Journal of Machine Learning Research ? (?) ? Submitted ?/21; Published ?/?

Efficient Global Optimization of Two-layer ReLU Networks: Quadratic-time Algorithms and Adversarial Training

Yatong Bai YATONG_BAI@BERKELEY.EDU
Department of Mechanical Engineering
University of California
Berkeley, CA 94720-1776, USA

Tanmay Gautam TGAUTAM23@BERKELEY.EDU
Department of Electrical Engineering and Computer Science
University of California
Berkeley, CA 94720-1776, USA

Somayeh Sojoudi SOJOUDI@BERKELEY.EDU
Department of Electrical Engineering and Computer Science and Department of Mechanical Engineering
University of California
Berkeley, CA 94720-1776, USA

Editor:

Abstract

The non-convexity of the artificial neural network (ANN) training landscape brings inherent optimization difficulties. While the traditional back-propagation stochastic gradient descent (SGD) algorithm and its variants are effective and efficient in certain cases, they can become stuck at spurious local minima and are sensitive to initializations and hyperparameters. Recent work has shown that the training of an ANN with rectified linear units (ReLU) activations can be reformulated as a convex program, bringing hope to globally optimizing interpretable ANNs. However, naively solving the convex training formulation has an exponential complexity, and even a relaxed approximation heuristic requires cubic time. In this work, we characterize the quality of this approximation and develop two efficient algorithms that train ANNs with global convergence guarantees. The first algorithm is based on the alternating direction method of multiplier (ADMM). It aims to solve both the exact convex formulation and the approximate counterpart. Linear asymptotic global convergence is achieved, and the first several iterations yield a solution that is often satisfactory (high prediction accuracy). When solving the approximate formulation, the time complexity is quadratic. The second algorithm is simpler to implement. It is based on the theory of sampled convex programs and solves unconstrained convex formulations. It converges to an approximate globally optimal classifier. The non-convexity of the ANN training landscape exacerbates when adversarial training is considered. We apply the robust convex optimization theory to convex training and develop convex formulations that train ANNs robust to adversarial input perturbations. Our analysis explicitly focuses on one-hidden-layer fully connected ANNs, but can extend to more sophisticated architectures.

Keywords: Robust Optimization, Convex Optimization, Adversarial Training, Neural Networks

1. Introduction

The artificial neural network (ANN) is one of the most powerful and popular machine learning tools. While the training formulations of training some particular ANNs are convex (Bengio et al., 2006b; Bach, 2017), optimizing a typical neural network with non-linear activation functions and a finite width requires solving non-convex optimization problems. Traditionally, training ANNs relies on stochastic gradient descent (SGD) back-propagation (Rumelhart et al., 1986). While SGD back-propagation has seen a tremendous empirical success, it is only guaranteed to converge to a local minimum when applied to the non-convex ANN training objective. While SGD back-propagation can converge to a global optimizer for one-hidden-layer ReLU-activated networks when the considered network is wide enough (Lacotte and Pilanci, 2020; Du et al., 2019) or when the inputs follow a Gaussian distribution (Brutzkus and Globerson, 2017), spurious local minima can exist in general applications. Moreover, the non-convexity of the training landscape and the structure of the back-propagation algorithm causes the issues listed below:

- **Poor interpretability:** With back-propagation, it is hard to monitor the training status. For example, when the progress slows down, we may or may not be close to a local minimum. Even if the algorithm arrives at a local optimum, this optimum may be spurious.
- **High sensitivity to hyperparameters:** Back-propagation SGD has several important hyperparameters to tune, including the number of epochs, batch size, and step size. Every parameter is crucial to the performance, but selecting the parameters can be difficult. Back-propagation is also sensitive to the initialization.
- **Vanishing / exploding gradients:** The gradient at shallower layers depends on the weights at deeper layers for back-propagation algorithms. The gradient at shallower layers can be tiny (or huge) if the deeper layer weights are tiny (or huge).

While more advanced back-propagation variants such as Adam (cite) may alleviate the above issues, avoiding them entirely can be cumbersome. These problems make ANNs more difficult to harness than many other machine learning tools that are inherently convex. Convex programs possess the desirable property that all local minima are global. To solve the issue of getting stuck at spurious local minima when training ANNs, the existing works have considered convexifying the neural network training problem (Bengio et al., 2006c; Bach, 2017; Arora et al., 2018). More recently, (Pilanci and Ergen, 2020) proposed a convex optimization formulation with the same global minimum as the non-convex cost function of a one-hidden-layer fully-connected ReLU neural network, enabling efficient global optimization. While the explicit focus is on the squared loss, their analysis extends to arbitrary convex loss functions. The favorable properties of convex optimization make convex training immune to the deficiencies of back-propagation discussed earlier. This "convex training" approach also extends to more complex ANNs such as convolutional neural networks (CNNs) (Ergen and Pilanci, 2021a), deeper networks (Ergen and Pilanci, 2021b), and vector-output networks (Sahiner et al., 2021). This work starts with one-hidden-layer ANNs for simplicity and shows that extending to more complex ANNs is possible. One-hidden-layer networks are the

simplest ANN instances possessing the vast representation power of neural networks (Hornik, 1991), and their theoretical analysis helps with understanding more complex networks (Du et al., 2019; Venturi et al., 2019).

Unfortunately, the $\mathcal{O}\big(d^3r^3(\frac{n}{r})^{3r}\big)$ computational complexity of (Pilanci and Ergen, 2020) is prohibitively high. The reason behind this high complexity is two folds:

- The size of the convex program grows exponentially in the training data matrix rank r. This high complexity is inherent due to the large number of possible ReLU activation patterns, and thus can be hard to reduce. While highly undesirable from a theoretical standpoint, this complexity is not a deal-breaker in practice: Pilanci and Ergen (2020) show that a heuristic stochastic approximation that forms much smaller convex optimizations works surprisingly well. In this work, we analyze this approximation and theoretically show that for a given level of suboptimality, the required size of the convex training programs is linear in the number of training training data points n.
- The convex training formulation is constrained. The naive choice of algorithm for solving a constrained convex optimization is often the interior-point method (IPM). The per-step computational complexity of IPM is cubic in the number of optimization variables. Since IPM does not take advantage of the problem structures, there is an enormous room for improvements. The focus of this paper is thus to develop more efficient algorithms that exploit the problem structure and achieve a faster global convergence. Specifically, an algorithm based on ADMM with a quadratic per-iteration complexity, as wells an Sampled Convex Program (SCP)-based algorithm with a linear per-iteration complexity, are introduced.

Detailed comparisons among the ADMM-based algorithm, the SCP-based algorithm, the original convex training algorithm in (Pilanci and Ergen, 2020), and back-propagation SGD are presented in Table 1. Compared to IPM, our ADMM-based algorithm converges significantly faster to a moderate accuracy with a much improved computational complexity. Compared with SGD back-propagation, ADMM has a higher theoretical complexity but is guaranteed to linearly converge to a global optimum. The ADMM training method balances global convergence and efficiency.

Prior literature has considered the application of the ADMM method to the training of ANNs (Taylor et al., 2016; Wang et al., 2019). These works used ADMM to separate the activations and the weights of each layer, enabling parallel computing. Compared to the SGD back-propagation, their ADMM formulations are gradient-free and immune to issues such as vanishing gradients and poor conditioning. While Wang et al. (2019) proved that their ADMM algorithm converges at an $\mathcal{O}(1/t)$ rate (t is the number of iterations here) to a critical point of the augmented Lagrangian of the training formulation, there is no guarantee that such critical point is a global optimizer of the considered cost function. In contrast, this paper uses ADMM as an efficient convex optimization algorithm, focusing less on the separation properties of ADMM. The novel ADMM-based algorithm discussed in this work has an entirely different splitting scheme and bases on the convex formulations conceived by

Method	Complexity	Global convergence
IPM (Pilanci and Ergen, 2020)	$\mathcal{O}\big(d^3r^3(\frac{n}{r})^{3r}\big)^{\dagger}$	Superlinear to the global optimum.
ADMM (exact)	$\mathcal{O}\big(d^2r^2(\frac{n}{r})^{2r}\big)^{\dagger}$	Rapid to a moderate accuracy; linear to the global optimum.
ADMM (approximate)	$\mathcal{O}\big(n^2d^2\big)^{\S}$	Rapid to a moderate accuracy; linear to an approximate global optimum.
SCP layer-wise	$\mathcal{O}\big(n^2\big)^{\S}$	Towards an approximate global optimum; $\mathcal{O}\big(1/T\big)$ rate for weakly convex loss; linear for strongly convex loss.
SGD back-propagation	$\mathcal{O}\big(mnd\big)^{\ddagger}/\mathcal{O}\big(n^2d\big)^{\dagger}$	No spurious valleys if $m \geq 2n+2$; no general results.

Table 1: Comparisons between the proposed neural network training methods and related methods. The middle column is the per-iteration complexity (per-epoch for back-propagation since the batch size is arbitrary) when the squared loss is considered. n is the number of training points; d is the data dimension; r is the training data matrix rank.
†: Towards the theoretically lowest loss – further increasing network width will not reduce the training loss;
§: Towards a fixed desired level of suboptimality in the sense defined in Theorem 2;
‡: For an arbitrary network width m. Since there exists a globally optimal neural network with no more than $n+1$ active hidden-layer neuron (Lacotte and Pilanci, 2020), the $\mathcal{O}(mnd)$ bound for SGD back-propagation evaluates to $\mathcal{O}(n^2d)$.

Pilanci and Ergen (2020). More importantly, our ADMM algorithm provably converges to a globally optimal classifier.

Combining SCP analysis and convex training framework leads to a further simplified convex training program that solves unconstrained convex optimizations. This SCP-based method converges to an approximate global optimum. The scale of the convex programs solved in the SCP-based method can be larger than those solved in the ADMM-based algorithm. However, the unconstrained nature enables the use of gradient methods and their stochastic and accelerated variants. Gradient updates are much cheaper than ADMM updates but generally converge slower. Intuitively, this simplified convex training method samples a large number of hidden-layer weights and only optimizes the output layer, drawing connections to layer-wise training. The idea of layer-wise training is not entirely new. This idea has previously been applied to training generative models (Bengio et al., 2006a) and convolutional models (Jangid and Srivastava, 2018). Unlike this work, these previous works focus on inserting more layers into a shallow network to form a deep network. More recently, (Belilovsky et al., 2019) designed a layer-wise training scheme that concatenates one-hidden-layer ANNs into a deep network, where each layer reduces the training error. This concatenation approach can be combined with the convex training of one-hidden-layer ANNs discussed in this work, ultimately leading to training deep networks with convex optimization.

High-performance ANNs can be vulnerable to adversarial attacks. In the field of computer vision, for instance, slight manipulations in the input images can elicit misclassifications in

neural networks with high confidence (Szegedy et al., 2014; Moosavi-Dezfooli et al., 2016; Goodfellow et al., 2015). ANNs can also apply to controls tasks, where robustness is a high priority. However, an adversarial attack on the underlying ANN may cause the control system to fail (Huang et al., 2017). Thus, it is crucial to analyze the adversarial robustness of ANNs, especially when applied to controls and other safety-critical technologies such as autonomous driving.

