achieve an objective value 1. We conclude that all feasible vectors with cardinality strictly greater than 1 are suboptimal since they achieve objective value strictly larger than 1.

□

Proof of Lemma 6.3. Let us define the set $\mathcal{A} = \{a_1, a_2, \ldots, a_d\}$ where a_i are integers. We need to show that the problem (62) finds a feasible solution u_1 if and only if there exists a subset $\mathcal{A}_S$ of the set $\mathcal{A}$ that satisfies $\sum_{a \in \mathcal{A}_S} a = z$.

We assume $n = 2d + 1$ and hence $\tilde{X}$ is $(d+1) \times d$ and $\tilde{y}$ is $(d+1)$ dimensional. Let $\tilde{X}_D \in \mathbb{R}^{d \times d}$ denote the matrix with the first d rows of $\tilde{X}$, and $\tilde{x}_{d+1}$ is the last sample in $\tilde{X}$. Let us define $\tilde{y}_i$ as

$$\tilde{y}_i = \begin{cases} (a_i/w_i)^2, & i = 1, \ldots, d \\ (2z - \sum_{j=1}^d a_j)^2, & i = d+1 \,, \end{cases} \tag{102}$$

where $w = \tilde{X}_D^{-T}\tilde{x}_{d+1} \in \mathbb{R}^d$.

Direction 1: Suppose there exists $u_1 \in \mathbb{R}^d$ such that $(\tilde{x}_i^T u_1)^2 = \tilde{y}_i$ for every $i = 1, \dots, d+1$ and $u_{1k}^2 = 1/d$ for every $k = 1, \dots, d$. Then there exists a subset $\mathcal{A}_S$ with $\sum_{a \in \mathcal{A}_S} a = z$.

Proof of direction 1: Assuming $\tilde{X}_D$ is invertible, it follows that $\tilde{X}\tilde{X}_D^{-1} = \begin{bmatrix} I_d & w \end{bmatrix}^T$ where I_d is the $d \times d$ identity matrix. Let us consider a feasible u_1. Then, $v = \tilde{X}_D u_1$ satisfies $((\tilde{X}_D^{-T}\tilde{x}_i)^T v)^2 = \tilde{y}_i$ for $i = 1, \dots, d+1$. Consequently, we have $\tilde{X}_D^{-T}\tilde{x}_i = e_i$ for $i = 1, \dots, d$, and $\tilde{X}_D^{-T}\tilde{x}_{d+1} = w$. As a result, we obtain the following relation between v and $\tilde{y}$:

$$\tilde{y}_i = \begin{cases} v_i^2, & i = 1, \dots, d \\ (w^T v)^2, & i = d+1\,. \end{cases}$$

Next, because of (102), we have

$$|v_i| = \left|\frac{a_i}{w_i}\right|,\ i = 1, \dots, d, \quad \text{and} \quad |w^T v| = \left|2z - \sum_{j=1}^{d} a_j\right|.$$

Let us define ε_i such that $v_i = \varepsilon_i a_i / w_i$ for $i = 1, \dots, d$. Note that $\varepsilon_i \in \{-1, 1\}$. Then,

$$\begin{aligned}\left|2z - \sum_{j=1}^{d} a_j\right| &= |w^T v| = \left|\sum_{i=1}^{d}\varepsilon_i a_i\right| = \left|\sum_{i:\varepsilon_i=1} a_i - \sum_{i:\varepsilon_i=-1} a_i\right| = \left|\sum_{i:\varepsilon_i=1} a_i - \sum_{i=1}^{d} a_i + \sum_{i:\varepsilon_i=1} a_i\right| \\ &= \left|2\sum_{i:\varepsilon_i=1} a_i - \sum_{i=1}^{d} a_i\right|.\end{aligned}$$

This means we either have $z = \sum_{i:\varepsilon_i=1} a_i$ or $z = -\sum_{i:\varepsilon_i=1} a_i + \sum_{i=1}^{d} a_i = \sum_{i:\varepsilon_i=-1} a_i$. This shows that the sum of the elements of $\mathcal{A}_S$ is equal to z when $\mathcal{A}_S$ is either equal to $\{a_i | \varepsilon_i = 1\}$ or $\{a_i | \varepsilon_i = -1\}$.