While there have been studies on robustness certification (Anderson et al., 2020; Ma and Sojoudi, 2020), researchers have also been working extensively on training classifiers whose predictions are robust to input perturbations (Kurakin et al., 2017; Goodfellow et al., 2015; Huang et al., 2015). "Adversarial training" is one of the most effective methods to train robust classifiers, compared with other methods such as obfuscated gradients (Athalye et al., 2018). Adversarial training replaces the standard loss function with an adversarial loss function and solves a bi-level mini-max optimization to train neural networks. More recently, (Cohen et al., 2019) analyzed the feasibility of achieving robustness via "randomized smoothing". Different from adversarial training, this method tends to smooth the decision boundary, making it more suitable for defending ℓ_2 attacks rather than the more common ℓ_∞ attacks (Blum et al., 2020).

When adversarial training is considered, the aforementioned issues of SGD back-propagation become worse: adversarial training can be highly unstable in practice, and convergence properties are pessimistic. Furthermore, most existing attack methods do not guarantee to generate the worst-case inputs. Therefore, extending convex training to adversarial training is crucial. In our conference version (Bai et al., 2021), we built upon the above results to develop "convex adversarial training", explicitly focusing on the cases of hinge loss (for binary classification) and squared loss (for regression). We theoretically showed that solving the proposed robust convex optimizations trains robust ANNs and empirically demonstrated the efficacy and advantages over traditional methods. This work extends the analysis to the binary cross-entropy loss and discusses the extensibility to more complex ANN architectures.

Previously, researchers have applied convex relaxation techniques to adversarial training. These works obtain robust convex certifications (semi-definite program (SDP) (Raghunathan et al., 2018) or linear program (LP) (Wong and Kolter, 2018)) that upper-bound the inner maximization of the adversarial training formulation and use weak duality to develop robust loss functions that can be optimized with back-propagation. Note that while these works use convex relaxation, the resulting training formulations are still non-convex. Furthermore, since multiple layers of relaxations stack together in these works, the analysis can be too conservative. Our numerical experiments confirm this speculation.

The structure of this paper is as follows:

- First, in Section 2 we mathematically bound the suboptimality of an approximate convex training formulation, significantly reducing the problem size.
- Then, in Section 3, we apply the ADMM algorithm to the "convex neural network training" framework and obtain a novel training method that efficiently optimizes ANNs with a linear global convergence guarantee. We also use the SCP framework to provide a theoretical intuition on the scalability and the optimality of a heuristic

approximation procedure that makes convex training practical. Coupling the ADMM algorithm and the approximation yields a quadratic (with respect to training data size) overall per-iteration computational complexity, a noticeable improvement compared with previous results.

- Then, in Section 4, we perform a convex relaxation on the neural network training formulation based on the SCP analysis. The result is an alternative convex training scheme that is easy to implement, flexible, and scalable. This SCP-based training method achieves a linear per-iteration complexity when first-order algorithms are applied. The connection between this new training scheme and "layer-wise training" provides new insights into the dynamics of neural network training.
- Next, in Section 5, we show that convex training introduces new possibilities to the "adversarial training" problem by addressing various computational issues associated with the severe non-convexity of adversarial training. We support the theoretical results with numeric experiments on real-world datasets.
- Finally, in Section 6, we provide numerical experiments to verify the effectiveness of the proposed neural network training techniques and show that they extend convex training to various machine learning problems such as image classification.

1.1 Notations

Throughout this work, we focus on fully-connected neural networks with one rectified linear activated (ReLU) hidden layer and a scalar output, defined as

$$\widehat{y} = \sum_{j=1}^{m} \left(Xu_j + b_j \mathbf{1}_n\right)_+ \alpha_j,$$

where $X \in \mathbb{R}^{n\times d}$ is the input data matrix with n data points in $\mathbb{R}^d$ and $\widehat{y} \in \mathbb{R}^n$ is the output vector of the neural network. We denote the target output used for training as $y \in \mathbb{R}^n$. The vectors $u_1, \ldots, u_m \in \mathbb{R}^d$ are the weights of the m neurons in the hidden layer while the scalars $\alpha_1, \ldots, \alpha_m \in \mathbb{R}$ are the weights of the output layer. $b_1 \ldots, b_m \in \mathbb{R}$ are the hidden layer bias terms. The symbol $(\cdot)_+ = \max\{0, \cdot\}$ indicates the ReLU activation function which sets all negative entries of a vector or a matrix to zero. The symbol $\mathbf{1}_n$ defines a column vector with all entries being 1, where the subscript n denotes the dimension of this vector.

Furthermore, for a vector $q \in \mathbb{R}^n$, $\operatorname{sgn}(q) \in \{-1, 0, 1\}^n$ denotes the signs of the entries of q. $[q \geq 0]$ denotes a boolean vector in $\{0, 1\}^n$ with ones at the locations of the nonnegative entries of q and zeros at the remaining locations. The symbol $\operatorname{diag}(q)$ denotes a diagonal matrix $Q \in \mathbb{R}^{n\times n}$ where $Q_{ii} = q_i$ for all i and $Q_{ij} = 0$ for all $i \neq j$. For a vector $q \in \mathbb{R}^n$ and a scalar $b \in \mathbb{R}$, the inequality $q \geq b$ means that $q_i \geq b$ for all $i \in [n]$. The symbol $\odot$ denotes the Hadamard product between two vectors with same dimensionalities. The notation $\|\cdot\|_p$ denotes the ℓ_p-norm within $\mathbb{R}^n$. For a matrix A, the max norm $\|A\|_{\max}$ is defined as $\max_{ij} |a_{ij}|$, where a_{ij} is the entry at the location (i, j).

Moreover, for a set $\mathcal{A}$, the notation $\Pi_{\mathcal{A}}(\cdot)$ denotes the projection onto the set and $|\mathcal{A}|$ denotes the cardinality of the set. The notation $\texttt{prox}_f$ denotes the proximal operator associated

with a function $f(\cdot)$. The notation $R \sim \mathcal{N}(0, I_n)$ indicates that a random variable $R \in \mathbb{R}^n$ is a standard normal random vector. For $P \in \mathbb{N}_+$, we define $[P]$ as the set $\{a \in \mathbb{N}_+ | a \leq P\}$, where $\mathbb{N}_+$ is the set of positive integer numbers.

2. Practical Convex Neural Network Training

2.1 Prior work – convex ANN training

We define the problem of training the above ANN with an ℓ_2 regularized convex loss function $\ell(\widehat{y}, y)$ as:

$$\min_{(u_j, \alpha_j, b_j)_{j=1}^m} \ell\Big(\sum_{j=1}^m \big(Xu_j + b_j \mathbf{1}_n\big)_+ \alpha_j, y \Big) + \frac{\beta}{2} \sum_{j=1}^m \Big(\|u_j\|_2^2 + b_j^2 + \alpha_j^2 \Big).$$

where $\beta > 0$ is a regularization parameter. Without loss of generality, we assume that $b_j = 0$ for all $j \in [m]$. We can safely make this simplification because concatenating a column of ones to the data matrix X absorbs the bias terms into the weight vectors.

The simplified training problem is then:

$$\min_{(u_j, \alpha_j)_{j=1}^m} \ell\Big(\sum_{j=1}^m (Xu_j)_+ \alpha_j, y \Big) + \frac{\beta}{2} \sum_{j=1}^m \big(\|u_j\|_2^2 + \alpha_j^2 \big). \tag{1}$$

Consider a set of diagonal matrices $\{\mathrm{diag}([Xu \geq 0]) | u \in \mathbb{R}^d\}$, and let the distinct elements of this set be denoted as $D_1, \ldots, D_P$. The constant P corresponds to the total number of partitions of $\mathbb{R}^d$ by hyperplanes passing through the origin that are also perpendicular to the rows of X (Pilanci and Ergen, 2020). Intuitively, P can be regarded as the number of possible ReLU activation patterns associated with X.

Consider the convex optimization problem

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

and its dual formulation

$$\max_v -\ell^*(v) \qquad \text{s.t.} \quad |v^\top (Xu)_+| \leq \beta, \ \ \forall u : \|u\|_2 \leq 1 \tag{3}$$

where $\ell^*(v) = \max_z z^\top v - \ell(z, y)$ is the Fenchel conjugate function. Note that (3) is a convex semi-infinite program. **FIXIT**The next theorem borrowed from (Pilanci and Ergen, 2020) explains the relationship between the non-convex training problem (1), the convex problem (2), and the dual problem (3) when the neural network is sufficiently wide.

Theorem 1 ((Pilanci and Ergen, 2020)) *Let $(v_i^\star, w_i^\star)_{i=1}^P$ denote a solution of (2) and define $m^\star$ as $|\{i : v_i^\star \neq 0\}| + |\{i : w_i^\star \neq 0\}|$. Suppose that the neural network width m is at*

Algorithm 1 Practical convex training

1: Generate P_s distinct diagonal matrices via $D_h \leftarrow \text{diag}([Xa_h \geq 0])$, where $a_h \sim \mathcal{N}(0, I_d)$ i.i.d. for all $h \in [P_s]$.

2: Solve

$$p_{s1}^\star = \min_{(v_h, w_h)_{h=1}^{P_s}} \ell\Big(\sum_{h=1}^{P_s} D_h X(v_h - w_h), y\Big) + \beta \sum_{h=1}^{P_s} \big(\|v_h\|_2 + \|w_h\|_2\big) \tag{5}$$
$$\text{s.t.} \quad (2D_h - I_n)Xv_h \geq 0,\ (2D_h - I_n)Xw_h \geq 0, \quad \forall h \in [P_s].$$

3: Recover $u_1, \ldots, u_{m_s}$ and $\alpha_1, \ldots, \alpha_{m_s}$ from the solution $(v_{s_h}^\star, w_{s_h}^\star)_{h=1}^{P_s}$ of (5) using (4).

least $m^\star$, where $m^\star$ is upper-bounded by $n+1$. If the loss function $\ell(\cdot, y)$ is convex, then (1), (2), and (3) share the same optimal objective. The optimal network weights $(u_j^\star, \alpha_j^\star)_{j=1}^m$ can be recovered using the formulas

$$\begin{aligned}(u_{j_{1i}}^\star, \alpha_{j_{1i}}^\star) &= \Big(\frac{v_i^\star}{\sqrt{\|v_i^\star\|_2}}, \sqrt{\|v_i^\star\|_2}\Big) \qquad \text{if } v_i^\star \neq 0;\\ (u_{j_{2i}}^\star, \alpha_{j_{2i}}^\star) &= \Big(\frac{w_i^\star}{\sqrt{\|w_i^\star\|_2}}, -\sqrt{\|w_i^\star\|_2}\Big) \quad \text{if } w_i^\star \neq 0.\end{aligned} \tag{4}$$

where the remaining $m - m^\star$ neurons are chosen to have zero weights.

The worst-case computational complexity of solving (2) for the case of squared loss is $\mathcal{O}\big(d^3 r^3 (\frac{n}{r})^{3r}\big)$ using standard interior-point solvers (Pilanci and Ergen, 2020). Here, r is the rank of the data matrix X and in many cases $r = d$. Such complexity is polynomial in n but exponential in r. This complexity is already a significant improvement over previous methods but still prohibitively high for many practical applications. Such high complexity is due to the large number of D_i matrices, which is upper-bounded by $\min\big\{2^n, 2r\big(\frac{e(n-1)}{r}\big)^r\big\}$ (Pilanci and Ergen, 2020).