In proving direction 2, it is straightforward to show the existence of u_1 that satisfies the constraint $(\tilde{x}_i^T u_1)^2 = \tilde{y}_i$. To show that there is a u_1 that satisfies the constraint $u_{1k}^2 = \frac{1}{d}$, we pick $\tilde{X}$ in a certain way that we discuss now: To prove direction 2, we will need to make sure $|\tilde{X}_D^{-1} v| = \bar{1}\,\frac{1}{\sqrt{d}}$ is satisfied, i.e.,

$$\left|\sum_{j=1}^{d} (\tilde{X}_D^{-1})_{i,j}\varepsilon_j \frac{a_j}{w_j}\right| = \frac{1}{\sqrt{d}} \quad \text{for} \quad i = 1, \dots, d\,.$$

We pick $\tilde{X}_D$ to be any diagonal matrix with arbitrary -1's and $+1$'s on the diagonal and pick $\tilde{x}_{d+1} = \sqrt{d}\begin{bmatrix} a_1 & \dots & a_d \end{bmatrix}^T$. Since $w = \tilde{X}_D^{-T}\tilde{x}_{d+1}$, we will have $|w_i| = |\tilde{x}_{d+1,i}| = \sqrt{d}|a_i|$ for $i = 1, \dots, d$. This choice for $\tilde{X}_D$ and $\tilde{x}_{d+1}$ ensures that $|\tilde{X}_D^{-1} v| = \bar{1}\,\frac{1}{\sqrt{d}}$.

Direction 2: Suppose there is a subset $\mathcal{A}_S$ with $\sum_{a \in \mathcal{A}_S} a = z$. Then there exists a feasible $u_1 \in \mathbb{R}^d$.

Proof of direction 2: Define ε_i such that for a_i in $\mathcal{A}_S$, it is equal to 1, and otherwise it is equal to -1. Next,

$$\begin{aligned}\left|\sum_{i=1}^{d}\varepsilon_i a_i\right| &= \left|\sum_{i:\varepsilon_i=1}\varepsilon_i a_i + \sum_{i:\varepsilon_i=-1}\varepsilon_i a_i\right| = \left|\sum_{i:\varepsilon_i=1} a_i - \sum_{i:\varepsilon_i=-1} a_i\right| = \left|2\sum_{i:\varepsilon_i=1} a_i - \sum_{i=1}^{d} a_i\right| \\ &= \left|2z - \sum_{i=1}^{d} a_i\right|. \qquad (103)\end{aligned}$$

Let us take $v_i = \varepsilon_i a_i / w_i$ for $i = 1, \ldots, d$. Now we show that the point defined by $\tilde{X}_D^{-1} v$ is a feasible point. First, we check if $\tilde{X}_D^{-1} v$ satisfies the constraints $(\tilde{x}_i^T \tilde{X}_D^{-1} v)^2 = \tilde{y}_i$ for $i = 1, \ldots, i+1$. Note that

$$(\tilde{x}_i^T \tilde{X}_D^{-1} v)^2 = (e_i^T v)^2 = v_i^2 = \frac{a_i^2}{w_i^2} = \tilde{y}_i \quad \text{for} \quad i = 1, \ldots, d, \quad \text{and}$$

$$(\tilde{x}_{d+1}^T \tilde{X}_D^{-1} v)^2 = (w^T v)^2 = (\sum_{j=1}^{d} \varepsilon_j a_j)^2 = (2z - \sum_{j=1}^{d} a_j)^2 = \tilde{y}_{i+1}\,,$$

where the last two equalities follow from (103) and the definition in (102). This shows that the constraints $(\tilde{x}_i^T \tilde{X}_D^{-1} v)^2 = \tilde{y}_i$ for $i = 1, \ldots, d+1$ are satisfied by $\tilde{X}_D^{-1} v$.

We now check for the other constraint; i.e. does $\tilde{X}_D^{-1} v$ satisfy $|\tilde{X}_D^{-1} v| = \frac{1}{\sqrt{d}} \bar{1}$ where the absolute value is elementwise? This is true because $|(\tilde{X}_D^{-1} v)_i| = |\sum_{j=1}^{d} (\tilde{X}_D^{-1})_{i,j} \varepsilon_j \frac{a_j}{w_j}| = \frac{1}{\sqrt{d}}$ for $i = 1, \ldots, d$. The second equality follows from how we picked $\tilde{X}_D$ and $\tilde{x}_{d+1}$. □

Proof of Corollary 3.3. Let us define the quadratic functions $f_1(u) = -u^T Q u - b^T u + \beta$ and $f_2(u) = \|u\|_2^2 - 1$. We note that $f_2(u)$ is strictly convex and takes both negative and positive values. Then by Lemma 3.2, we have that the system $-u^T Q u - b^T u < -\beta$ (or $u^T Q u + b^T u > \beta$) and $\|u\|_2 = 1$ is not solvable if and only if there exists λ such that $-u^T Q u - b^T u + \beta + \lambda(\|u\|_2^2 - 1) \geq 0$, $\forall u$.