2.2 A practical algorithm for convex training

A natural direction of mitigating this high complexity is to reduce the number of D_i matrices by sampling a subset of them. This idea leads to Algorithm 1, which approximately solves the training problem. Algorithm 1 is an instance of the approximation described in (Pilanci and Ergen, 2020, Remark 3.3), but Pilanci and Ergen (2020) did not provide theoretical insights regarding its level of suboptimality. The following theorem bridges the gap by providing a probabilistic bound on the suboptimality of the neural network trained with Algorithm 1. Algorithm 1 can train ANNs with widths much less than $m^\star$. The following theorem provides a probabilistic bound on the level of suboptimality of the neural network trained using Algorithm 1.

Theorem 2 *Consider an additional diagonal matrix D_{P_s+1} sampled uniformly, and then construct*

$$p_{s2}^\star = \min_{(v_h,w_h)_{h=1}^{P_s+1}} \ell\Big(\sum_{h=1}^{P_s+1} D_h X(v_h - w_h), y\Big) + \beta \sum_{h=1}^{P_s+1} \big(\|v_h\|_2 + \|w_h\|_2\big) \tag{6}$$
$$\text{s.t.} \quad (2D_h - I_n)Xv_h \geq 0,\ (2D_h - I_n)Xw_h \geq 0,\ \ \forall h \in [P_s + 1].$$

It holds that $p_{s2}^\star \leq p_{s1}^\star$. *Furthermore, if* $P_s \geq \min\left\{\frac{n+1}{\psi\xi} - 1, \frac{2}{\xi}(n + 1 - \log\psi)\right\}$, *where* ψ *and* ξ *are preset confidence level constants between 0 and 1, then with probability at least* $1 - \xi$, *it holds that* $\mathbb{P}\{p_{s2}^\star < p_{s1}^\star\} \leq \psi$.

The proof of Theorem 2 is presented in Section C.1. Intuitively, Theorem 2 shows that sampling an additional D_{P_s+1} matrix will not reduce the training cost with high probability when P_s is large. One can recursively apply this bound T times to show that when P_s is large, the solution with P_s matrices is close to the solution with $P_s + T$ matrices for an arbitrary number T. Therefore, although the theorem does not directly bound the gap between the approximated optimization problem and its exact counterpart, it states that the optimality gap due to sampling is not too large for a suitable value of P_s, and the trained network is nearly optimal.

Compared with the exponential relationship between P and r, a satisfactory value of P_s should be on the order of $\mathcal{O}(n)$. Therefore, P_s is linear in n and is independent from r. Thus, when r is large, solving the approximated formulation (5) is significantly (exponentially) more efficient than solving the exact formulation (2). On the other hand, Algorithm 1 is no longer deterministic due to the stochastic sampling of the D_h matrices, and yields solutions that upper-bound those of (2). While Algorithm 1 is not exact, we have verified empirically (shown in Section 6.1) that even when P_s is significantly smaller than P, Algorithm 1 still reliably returns a low training cost.

Since the confidence constants ψ and ξ are no greater than one, Theorem 2 only applies to overparameterized ANNs, where $P_s \geq n$. Intuitively, selecting P_s in practice is equivalent to choosing the neural network width. While Theorem 2 provides a guideline on how P_s should scale with n, selecting a much smaller P_s will not necessarily become an issue. Our experiments show that even when P_s is much less than n, Algorithm 1 still reliably returns high-performance classifiers.

FIXIT(Pilanci and Ergen, 2020) shows that there exists a globally optimal ANN whose width is at most $n+1$, while Theorem 2 only provides a probabilistic bound for ANNs wider than n. Although Theorem 2 seems loose by this comparison, it bounds a different quantity and is meaningful. The bound by (Pilanci and Ergen, 2020) does not provide a method that scales linearly, and therefore while a globally optimal ANN narrower than $n + 1$ exists, finding such an ANN requires solving a convex program with an exponential number of constraints. In contrast, Theorem 2 characterizes the optimality of a convex optimization with a manageable number of constraints.

3. An ADMM Algorithm for Global Neural Network Training

The convex ReLU neural network training program (2) may be solved with the interior point method (IPM). The IPM is an iterative algorithm that repeatedly performs Newton updates.

Each Newton update requires solving a linear system, which has a cubic complexity, hindering the application of IPM to large-scale optimization problems. Unfortunately, large-scale problems are ubiquitous in the field of machine learning. This section proposes an algorithm based on the alternating direction method of multipliers (ADMM). ADMM breaks down the optimization (2) to smaller subproblems that are easier to solve. When $\ell(\cdot)$ is the squared loss, each subproblem has a closed-form solution. We will show that the complexity of each ADMM iteration is linear in n and quadratic in d and P, and the number of required ADMM steps to reach a desired precision is logarithmic in the precision level. When other convex loss functions are used, a closed-form solution may not always exist. We illustrate that iterative methods can solve the subproblems for general convex losses efficiently.

Define $F_i := D_i X$ and $G_i := (2D_i - I_n)X$ for all $i \in [P]$. Furthermore, we introduce v_i, w_i, s_i, t_i as slack variables and let $v_i = u_i$, $w_i = z_i$, $s_i = G_i v_i$, and $t_i = G_i w_i$. For a vector $q = (q_1, \dots, q_n) \in \mathbb{R}^n$, let the indicator function of the positive quadrant $\mathbb{I}_{\geq 0}$ be defined as

$$\mathbb{I}_{\geq 0}(q) := \begin{cases} 0 & \text{if } q_i \geq 0,\ \forall i \in [N]; \\ +\infty & \text{otherwise.} \end{cases}$$

The convex training formulation (2) can be reformulated as a convex optimization problem with positive quadrant indicator functions and linear equality constraints:

$$\begin{aligned} \min_{(v_i,w_i,s_i,t_i,u_i,z_i)_{i=1}^P} \ & \ell\Big(\sum_{i=1}^P F_i(u_i - z_i), y\Big) + \beta \sum_{i=1}^P \|v_i\|_2 + \beta \sum_{i=1}^P \|w_i\|_2 + \sum_{i=1}^P \mathbb{I}_{\geq 0}(s_i) + \sum_{i=1}^P \mathbb{I}_{\geq 0}(t_i) \\ \text{s.t.} \quad & G_i u_i - s_i = 0, \quad G_i z_i - t_i = 0, \quad v_i - u_i = 0, \quad w_i - z_i = 0, \quad \forall i \in [P]. \end{aligned} \tag{7}$$

Next, we simplify the notations by concatenating the matrices. Define

$$\begin{gathered} u := [u_1^\top \ \cdots \ u_P^\top \ \ z_1^\top \ \cdots \ z_P^\top]^\top, \quad v := [v_1^\top \ \cdots \ v_P^\top \ \ w_1^\top \ \cdots \ w_P^\top]^\top, \\ s := [s_1^\top \ \cdots \ s_P^\top \ \ t_1^\top \ \cdots \ t_P^\top]^\top, \\ F := [F_1 \ \cdots \ F_P \ \ -F_1 \ \cdots \ -F_P], \ \text{ and } \ G := \text{blkdiag}(G_1, \cdots, G_P, G_1, \cdots, G_P), \end{gathered}$$

where $\text{blkdiag}(\cdot, \dots, \cdot)$ denotes the block diagonal matrix formed by the submatrices in the parentheses. The formulation (7) is then equivalent to the compact notation

$$\min_{v,s,u} \ell(Fu, y) + \beta \|v\|_{2,1} + \mathbb{I}_{\geq 0}(s) \quad \text{s.t.} \quad \begin{bmatrix} I_{2dP} \\ G \end{bmatrix} u - \begin{bmatrix} v \\ s \end{bmatrix} = 0, \tag{8}$$

where $\|\cdot\|_{2,1}$ denotes the group sparse regularization and I_{2dP} is the idendity matrix in $\mathbb{R}^{2dP \times 2dP}$. The corresponding augmented Lagrangian (Hestenes, 1969) of (8), denoted as $L(u, v, s, \nu, \lambda)$, is:

$$\begin{aligned} &L(u, v, s, \nu, \lambda) = \\ &\quad \ell(Fu, y) + \beta \|v\|_{2,1} + \mathbb{I}_{\geq 0}(s) + \frac{\gamma_a}{2}\Big(\|u - v + \lambda\|_2^2 - \|\lambda\|_2^2\Big) + \frac{\gamma_a}{2}\Big(\|Gu - s + \nu\|_2^2 - \|\nu\|_2^2\Big) \end{aligned}$$

where $\lambda := [\lambda_{11} \ \dots \ \lambda_{1P} \ \ \lambda_{21} \ \dots \ \lambda_{2P}]^\top \in \mathbb{R}^{2dP}$ and $\nu := [\nu_{11} \ \dots \ \nu_{1P} \ \ \nu_{21} \ \dots \ \nu_{2P}]^\top \in \mathbb{R}^{2nP}$ are dual variables, and γ_a is a positive step-size constant.

Algorithm 2 An ADMM algorithm for the convex neural network training problem.

repeat

Solve $$u^{k+1} = \arg\min_{u} \ell(Fu, y) + \frac{\gamma_a}{2}\|u - v^k + \lambda^k\|_2^2 + \frac{\gamma_a}{2}\|Gu - s^k + \nu^k\|_2^2 \tag{9a}$$

Solve $$\begin{bmatrix} v^{k+1} \\ s^{k+1} \end{bmatrix} = \arg\min_{v,s} \beta\|v\|_{2,1} + \mathbb{I}_{\geq 0}(s) + \frac{\gamma_a}{2}\|u^{k+1} - v + \lambda^k\|_2^2 + \frac{\gamma_a}{2}\|Gu^{k+1} - s + \nu^k\|_2^2 \tag{9b}$$

Dual update: $$\begin{bmatrix} \lambda^{k+1} \\ \nu^{k+1} \end{bmatrix} = \begin{bmatrix} \lambda^k + \gamma_a(u^{k+1} - v^{k+1}) \\ \nu^k + \gamma_a(Gu^{k+1} - s^{k+1}) \end{bmatrix} \tag{9c}$$

until FIXIT

We can apply the ADMM iterations described in Algorithm 2 to globally optimize (8). As will be shown next, (9b) and (9c) have simple closed-form solutions. The update (9a) has a closed-form solution when $\ell(\cdot)$ is the squared loss, and can be efficiently solved numerically for general convex loss functions.

The following theorem shows the linear convergence of Algorithm 2, with the proof provided in Appendix C.2:

Theorem 3 *If $\ell(\widehat{y}, y)$ is strictly convex and continuously differentiable with a uniform Lipschitz continuous gradient with respect to $\widehat{y}$, then the sequence $\{(u^k, v^k, s^k, \lambda^k, \nu^k)\}$ generated by Algorithm 2 converges linearly to an optimal primal-dual solution for (8), provided that the step size γ_a is sufficiently small.*

Many popular loss functions satisfy the conditions of Theorem 3. Examples include the squared loss (for regression) and the binary cross-entropy loss coupled with the tanh output activation (for binary classification, details shown in Section 6.2.4).

When we apply Algorithm 2 to solve the approximated convex training formulation (5), Algorithm 2 becomes a subalgorithm of Algorithm 1.

3.1 s and v updates

The update step (9b) can be separated for v^{k+1} and s^{k+1} as:

$$v^{k+1} = \arg\min_{v} \beta\|v\|_{2,1} + \frac{\gamma_a}{2}\|u^{k+1} - v + \lambda^k\|_2^2; \tag{10a}$$

$$s^{k+1} = \arg\min_{s} \mathbb{I}_{\geq 0}(s) + \|Gu^{k+1} - s + \nu^k\|_2^2 = \arg\min_{s \geq 0} \|Gu^{k+1} - s + \nu^k\|_2^2. \tag{10b}$$

Note that (10a) can be separated for each v_i and w_i (allowing parallelization) and solved analytically using the formulas:

$$\begin{aligned} v_i^{k+1} &= \arg\min_v \beta\|v_i\|_2 + \frac{\gamma_a}{2}\|u_i^{k+1} - v + \lambda_{1i}^k\|_2^2 = \texttt{prox}_{\frac{\beta}{\gamma_a}\|\cdot\|_2}\big(u_i^{k+1} + \lambda_{1i}^k\big) \\ &= \left(1 - \frac{\beta}{\gamma_a \cdot \big\|u_i^{k+1} + \lambda_{1i}^k\big\|_2}\right)_+ \big(u_i^{k+1} + \lambda_{1i}^k\big), \qquad \forall i \in [P]; \\ w_i^{k+1} &= \arg\min_v \beta\|w_i\|_2 + \frac{\gamma_a}{2}\|s_i^{k+1} - w + \lambda_{2i}^k\|_2^2 = \texttt{prox}_{\frac{\beta}{\gamma_a}\|\cdot\|_2}\big(z_i^{k+1} + \lambda_{2i}^k\big) \\ &= \left(1 - \frac{\beta}{\gamma_a \cdot \big\|z_i^{k+1} + \lambda_{2i}^k\big\|_2}\right)_+ \big(z_i^{k+1} + \lambda_{2i}^k\big), \qquad \forall i \in [P], \end{aligned}$$

where $\texttt{prox}_{\frac{\beta}{\gamma_a}\|\cdot\|_2}$ denotes the proximal operation on the function $f(\cdot) = \frac{\beta}{\gamma_a}\|\cdot\|_2$. The computational complexity of finding v_i and w_i is $\mathcal{O}(d)$. Similarly, (10b) can also be separated for each s_i and t_i and solved analytically using the formulas:

$$\begin{aligned} s_i^{k+1} &= \arg\min_{s_i \geq 0} \big\|G_i u_i^{k+1} - s_i + \nu_{1i}^k\big\|_2^2 = \Pi_{\geq 0}\big(G_i u_i^{k+1} + \nu_{1i}^k\big) = \big(G_i u_i^{k+1} + \nu_{1i}^k\big)_+, \quad \forall i \in [P]; \\ t_i^{k+1} &= \arg\min_{t_i \geq 0} \big\|G_i z_i^{k+1} - s_i + \nu_{2i}^k\big\|_2^2 = \Pi_{\geq 0}\big(G_i z_i^{k+1} + \nu_{2i}^k\big) = \big(G_i z_i^{k+1} + \nu_{2i}^k\big)_+, \quad \forall i \in [P]. \end{aligned}$$

where $\Pi_{\geq 0}$ denotes the projection onto the non-negative quadrant. The computational complexity of finding s_i and t_i is $\mathcal{O}(n)$. The updates (10a) and (10b) can be performed in $\mathcal{O}(nP + dP)$ time in total.

3.2 u updates

The u update step depends on the specific structure of $\ell(\cdot)$. For the squared loss, the u update step can be solved in closed form. For many other loss functions, the update can be performed with numerical methods.

3.2.1 Squared loss

The squared loss $\ell(\widehat{y}, y) = \frac{1}{2}\|\widehat{y} - y\|_2^2$ is a commonly used loss function in machine learning. It is widely used for regression tasks, but can also be used for classification. For the squared loss, (9a) amounts to

$$u^{k+1} = \arg\min_u \left\{\|Fu - y\|_2^2 + \frac{\gamma_a}{2}\|u - v^k + \lambda^k\|_2^2 + \frac{\gamma_a}{2}\|Gu - s^k + \nu^k\|_2^2\right\} \tag{11}$$

Setting the gradient to zero yields that

$$\big(I + \tfrac{1}{\gamma_a}F^\top F + G^\top G\big)u^{k+1} = \tfrac{1}{\gamma_a}F^\top y + v^k - \lambda^k + G^\top s^k - G^\top \nu^k \tag{12}$$

Therefore, the u update can be performed by solving the linear system (12) in each iteration. While solving a linear system $Ax = b$ for a square matrix A has a cubic time complexity in

general, by taking advantage of the structure of (12), a quadratic per-iteration complexity can be achieved. Specifically, the matrix $I + \frac{1}{\gamma_a} F^\top F + G^\top G$ is symmetric, positive definite, and fixed throughout the ADMM iterations. In general, solving $Ax = b$ for some symmetric $A \in \mathbb{S}^{2dP \times 2dP}$, $A \succ 0$ and $b \in \mathbb{R}^{2dP}$ can be done via the procedure:

1. Perform the Cholesky decomposition $A = LL^\top$, where L is lower-triangular (cubic in $2dP$);
2. Solve $L\widehat{b} = b$ by forward substitution (quadratic in $2dP$);
3. Solve $L^\top x = \widehat{b}$ by back substitution (quadratic in $2dP$).

Throughout the ADMM iterations, the first step only needs to be performed once, while the second and the third steps are required for every iteration. Since the dimension of the matrix $(I + \frac{1}{\gamma_a} F^\top F + G^\top G)$ is $2dP \times 2dP$, the per-iteration time complexity of the u update is $\mathcal{O}(d^2P^2)$, making it the most time-consuming step of the ADMM algorithm when d and P are large. Therefore, the overall complexity of a full ADMM primal-dual iteration for the case of squared loss is $\mathcal{O}(nP + d^2P^2)$, which is quadratic. In contrast, the linear system for IPM's Newton updates can be completely different for each iteration, and thus generally has a cubic complexity. Therefore, the proposed ADMM method achieves a notable efficiency improvement over the IPM baseline.

In the case when the approximated formulation (5) is considered and P_s diagonal matrices are sampled in place of the full set of P matrices, obtaining a given level of optimality requires P_s to be linear in n, as discussed in Section 2. Coupling with the above analysis, we obtain an overall per-iteration complexity of $\mathcal{O}(d^2n^2)$, a significant improvement compared with the $\mathcal{O}\big(d^3r^3(\frac{n}{r})^{3r}\big)$ per-iteration complexity of (Pilanci and Ergen, 2020). The total computational complexity for reaching a solution satisfying **FIXIT**is $\mathcal{O}(d^2n^2/\log(\epsilon))$. In Section 6.2, we provide numerical experiments to demonstrate that the improved efficiency of the ADMM algorithm enables the application of convex ANN training on image classification tasks, which was not possible before. Moreover, our experiments show that it does not require a high optimization precision to achieve a favorable prediction accuracy.

3.2.2 General convex loss functions

When a general convex loss function $\ell(\widehat{y}, y)$ is considered, a closed-form solution to (9a) does not always exist and one may need to use iterative methods to solve (9a). One natural use of an iterative optimization method is gradient descent. However, for large-scale problems, a full gradient evaluation can be too expensive. To address this issue, we exploit the symmetric and separable property of each u_i and z_i in (9a) and propose an application of the randomized block coordinate descent (RBCD) method. The details of RBCD are presented in Algorithm 3. The superscript $^+$ denotes the updated quantities for each iteration, and the notation γ_r is the step size. In practice, the RBCD step size γ_r can be adaptively chosen with a backtracking line search. Steps 5 and 6 of Algorithm 3 are derived via the chain rule of differentiation. It can be verified that (9a) is always strongly convex because its second term is strongly convex while the first and third terms are convex. (Lu and Xiao, 2015, Theorem 1) has shown that when minimizing strongly-convex functions, RBCD converges linearly.

The theoretical convergence rate is higher when the convexity of (9a) is stronger and P is smaller.

Algorithm 3 Randomized Block Coordinate Descent (RBCD)

Initialize $\widehat{y} = \sum_{i=1}^{P} F_i(u_i - z_i)$;
Fix $\tilde{s}_i = G_i^\top (s_i - \nu_{1i})$, $\tilde{t}_i = G_i^\top (t_i - \nu_{2i})$ for all $i \in [P]$;
Select accuracy thresholds $\tau > 0, \varphi > 0$;
repeat
  $\tilde{y} \leftarrow \nabla_{\widehat{y}} \ell(\widehat{y}, y)$
  Uniformly select i from $[P]$ at random;
  $u_i^+ \leftarrow u_i - \gamma_r F_i^\top \tilde{y} - \gamma_r \gamma_a (u_i - v_i + \lambda_{1i} + G_i^\top G_i u_i - \tilde{s}_i)$;
  $z_i^+ \leftarrow z_i + \gamma_r F_i^\top \tilde{y} - \gamma_r \gamma_a (z_i - w_i + \lambda_{2i} + G_i^\top G_i z_i - \tilde{t}_i)$;
  $\widehat{y}^+ \leftarrow \widehat{y} + F_i\big((u_i^+ - z_i^+) - (u_i + z_i)\big)$;
until $\|\nabla_u L(u, v, s, \nu, \lambda)\|_2 \leq \dfrac{\varphi}{\max\{\tau, \|u\|_2\}}$.

Furthermore, $G_i^\top G_i = X^\top X$ for all $i \in [P]$. To see this, recall that $G_i = (2D_i - I_n)X$ by definition. Since $(2D_i - I_n)$ is a diagonal matrix with all entries being ± 1, it holds that $(2D_i - I_n)^\top (2D_i - I_n) = I_n$. Thus, $G_i^\top G_i = X^\top (2D_i - I_n)^\top (2D_i - I_n) X = X^\top X$. Consequently, $X^\top X$ can be calculated in advance, and there is no need to calculate $G_i^\top G_i$ in each RBCD iteration. Therefore, the most expensive calculations per RBCD update have the followings complexities:

$F_i^\top \tilde{y}$	$F_i\Big((u_i^+ - z_i^+) - (u_i + z_i)\Big)$	$(X^\top X) u_i$	$(X^\top X) z_i$
$\mathcal{O}(nd)$	$\mathcal{O}(nd)$	$\mathcal{O}(d^2)$	$\mathcal{O}(d^2)$.

While it can be costly to solve (9a) to a high accuracy using iterative methods, especially during the early iterations of ADMM, (Eckstein and Yao, 2017, Algorithm 1, Prop 6) has shown that even when (9a) is solved approximately, as long as the accuracy threshold φ of each ADMM iteration forms a convergent sequence, the ADMM algorithm can eventually converge to the global optimum of (8). Each iterative solution of the u-update subproblem can also take advantage of warm-starting by initializing from the result of the previous ADMM iteration. In other words, in practical implementations, we alternate between an ADMM update and several RBCD updates in a disciplined manner.

4. SCP-based Layer-wise Convex Training

While the practical training formulation (5) coupled with the ADMM algorithm (Algorithm 2) was proved to vastly improve the efficiency and the practicality of globally optimizing neural networks compared with prior works, the complexity of the aforementioned methods can still be too high for large-scale machine learning problems due to the complicated structure of (2). A natural question is then: Can we build simpler convex training formulations that are easier to optimize? In this section, we propose a "sampled convex program (SCP)"-based

alternative approach to approximately globally optimize scalar-output one-hidden-layer neural networks. This approach constructs scalable unconstrained convex optimization problems with simpler structures. Unconstrained convex optimization problems are much easier to numerically solve compared to constrained ones. Scalable and simple first-order methods can be easily applied to unconstrained convex programs, while the same cannot be said for constrained optimization in general due to feasibility issues.