Equivalently, we have $\max_{\|u\|_2 = 1} u^T Q u + b^T u \leq \beta$ if and only if there exists λ such that

$$u^T (\lambda I - Q) u - b^T u + \beta - \lambda \geq 0, \quad \forall u. \tag{104}$$

We note that if we make the change of variable $u \leftarrow \frac{u}{c}$ with $c \neq 0$, then (104) implies

$$\frac{1}{c^2} u^T (\lambda I - Q) u - \frac{1}{c} b^T u + \beta - \lambda \geq 0, \quad \forall u, \forall c \neq 0$$

which is the same as

$$u^T (\lambda I - Q) u - c b^T u + c^2 (\beta - \lambda) \geq 0, \quad \forall u, \forall c \neq 0.$$

We express this inequality in matrix form as follows

$$\begin{bmatrix} u^T & c \end{bmatrix} \begin{bmatrix} \lambda I - Q & -\frac{1}{2} b \\ -\frac{1}{2} b^T & \beta - \lambda \end{bmatrix} \begin{bmatrix} u \\ c \end{bmatrix} \geq 0, \quad \forall u, \forall c \neq 0. \tag{105}$$

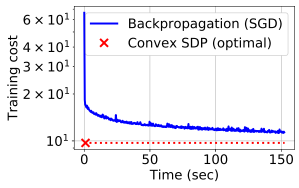
For the matrix in (105) to be PSD, we first need to show that (104) implies the inequality in (105) for $c = 0$ as well. We note that (104) implies

$$\frac{u^T}{\|u\|_2} (\lambda I - Q) \frac{u}{\|u\|_2} - b^T \frac{u}{\|u\|_2^2} + \frac{\beta - \lambda}{\|u\|_2^2} \geq 0, \quad \forall u \text{ s.t. } \|u\|_2 \neq 0.$$

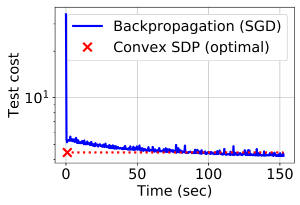
Next, taking the norm of u to infinity, we have

$$\lim_{\|u\|_2 \to \infty} \left(\frac{u^T}{\|u\|_2} (\lambda I - Q) \frac{u}{\|u\|_2} - b^T \frac{u}{\|u\|_2^2} + \frac{\beta - \lambda}{\|u\|_2^2} \right) = u_n^T (\lambda I - Q) u_n,$$

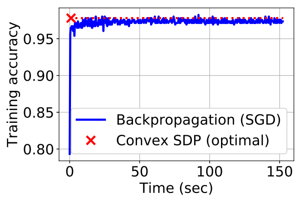
where $u_n = u / \|u\|_2$ is unit norm. We note that $u_n^T (\lambda I - Q) u_n$ is non-negative for all unit norm u_n, which is the same as the statement that it is non-negative for all u_n (not necessarily unit norm).



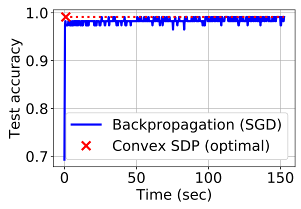
(a) DS1, training cost



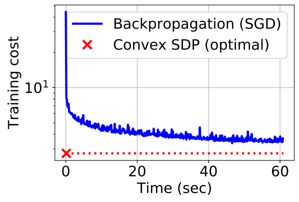
(b) DS1, test cost



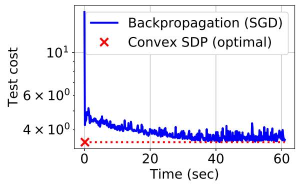
(c) DS1, training accuracy



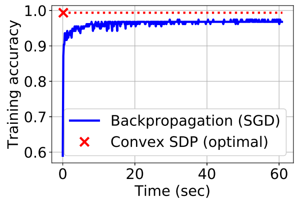
(d) DS1, test accuracy



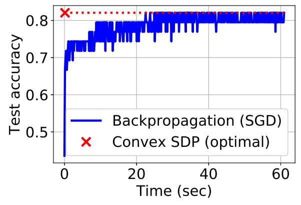
(e) DS2, training cost



(f) DS2, test cost



(g) DS2, training accuracy



(h) DS2, test accuracy

Figure 15: SGD with minibatch size 13 for two UCI datasets, DS1 is the breast-cancer-wisc-diag dataset with $n = 455, d = 30$ and DS2 is parkinsons dataset with $d = 156, d = 22$. The regularization coefficient is set to $\beta = 1$ and $\beta = 0.1$ and the number of neurons m^* is found as 34 and 27 for DS1 and DS2, respectively.

This shows that (104) implies $u^T(\lambda I - Q)u \geq 0$ for all u, which, we note, is the same as (105) with $c = 0$. Hence, because the inequality holds for all $\begin{bmatrix} u^T & c \end{bmatrix}^T$, we obtain the matrix inequality

$$\begin{bmatrix} \lambda I - Q & -\frac{1}{2}b \\ -\frac{1}{2}b^T & \beta - \lambda \end{bmatrix} \succeq 0. \tag{106}$$

The proof for the other direction of the if and only if statement is straightforward. We note that, by the definition of a PSD matrix, (106) implies that $u^T(\lambda I - Q)u - cb^T u + c^2(\beta - \lambda) \geq 0$, $\forall u, c$. Setting $c = 0$, we obtain the inequality in (104). □

C Additional Numerical Results

Figure 15 compares the costs and accuracy performance of the convex formulation with minibatch SGD.