Compared with the ADMM approach in Algorithm 2, the SCP approach is easier to implement and has a lower per-iteration complexity. The tradeoff is that while Algorithm 2 can be applied to find the exact global minimum of (1) (albeit with an exponential complexity with respect to the data matrix rank), the SCP approach only finds an approximately global solution. In the approximate case, the qualities of the ADMM solution and the SCP solution can both be characterized.

4.1 One-shot sampling of hidden-layer weights

In this subsection, we focus on scalar-output one-hidden-layer ReLU neural networks. Appendix B.2 discusses the extensions to vector-output networks, which see a broader set of applications.

Pilanci and Ergen (2020) has shown that the non-convex training formulation (1) has the same global optimum as

$$p^\star = \min_{(u_j,\alpha_j)_{j=1}^m} \ell\Big(\sum_{j=1}^m (Xu_j)_+\alpha_j, y\Big) + \frac{\beta}{2}\sum_{j=1}^m |\alpha_j| \quad \text{s.t.} \quad \|u_j\|_2 \le 1,\ \forall j \in [m]. \tag{13}$$

Note that we can replace the perturbation set $\{u \mid \|u\|_2 \le 1\}$ with $\{u \mid \|u\|_2 = 1\}$ without changing the optimum. This is because for any pair (u_j, α_j) such that $\|u_j\|_2 < 1$, replacing (u_j, α_j) with the scaled weights $(\frac{u_j}{\|u_j\|_2}, \|u_j\|_2 \cdot \alpha_j)$ will reduce the regularization term of (13) while keeping the loss function term unchanged, meaning that the optimal $u_j^\star$ must satisfy $\|u_j^\star\|_2 = 1$.

To approximate the semi-infinite program (13), we randomly sample a total of N vectors, namely $u_1, \ldots, u_N$, on the ℓ_2 unit norm sphere $\mathcal{S}^{d-1}$ following a uniform distribution. It is well-known that such a procedure can be performed by randomly sampling $\widehat{u}_i \sim \mathcal{N}(0, I_d)$ for all $i \in [N]$ and projecting each $\widehat{u}_i$ onto the unit ℓ_2 norm sphere by calculating $u_i = \frac{\widehat{u}_i}{\|\widehat{u}_i\|_2}$ for all $i \in [N]$. Next, $u_1, \ldots, u_N$ are used to construct the following SCP:

$$p_{s3}^\star = \min_{(\alpha_i)_{i=1}^N} \ell\Big(\sum_{i=1}^N \big(Xu_i\big)_+\alpha_i, y\Big) + \beta\sum_{i=1}^N |\alpha_i|, \tag{14}$$

where the sampled hidden-layer weights $(u_i)_{i=1}^N$ are fixed.

The finite-dimensional unconstrained convex formulation (14) is a relaxation of (13), and can be used as a surrogate for the optimization (1) to approximately globally optimize one-hidden-layer neural networks. The formulation (14) optimizes the output layer of the ANN while keeping the hidden layer fixed. When the squared loss $\ell(\widehat{y}, y) = \frac{1}{2}\|\widehat{y} - y\|_2^2$ is

considered, (14) is a Lasso Regression problem. Intuitively, the sampled hidden-layer weights transform the training data points into a higher-dimensional space. While some of the sampled weights will inevitably be far from the optimum weights for the neural network, the ℓ_1 regularization term promotes sparsity, providing a tendency to assign zero weights to "disable" the suboptimal hidden neurons.

The SCP training formulation (14) recovers the training formulation of one-hidden-layer RVFL and ELM (**FIXIT**). Such an equivalence shows that training an ELM is a convex relaxation to training a neural network. Compared with traditional ELMs, (14) contains a sparsity-promoting regularization, and requires a different initialization of the untrained hidden layer weights. This connection supports the finding (cite) that neural networks seek sparsity.

The method in this subsection is referred to as "one-shot sampling" because all hidden layer weights are sampled in advance, in contrast with the iterative sampling procedure described in Section 4.2. The neural networks trained with (14) can be suboptimal in terms of empirical loss compared with the network that globally minimizes the non-convex cost function, but are expected to be close to the optimal classifier. The next theorem characterizes the level of suboptimality of the SCP optimizer, with the proof provided in Appendix C.3.

Theorem 4 *Suppose that an additional hidden neuron u_{N+1} is randomly sampled on the unit Euclidean norm sphere via a uniform distribution to augment the neural network. Consider the following formulation to train the augmented network:*

$$p_{s4}^\star = \min_{(\alpha_i)_{i=1}^{N+1}} \ell\Big(\sum_{i=1}^{N+1} \big(Xu_i\big)_+ \alpha_i, y\Big) + \beta \sum_{i=1}^{N+1} |\alpha_i|. \tag{15}$$

It holds that $p_{s4}^\star \leq p_{s3}^\star$. Furthermore, if $N \geq \min\big\{\frac{n+1}{\psi\xi} - 1, \frac{2}{\xi}(n+1-\log\psi)\big\}$, where ψ and ξ are preset confidence level constants between 0 and 1, then with probability no smaller than $1-\xi$, it holds that $\mathbb{P}\{p_{s4}^\star < p_{s3}^\star\} \leq \psi$.

Intuitively, this bound means that uniformly sampling another hidden layer weight u_{N+1} on the unit norm sphere will not improve the training cost with high probability. For a fixed level of suboptimality, the required scale of the SCP formulation (14) has a linear relationship with respect to the number of training data points.

Similar to Algorithm 1, the SCP (14) converges to an approximate global minimum of the ANN cost function. If we consider training an ANN with the same width using back-propagation, Algorithm 1, and SCP (in this case, $m = P_s = N < P$), then both Algorithm 1 and SCP achieve convexity at the price of leaving out a part of the parameter space. The reason is that Algorithm 1 and SCP both impose assumptions on the network weights. Specifically, Algorithm 1 solves (5), which restricts the ReLU activation pattern of the hidden layer, while the SCP relaxation (14) imposes a stronger restriction by limiting the choice of hidden layer weights. Thus, if $P_s = N$, then (5) is expected to have a larger search space than (14) and may perform better as a consequence. Furthermore, when P_s in (5) is the same as P in (2), then the exact convex reformulation is recovered. However,

recovering the exact counterpart from (14) requires $N \to \infty$, confirming that (14) is a cruder approximation than (5).

However, somewhat surprisingly, from the perspective of the probabilistic optimality, the bound provided by Theorem 4 is the same as the bound associated with Algorithm 1 presented in Theorem 2. The reason is that both bounds are obtained via the sampled convex program analysis framework.

The main advantage of the SCP-based training approach is that when $P_s = N$, the unconstrained optimization (14) is much easier (and thus faster) to solve than the constrained optimization (5). Specifically, the iterative soft-thresholding algorithms (ISTA) (Beck and Teboulle, 2009) and their accelerated or stochastic variants can be readily applied to solve (14). Specifically, ISTA converges at a a linear rate if $\ell\big(\sum_{i=1}^{N}(Xu_i)_+\alpha_i, y\big)$ is strongly convex over each α_i, and converges at a $\mathcal{O}(1/T)$ rate for weakly convex cases, where T is the iteration count. As a result, with the same amount of computational resources, one can solve (14) with $N \gg P_s$, allowing for training wider networks (with stronger representation powers) than those trainable with (2) within a reasonable amount of time. Such an advantage is especially significant when a large-scale problem is considered. Numerical experiments presented in Section 6.3 verify that the SCP relaxation (14) can train accurate classifiers with reasonable computing effort.

When $\ell(\cdot)$ is the squared loss, the SCP formulation (14) evaluates to $\min_\alpha \|H\alpha - y\|_2^2 + \beta\|\alpha\|_1$, where $H = \begin{bmatrix}(Xu_1)_+ & \dots & (Xu_N)_+\end{bmatrix} \in \mathbb{R}^{n\times N}$ and $\alpha = (\alpha_1, \dots, \alpha_N) \in \mathbb{R}^N$. The ISTA update is then $\alpha^+ = \texttt{prox}_{\gamma_r\beta\|\cdot\|_1}(\alpha - \gamma_r H^\top H\alpha + \gamma_r H^\top y)$, where $\texttt{prox}_{\gamma_r\beta\|\cdot\|_1}(\cdot)$ evaluates to $\operatorname{sgn}(\cdot)\max(|\cdot| - \gamma_r\beta, 0)$, α^+ denotes the updated α at each iteration, and γ_r is a step size that can be determined with backtracking line search. Therefore, the per-iteration complexity is $\mathcal{O}(N^2)$. Since N is linear in n for a fixed solution quality (cf. Theorem 4), the complexity amounts to $\mathcal{O}(n^2)$. In comparison, while back-propagation SGD's $\mathcal{O}(n^2 d)$ complexity seems worse than ISTA, each back-propagation SGD epoch is likely to be faster than an ISTA iteration in practice. This is because while N scales linearly in the number of data points n, the slope of this linear relationship can be very steep, and therefore N can be large if an accurate solution is desired.

Theorem 2 also sheds light on ANN training dynamics: for the purpose of approximating the training data, when the network is wide, the hidden layers are less important than the output layer. The role of the hidden layers is to map the data to features in higher-dimensional spaces, facilitating the output layer to extract the most important information. Comparatively, the convex formulations (Pilanci and Ergen, 2020, Equation 8) show that one-hidden-layer ANNs can be regarded as combinations of linear classifiers, where the mixed regularization terms promote group sparsity and discard suboptimal linear classifiers. Similarly, our SCP-based convex formulation shows that the output of a one-hidden-layer ANNs is a weighted average of features, where the ℓ_1 regularization term promotes sparsity and discards less informative features.

4.2 Iterative sampling of hidden-layer weights

While the efficacy of the SCP-based convex training formulation with a one-shot sampling of the hidden layer neurons can be proved theoretically and experimentally, the probabilistic optimality bound provided in Theorem 4 may be too conservative in some cases. To provide a more accurate and robust estimation of the level of suboptimality of the SCP relaxation (14), we propose a scheme (Algorithm 4) that iteratively samples hidden layer neurons used in (14) to train classifiers.

The convex semi-infinite training formulation (13) has a dual problem: (Pilanci and Ergen, 2020, Appendix A.4)

$$d^\star = \max_{v\in\mathbb{R}^n} -\ell^*(v) \quad \text{s.t.} \quad |v^\top (Xu)_+| \le \beta, \;\; \forall u : \|u\|_2 \le 1, \tag{16}$$

where $\ell^*(\cdot)$ is the Fenchel conjugate function defined as $\ell^*(v) = \max_z z^\top v - \ell(z, y)$. When $m \ge m^*$, where m^* is upper-bounded by $n+1$, strong duality holds $p^\star = d^\star$. Moreover, the dual problem (16) is a convex semi-infinite problem, which is a category of uncertain convex programs (UCP) (Calafiore and Campi, 2005).

We then use the sampled vectors $u_1, \dots, u_N$ to construct the following sampled convex program (SCP) that approximates the UCP (16):

$$d^\star_{s3} = \max_{v\in\mathbb{R}^n} -\ell^*(v) \quad \text{s.t.} \quad |v^\top (Xu_i)_+| \le \beta, \;\; \forall i \in [N]. \tag{17}$$

Similarly, strong duality holds between (17) and (14) and it holds that $p^\star_{s3} = d^\star_{s3}$. The level of suboptimality of the dual solution $v^\star$ to (17) can be easily verified by checking the feasibility of $v^\star$ to the UCP (16).

While it is easier to check the quality of the dual solution, it is desirable to solve the primal problem (14) because the primal is unconstrained and thus easier to solve. Suppose that $(\alpha_i^\star)_{i=1}^N$ is a solution to (14). By following the procedure described in Appendix C.4, one can recover the optimal dual variable $v^\star$ from $(\alpha_i^\star)_{i=1}^N$ by exploiting the strong duality between (14) and (17). Next, we independently sample another set of N_1 hidden layer weights $(u_i^1)_{i=1}^{N_1}$ via uniform distribution and check if $|v^{\star\top}(Xu_i^1)_+| > \beta$ for each $i \in [N_1]$. If $|v^{\star\top}(Xu_i^1)_+| > \beta$ for a particular i, then adding u_i^1 to the set of sampled constraint set of (17) will change (reduce) the value of $d^\star_{s3}$ and thereby reduce the relaxation gap between $p^\star_{s3}$ and $p^\star$. In other words, by incorporating u_i^1 as another hidden layer node, the considered ANN can be improved.

Define the notations

$$Z_i := \begin{cases} 1 & \text{if } |v^{\star\top}(Xu_i^1)_+| > \beta \\ 0 & \text{otherwise} \end{cases}, \text{ for all } \forall i \in [N_1],$$

$$\overline{Z} := \frac{\sum_{i=1}^{N_1} Z_i}{N_1}, \quad \text{and} \quad \theta := \mathbb{E}[Z_i] = \mathbb{P}_{u\sim \mathrm{Unif}(\mathcal{S}^{d-1})}\Big[|v^{\star\top}(Xu)_+| > \beta\Big].$$

where $\mathrm{Unif}(\mathcal{S}^{d-1})$ denotes the uniform distribution on a $(d-1)$-sphere.

Algorithm 4 Convex ANN training based on iterative sampling hidden-layer weights

Let $t = 0$; sample $\widehat{u}_1^0, \dots, \widehat{u}_{N_0}^0 \sim \mathcal{N}(0, I_d)$ i.i.d., and let $u_i^0 = \frac{\widehat{u}_i^0}{\|\widehat{u}_i^0\|_2}$ for all $i \in [N_0]$.
Construct $\mathcal{U}^0 := \{u_1^0, \dots, u_{N_0}^0\}$; let $U_0 = N_0$.
repeat
  Solve $(\alpha_i^t)_{i=1}^{U_t} = \arg\min_{(\alpha_i)_{i=1}^{U_t}} \ell\big(\sum_{i=1}^{U_t} \big(Xu_i^t\big)_+ \alpha_i, y\big) + \beta \sum_{i=1}^{U_t} |\alpha_i|$, the same formulation as (14).
  Update $v^t = y - \sum_{i=1}^{U_t} (Xu_i)_+ \alpha_i^t$.
  Sample $\widehat{u}_1^{t+1}, \dots, \widehat{u}_{N_{t+1}}^{t+1} \sim \mathcal{N}(0, I_d)$ i.i.d., and let $\bar{u}_i^{t+1} = \frac{\widehat{u}_i^{t+1}}{\|\widehat{u}_i^{t+1}\|_2}$ for all $i \in [N_{t+1}]$.
  Construct $\mathcal{E}^{t+1} = \big\{\bar{u}_i^{t+1} \;\big|\; |v^{t\top}(X\bar{u}_i^{t+1})_+| > \beta\big\}$ to be the set of newly sampled weight vectors that tighten the dual constraint.
  Construct $\mathcal{U}^{t+1} = \mathcal{U}^t \cup \mathcal{E}^{t+1}$ and rename all vectors in $\mathcal{U}^{t+1}$ as $u_1^{t+1}, \dots, u_{U_{t+1}}^{t+1}$, where U_{t+1} is the cardinality of $\mathcal{U}^{t+1}$.
  $t \leftarrow t + 1$.
until $\frac{|\mathcal{E}^t|}{N_t} + \frac{\log(1/\xi)}{2N_t} \le \psi$ or/and $U_{t-1} \ge \frac{n+1}{\psi\xi} - 1$, where ψ and ξ are preset thresholds.

By Hoeffding's inequality, it holds that $\mathbb{P}\big(\theta - \overline{Z} \ge t\big) \le \exp(-2N_1 t^2)$. Therefore, with probability at least $1 - \xi$, it holds that $\theta \le \overline{Z} + \frac{\log(1/\xi)}{2N_1}$, where $\xi \in (0, 1]$. In other words, by evaluating the feasibility of the additional set of hidden layer weights $u_1^1 \dots u_{N_1}^1$, one can obtain a probabilistic bound on the level of suboptimality of the solution to (17) constructed with $u_1 \dots u_N$: as long as $\overline{Z} + \frac{\log(1/\xi)}{2N_1} \le \psi$ for a constant $\psi \in (0, 1]$, it holds that $\theta \le \psi$ with probability at least $1 - \xi$.

We now introduce a scheme of training scalar-output fully-connected ReLU neural networks to an arbitrary degree of suboptimality by repeating the evaluation and sampling procedure, described in Algorithm 4. Let T denote the total iterations of Algorithm 4, U_t denote the total number of hidden layer neurons at iteration t, and N_t denote the number of hidden layer neurons sampled at iteration t. In light of Theorem 4, it holds that the solution $(\alpha_i^\star)_{i=1}^{U_T}$ yielded by Algorithm 4 satisfies the following property with probability at least $1 - \xi$: if an additional vector $\tilde{u}$ is sampled on the unit Euclidean norm sphere $\mathcal{S}^{d-1}$ via a uniform distribution, then adding $\tilde{u}$ to the set of hidden layer weights used in (14) will not improve the training loss of the neural network with probability at least $1 - \psi$.

5. Convex Adversarial Training

The inherent difficulties with adversarial training can be addressed by taking advantage of the convex training framework and the related algorithms.

5.1 Background about adversarial training

A classifier is considered robust against adversarial perturbations if it assigns the same label to all inputs within an ℓ_∞ bound with radius ϵ (Goodfellow et al., 2015). The perturbation

set can then be defined as

$$\mathcal{X} = \Big\{ X + \Delta \in \mathbb{R}^{n \times d} \;\Big|\; \Delta = [\delta_1, \dots, \delta_n]^\top,\ \delta_k \in \mathbb{R}^d,\ \|\delta_k\|_\infty \leq \epsilon,\ \forall k \in [n] \Big\}.$$

In this work, we consider the "white box" setting, where the adversary has complete knowledge about the neural network. As stated in (Madry et al., 2018), one common method for training robust classifiers is to minimize the maximum loss within the perturbation set by solving the following minimax problem:

$$\min_{(u_j, \alpha_j)_{j=1}^m} \left(\max_{\Delta : X + \Delta \in \mathcal{X}} \ell\Big(\sum_{j=1}^m \big((X+\Delta)u_j\big)_+ \alpha_j, y \Big) + \frac{\beta}{2} \sum_{j=1}^m \big(\|u_j\|_2^2 + \alpha_j^2 \big) \right) \tag{18}$$

This process of "training with adversarial data" is often referred to as "adversarial training", as opposed to "standard training" that trains on clean data. In the prior literature, Fast Gradient Sign Method (FGSM) and Projected Gradient Descent (PGD) are commonly used to numerically solve the inner maximization of (18) and generate adversarial examples in practice (Madry et al., 2018). More specifically, FGSM generates adversarial examples $\tilde{x}$ using

$$\tilde{x} = x + \epsilon \cdot \operatorname{sgn}\Big(\nabla_x \ell\big(\sum_{j=1}^m (x^\top u_j)_+ \alpha_j, y \big) \Big). \tag{19}$$

Since FGSM is a one-shot method that assumes linearity, it may miss the worst-case adversarial input. PGD better explores the nonlinear landscape of the problem and is capable of generating "universal" first-order adversaries by running the iterations

$$\tilde{x}^{t+1} = \Pi_{\mathcal{X}} \left(\tilde{x}^t + \gamma_p \cdot \operatorname{sgn}\Big(\nabla_x \ell\big(\sum_{j=1}^m (x^\top u_j)_+ \alpha_j, y \big) \Big) \right) \tag{20}$$

for $t = 0, 1, \dots$, where x^t is the perturbed data vector at the t^{th} iteration, $\Pi_{\mathcal{X}}$ denotes the projection onto the perturbation set $\mathcal{X}$, and $\gamma_p > 0$ is the step size. The initial vector $\tilde{x}^0$ is the unperturbed data x.

5.2 The convex adversarial training formulation

While PGD adversaries have been considered "universal" in the literature, adversarial training with PGD adversaries has several limitations. Since the optimization landscapes of ANNs are generally non-concave over Δ, there is no guarantee that PGD will find the true worst-case adversary within the perturbation bound. Furthermore, traditional adversarial training algorithms solve complex bi-level minimax optimization problems, exacerbating the instability issue of non-convex ANN training. Our experiments show that back-propagation gradient methods can struggle to solve (18) and can be highly sensitive to initializations. Moreover, iteratively solving the bi-level optimization (18) requires an algorithm with a nested loop structure, which is computationally cumbersome. To conquer such difficulties, we leverage Theorem 1 to re-characterize (18) as robust, convex upper-bound problems that can be efficiently solved globally.

We first develop a result about adversarial training involving general convex loss functions. The connection between the convex training objective and the non-convex neural network loss function holds only when the linear constraints in (2) are satisfied. For adversarial training, we need this connection to hold at all perturbed data matrices $X + \Delta \in \mathcal{X}$. Otherwise, if some matrix $X + \Delta$ violates the linear constraints, then this perturbation Δ can correspond to a low convex objective value but a high actual loss. To ensure the correctness of the convex reformulation throughout $\mathcal{X}$, we introduce some robust constraints below.

Since the D_i matrices in (2) reflects the ReLU patterns of X, these matrices can change when X is perturbed. Therefore, we include all distinct diagonal matrices $\mathrm{diag}([(X+\Delta)u \geq 0])$ that can be obtained for all $u \in \mathbb{R}^d$ and *all* $\Delta : X + \Delta \in \mathcal{U}$, denoted as $D_1, \ldots, D_{\widehat{P}}$, where $\widehat{P}$ is the total number of such matrices. Since $D_1, \ldots, D_{\widehat{P}}$ include $D_1, \ldots, D_P$ in (2), we have $\widehat{P} \geq P$. While $\widehat{P}$ is at most 2^n in the worst case, since ϵ is often small, we expect $\widehat{P}$ to be relatively close to P, where $P \leq 2r\big(\frac{e(n-1)}{r}\big)^r$ as discussed above.

Finally, we replace the objective of the convex standard training formulation (2) with its robust counterpart, giving rise to the optimization

$$\min_{(v_i,w_i)_{i=1}^{\widehat{P}}} \left(\max_{\Delta: X+\Delta \in \mathcal{U}} \ell\Big(\sum_{i=1}^{\widehat{P}} D_i(X+\Delta)(v_i - w_i), y \Big) + \beta \sum_{i=1}^{\widehat{P}} \big(\|v_i\|_2 + \|w_i\|_2 \big) \right) \tag{21a}$$

$$\text{s.t.} \quad \min_{\Delta: X+\Delta \in \mathcal{U}} (2D_i - I_n)(X+\Delta)v_i \geq 0, \quad \min_{\Delta: X+\Delta \in \mathcal{U}} (2D_i - I_n)(X+\Delta)w_i \geq 0, \quad \forall i \in [\widehat{P}] \tag{21b}$$

where $\mathcal{U}$ is any convex additive perturbation set. The next theorem shows that (21) is an upper-bound to the robust loss function (18), with the proof provided in Appendix C.5.

Theorem 5 *Let $(v^\star_{rob_i}, w^\star_{rob_i})_{i=1}^{\widehat{P}}$ denote a solution of (21) and define $\widehat{m}^\star$ as $|\{i : v^\star_{rob_i} \neq 0\}| + |\{i : w^\star_{rob_i} \neq 0\}|$. When the neural network width m satisfies $m \geq \widehat{m}^\star$, the optimization problem (21) provides an upper-bound on the non-convex adversarial training problem (18). The robust neural network weights $(u^\star_{rob_j}, \alpha^\star_{rob_j})_{j=1}^{\widehat{m}}$ can be recovered using (4).*

When the perturbation set is zero, Theorem 5 reduces to Theorem 1. In light of Theorem 5, we use optimization (21) as a surrogate for the optimization (18) to train the neural network. We will show that the new problem can be efficiently solved in important cases. By the analogy to Theorem 2, an approximation to (21) can be applied to train neural networks with width much less than $\widehat{m}^\star$. Since (21) includes all D_i matrices in (2), we have $\widehat{P} \geq P$. While $\widehat{P}$ is at most 2^n in the worst case, since ϵ is often small, we expect $\widehat{P}$ to be relatively close to P, where $P \leq 2r\big(\frac{e(n-1)}{r}\big)^r$ as discussed above.

The robust constraints in (21b) force all points within the perturbation set to be feasible. Intuitively, for every $j \in [\widehat{m}^\star]$, (21b) forces the ReLU activation pattern $\mathrm{sgn}\big((X+\Delta)u^\star_{\mathrm{rob}_j}\big)$ to stay the same for all Δ such that $X + \Delta \in \mathcal{U}$. Moreover, if $\Delta^\star_{\mathrm{rob}}$ denotes a solution to the inner maximization in (21a), then $X + \Delta^\star_{\mathrm{rob}}$ corresponds to the worst-case adversarial inputs for the recovered neural network.

Corollary 6 *For the perturbation set $\mathcal{X}$, the constraints in (21b) can be equivalently replaced by*

$$(2D_i - I_n)Xv_i \geq \epsilon\|v_i\|_1, \quad (2D_i - I_n)Xw_i \geq \epsilon\|w_i\|_1, \quad \forall i \in [\widehat{P}]. \tag{22}$$

The proof of Corollary 6 is provided in Appendix C.6. Note that the left side of each inequality in (22) is a vector while the right side is a scalar, which means that each element of the corresponding vector should be greater than or equal to that scalar.

5.3 Practical algorithm for convex adversarial training

Since Theorem 2 does not rely on assumptions about the matrix X, it applies to an arbitrary $X + \Delta$ matrix, and naturally extends to the convex adversarial training formulation (21). Therefore, an approximation to (21) can be applied to train robust neural networks with widths much less than $\widehat{m}^\star$. Similar to the strategy rendered in Algorithm 1, we use a subset of the D_i matrices for practical adversarial training. Since the D_i matrices depend on the perturbation Δ, we also add randomness to the data matrix X in the sampling process to cover D_i matrices associated with different perturbations, leading to Algorithm 5. P_a and S are preset parameters that determine the number of random weight samples, with $P_a \times S \geq P_s$.

Algorithm 5 Practical convex adversarial training

for $h = 1$ to P_a **do**
  $a_h \sim \mathcal{N}(0, I_d)$ i.i.d.
  $D_{h1} \leftarrow \mathrm{diag}([Xa_h \geq 0])$
  for $j = 2$ to S **do**
    $R_{hj} \leftarrow [r_1, \ldots, r_d]$, where $r_\kappa \sim \mathcal{N}(\mathbf{0}, I_n), \forall \kappa \in [d]$
    $D_{hj} \leftarrow \mathrm{diag}([\overline{X}_{hj} a_h \geq 0])$, where $\overline{X}_{hj} \leftarrow X + \epsilon \cdot \mathrm{sgn}(R_{hj})$
    Discard repeated D_{hj} matrices
    break if P_s distinct D_{hj} matrices has been generated
  end for
end for
Solve

$$\begin{aligned} \min_{(v_i, w_i)_{i=1}^{\widehat{P}}} \; & \left(\max_{\Delta: X+\Delta \in \mathcal{U}} \ell\Big(\sum_{h=1}^{P_s} D_h(X+\Delta)(v_h - w_h), y \Big) + \beta \sum_{h=1}^{P_s} \big(\|v_h\|_2 + \|w_h\|_2 \big) \right) \\ \text{s.t.} \; & \min_{\Delta: X+\Delta \in \mathcal{U}} (2D_h - I_n)(X+\Delta)v_h \geq 0, \quad \forall h \in [P_s], \\ & \min_{\Delta: X+\Delta \in \mathcal{U}} (2D_h - I_n)(X+\Delta)w_h \geq 0, \quad \forall h \in [P_s]. \end{aligned} \tag{23}$$

Recover $u_1, \ldots, u_{m_s}$ and $\alpha_1, \ldots, \alpha_{m_s}$ from the solution $(v^\star_{\mathrm{robs}_h}, w^\star_{\mathrm{robs}_h})_{h=1}^{P_s}$ of (23) using (4).

5.4 Convex hinge loss adversarial training

While the inner maximization of the robust problem (21) is still hard to solve in general, it is tractable for some loss functions. The simplest case is the piecewise-linear hinge loss $\ell(\widehat{y}, y) = (1 - \widehat{y} \odot y)_+$, which is widely used for classification. Here, we focus on binary classification with $y \in \{-1, 1\}^n$. [1]

Consider the training problem for a one-hidden-layer ANN with ℓ_2 regularized hinge loss:

$$\min_{(u_j,\alpha_j)_{j=1}^m} \left(\frac{1}{n} \cdot \mathbf{1}^\top \Big(\mathbf{1} - y \odot \sum_{j=1}^m (Xu_j)_+ \alpha_j \Big)_+ + \frac{\beta}{2} \sum_{j=1}^m \left(\|u_j\|_2^2 + \alpha_j^2 \right) \right) \tag{24}$$

The adversarial training problem considering the ℓ_∞-bounded adversarial data perturbation set $\mathcal{X}$ is:

$$\min_{(u_j,\alpha_j)_{j=1}^m} \left(\max_{\Delta : X+\Delta \in \mathcal{X}} \frac{1}{n} \cdot \mathbf{1}^\top \Big(\mathbf{1} - y \odot \sum_{j=1}^m \big((X+\Delta) u_j \big)_+ \alpha_j \Big)_+ + \frac{\beta}{2} \sum_{j=1}^m \left(\|u_j\|_2^2 + \alpha_j^2 \right) \right) \tag{25}$$

Applying Theorem 5 and Corollary 6 leads to the following formulation as an upper bound on (25):

$$\begin{aligned} \min_{(v_i,w_i)_{i=1}^{\widehat{P}}} & \left(\max_{\Delta : X+\Delta \in \mathcal{X}} \frac{1}{n} \cdot \mathbf{1}^\top \Big(\mathbf{1} - y \odot \sum_{i=1}^{\widehat{P}} D_i (X+\Delta)(v_i - w_i) \Big)_+ + \beta \sum_{i=1}^{\widehat{P}} \left(\|v_i\|_2 + \|w_i\|_2 \right) \right) \\ \text{s.t. } & (2D_i - I_n) X v_i \geq \epsilon \|v_i\|_1, \;\; (2D_i - I_n) X w_i \geq \epsilon \|w_i\|_1, \;\; \forall i \in [\widehat{P}] \end{aligned} \tag{26}$$

For the purpose of generating the $D_1, \ldots, D_{\widehat{P}}$ matrices, instead of enumerating an infinite number of points in $\mathcal{X}$, we only need to enumerate all vertices of $\mathcal{X}$, which is finite. This is because the solution $\Delta^\star_{\text{hinge}}$ to the inner maximum always occurs at a vertex of $\mathcal{X}$, as will be shown in Theorem 7. Solving the inner maximization of (26) in closed form leads to the next theorem, whose proof is provided in Section C.7.

Theorem 7 *For the binary classification problem, the inner maximum of (26) is attained at* $\Delta^\star_{hinge} = -\epsilon \cdot sgn\Big(\sum_{i=1}^{\widehat{P}} D_i y (v_i - w_i)^\top \Big)$, *and the bi-level optimization problem (26) is equivalent to the classic optimization problem:*

$$\begin{aligned} \min_{(v_i,w_i)_{i=1}^{\widehat{P}}} & \left(\frac{1}{n} \sum_{k=1}^n \left(1 - y_k \sum_{i=1}^{\widehat{P}} d_{ik} x_k^\top (v_i - w_i) + \epsilon \left\| \sum_{i=1}^{\widehat{P}} d_{ik} (v_i - w_i) \right\|_1 \right)_+ \right. \\ & \qquad \left. + \beta \textstyle\sum_{i=1}^{\widehat{P}} \left(\|v_i\|_2 + \|w_i\|_2 \right) \right) \\ \text{s.t.} \quad & (2D_i - I_n) X v_i \geq \epsilon \|v_i\|_1, \;\; (2D_i - I_n) X w_i \geq \epsilon \|w_i\|_1, \;\; \forall i \in [\widehat{P}] \end{aligned} \tag{27}$$

where d_{ik} *denotes the* k^{th} *diagonal element of* D_i.

1. Other ℓ_p norm-bounded additive perturbation sets can be similarly analyzed, as shown in Appendix B.3. It is also straightforward to extend the analysis in this section to any convex piecewise-affine loss functions.

The problem (27) is a finite-dimensional convex program that provides an upper bound on (25), which can be considered as the robust counterpart of (24). We can thus solve (27) to robustly train the neural network. The ℓ_1 norm term in (27) explains the regularization effect of adversarial training.

5.5 Convex squared loss adversarial training

As discussed before, the squared loss $\ell(\widehat{y}, y) = \frac{1}{2}\|\widehat{y} - y\|_2^2$ is another commonly used loss function in machine learning. Consider the non-convex training problem of a one-hidden-layer ReLU ANN trained with the ℓ_2-regularized squared loss:

$$\min_{(u_j,\alpha_j)_{j=1}^m} \frac{1}{2}\left\| \sum_{j=1}^m (Xu_j)_+\alpha_j - y \right\|_2^2 + \frac{\beta}{2}\sum_{j=1}^m \left(\|u_j\|_2^2 + \alpha_j^2\right). \tag{28}$$

Coupling this nominal problem with the perturbation set $\mathcal{X}$ gives us the robust counterpart of (28) as

$$\min_{(u_j,\alpha_j)_{j=1}^m} \left(\max_{\Delta: X+\Delta\in\mathcal{X}} \frac{1}{2}\left\| \sum_{j=1}^m \left((X+\Delta)u_j\right)_+\alpha_j - y \right\|_2^2 + \frac{\beta}{2}\sum_{j=1}^m \left(\|u_j\|_2^2 + \alpha_j^2\right) \right). \tag{29}$$

Applying Theorem 5 and Corollary 6 leads to the following formulation as an upper bound on (29):

$$\begin{aligned} \min_{(v_i,w_i)_{i=1}^{\widehat{P}}} & \left(\max_{\Delta: X+\Delta\in\mathcal{X}} \frac{1}{2}\left\| \sum_{i=1}^{\widehat{P}} D_i(X+\Delta)(v_i - w_i) - y \right\|_2^2 + \beta \sum_{i=1}^{\widehat{P}} \left(\|v_i\|_2 + \|w_i\|_2\right) \right) \\ \text{s.t.} \quad & (2D_i - I_n)Xv_i \geq \epsilon\|v_i\|_1, \quad (2D_i - I_n)Xw_i \geq \epsilon\|w_i\|_1, \quad \forall i \in [\widehat{P}]. \end{aligned} \tag{30}$$

Solving the maximization over Δ in closed form leads to the next result, with the proof provided in Appendix C.8.

Theorem 8 *The optimization problem (30) is equivalent to the convex program:*

$$\begin{aligned} \min_{(v_i,w_i)_{i=1}^{\widehat{P}},a,z} \quad & a + \beta\sum_{i=1}^{\widehat{P}}(\|v_i\|_2 + \|w_i\|_2) \\ \text{s.t.} \quad & (2D_i - I_n)Xv_i \geq \epsilon\|v_i\|_1, \quad (2D_i - I_n)Xw_i \geq \epsilon\|w_i\|_1, \quad \forall i \in [\widehat{P}] \\ & z_k \geq \left| \sum_{i=1}^{\widehat{P}} D_{ik}x_k^\top(v_i - w_i) - y_k \right| + \epsilon\left\| \sum_{i=1}^{\widehat{P}} D_{ik}(v_i - w_i) \right\|_1, \quad \forall k \in [n] \\ & z_{n+1} \geq \left|2a - \tfrac{1}{4}\right|, \ \ \|z\|_2 \leq 2a + \tfrac{1}{4}. \end{aligned} \tag{31}$$

Problem (31) is a convex optimization that can train robust neural networks. However, directly using (31) for adversarial training can be intractable due to the large number of

constraints that arise when we include all D_i matrices associated with all Δ such that $X+\Delta \in \mathcal{X}$. To this end, one can use the approximation in Algorithm 5 and sample a subset of the diagonal matrices $D_1, \ldots, D_{P_s}$. As before, the optimality gap can be characterized with Theorem 2.

5.6 Convex binary cross-entropy loss adversarial training

The binary cross-entropy loss is also widely used in binary classification. Here, we consider a scalar-output ANN with a scaled tanh output layer for binary classification with $y \in \{0,1\}^n$. The loss function $\ell(\cdot)$ in this case is $\ell(\widehat{y}, y) = -2\widehat{y}^\top y + \mathbf{1}^\top \log(e^{2\widehat{y}}+1)$, with the detailed derivation shown in Appendix 6.2.4.

The non-convex adversarial training formulation considering the ℓ_∞-bounded adversarial data uncertainty $\mathcal{X}$ is then:

$$\begin{aligned} \min_{(u_j,\alpha_j)_{j=1}^m} \ & \left(\max_{\|\Delta\|_{\max}\le\epsilon} \frac{1}{n}\sum_{k=1}^n \Big(-2\widehat{y}_k y_k + \log(e^{2\widehat{y}_k}+1)\Big)\right) + \frac{\beta}{2}\sum_{j=1}^m \big(\|u_j\|_2^2 + \alpha_j^2\big) \\ \text{s.t.} \ & \widehat{y} = \sum_{j=1}^m \big((X+\Delta)u_j\big)_+ \alpha_j. \end{aligned} \tag{32}$$

Applying Theorem 5 and Corollary 6 leads to the following optimization as an upper bound on (32):

$$\begin{aligned} \min_{(v_i,w_i)_{i=1}^{\widehat{P}}} \ & \left(\max_{\|\Delta\|_{\max}\le\epsilon} \frac{1}{n}\sum_{k=1}^n \Big(-2\widehat{y}_k y_k + \log(e^{2\widehat{y}_k}+1)\Big)\right) + \beta\sum_{i=1}^{\widehat{P}} \Big(\|v_i\|_2 + \|w_i\|_2\Big) \\ \text{s.t.} \ & (2D_i - I_n)Xv_i \ge \epsilon\|v_i\|_1,\ (2D_i-I_n)Xw_i \ge \epsilon\|w_i\|_1, \quad \forall i \in [\widehat{P}], \\ & \widehat{y}_k = \sum_{i=1}^{\widehat{P}} d_{ik}x_k^\top(v_i - w_i) + \sum_{i=1}^{\widehat{P}} d_{ik}\delta_k^\top(v_i-w_i). \end{aligned} \tag{33}$$

Consider the formulation

$$\begin{aligned} \min_{(v_i,w_i)_{i=1}^{\widehat{P}}} \ & \frac{1}{n}\Big(\sum_{k=1}^n f\circ g_k\big(\{v_i,w_i\}_{i=1}^{\widehat{P}}\big)\Big) + \beta\sum_{i=1}^{\widehat{P}}\Big(\|v_i\|_2+\|w_i\|_2\Big) \\ \text{s.t.} \ & (2D_i-I_n)Xv_i \ge \epsilon\|v_i\|_1,\ (2D_i-I_n)Xw_i\ge\epsilon\|w_i\|_1, \quad \forall i\in[\widehat{P}] \\ & f(u) = \log(e^{2u}+1), \\ & g_k\big(\{v_i,w_i\}_{i=1}^{\widehat{P}}\big) = (2y_k-1)\sum_{i=1}^{\widehat{P}} d_{ik}x_k^\top(v_i-w_i) + \epsilon\cdot\Big\|\sum_{i=1}^{\widehat{P}} d_{ik}(v_i-w_i)\Big\|_1, \ \ \forall k\in[n]. \end{aligned} \tag{34}$$

The next theorem establishes the equivalence between (34) and (33). The proof is provided in Appendix C.9.

Theorem 9 *The optimization (34) is a convex program that is equivalent to the bi-level optimization (33), and can be used as a surrogate for (32) to train robust neural networks. The worst-case perturbation is* $\Delta^\star_{BCE} = -\epsilon \cdot sgn\Big((2y-1)\sum_{i=1}^{\widehat{P}} D_i(v_i - w_i)^\top\Big)$.

Note that the worst-case perturbation occurs at the same location as for the hinge loss case, which is a vertex in $\mathcal{X}$. Thus, for the purpose of generating the $D_1, \ldots, D_{\widehat{P}}$ matrices, we again only need to enumerate all vertices of $\mathcal{X}$ instead of all points in $\mathcal{X}$.

5.7 More complex ANN structures

While our discussions explicitly focus on one-hidden-layer scalar-output ReLU networks, the derived training methods can be used for more sophisticated ANN architectures. As discussed before, greedily training one-hidden-layer ANNs leads to a well-performing deep network (Belilovsky et al., 2019). Leveraging recent works that reform the training of more complex ANNs into convex programs (cite), our analysis can also extend to those ANNs because most convex training formulations share similar structures. Specifically, the convex training formulations rely on binary matrices to represent ReLU activation patterns and rely on convex (and often linear) constraints to enforce the patterns, with different regularizations revealing the sparse properties of different architectures. As an example, in Appendix B.1, we extend our convex adversarial training analysis to various CNN formulations used in (Ergen and Pilanci, 2021a). In Appendix B.2, we extend the SCP-based convex training algorithm to vector-output networks. Coupling layer-wise training and SCP convex training recovers multi-layer ELMs.

6. Numerical Experiments

6.1 Approximated convex standard training

In this subsection, we use numerical experiments to demonstrate the efficacy of practical standard training (Algorithm 1) and to show the level of suboptimality of the neural network trained using Algorithm 1.[2] The experiment was performed on a randomly-generated dataset with $n = 40$ and $d = 2$. The upper bound on the number of ReLU activation patterns is $4\big(\frac{e(39)}{2}\big)^2 = 11239$. We ran Algorithm 1 to train neural networks using the hinge loss with the number of D_h matrices equal to $4, 8, 16, \ldots, 2048$ and compared the optimized loss. We repeated this experiment 15 times for each setting, and plotted the loss in Figure 1.[3] The error bars show the loss values achieved in the best and the worst runs. When there are more than 128 matrices (much less than the theoretical bound on P), Algorithm 1 yields

2. For all experiments in this paper, CVX (Grant and Boyd, 2014) and CVXPY (Agrawal et al., 2018; Diamond and Boyd, 2016) with the MOSEK (ApS, 2019) solver was used for solving optimization on a MacBook Pro laptop computer, unless otherwise stated. Off-the-shelf solvers supported by CVX and CVXPY often treat the convex training problem as a general SOCP. Among all solvers that we experimented on the convex training formulation, MOSEK is the most efficient.
3. To reliably sample P_s matrices, $P_a \cdot S$ in Algorithm 5 was set to a large number (81920), and the sampling was terminated when a sufficient number of D_h matrices was generated. The regularization strength β was chosen as 10^{-4}.

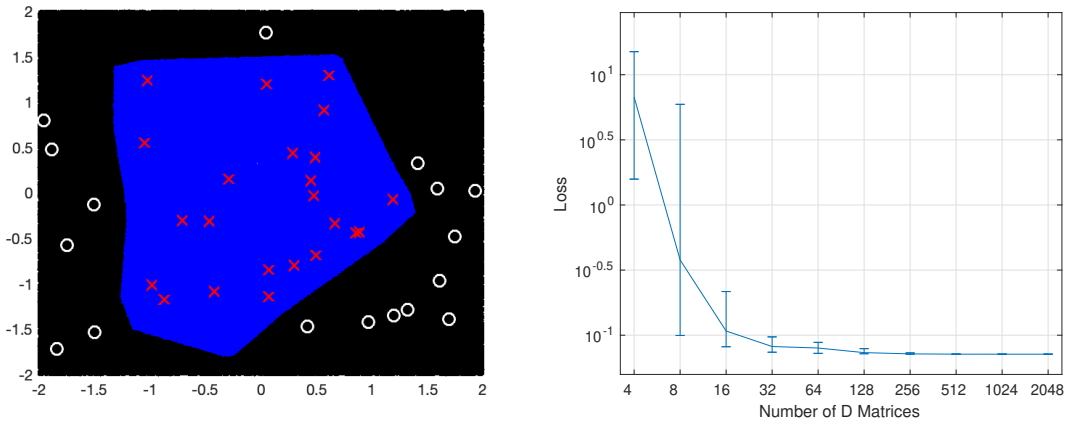

Figure 1: **FIXIT**The left figure is a randomized 2-dimensional dataset. The right figure is the optimized training loss for each P_s. The red crosses are positive training points and the white circles are negative training points. The region classified as positive is in blue, whereas the negative region is in black. When P_s reaches 128, the mean and variance of the optimized loss become very small.

consistent and favorable results. Further increasing the number of D matrices does not produce a significantly lower loss. By Theorem 2, $P_s = 128$ corresponds to $\psi\xi = 0.318$.

6.2 The ADMM convex training algorithm

In this section, we present the experiment results of the ADMM training algorithm (Algorithm 2). For the best efficiency, throughout this section, we use Algorithm 2 to solve the approximate convex training formulation (5) with the sampled D_h matrices.

6.2.1 Squared loss (closed form u updates) – convergence

For the case of the squared loss, the closed-form solution (12) is used for the u updates. We first demonstrate the convergence of the proposed ADMM algorithm using contrived random data with dimensions $n = 6, d = 5, P_s = 8$. CVX (Grant and Boyd, 2014) with the IPM-based MOSEK solver (ApS, 2019) was used to solve the optimal objective of (2) as the ground truth.

Figure 2 demonstrates the behavior of the ADMM algorithm when it converge to the global optimum of (2). Before discussing the results, we first explain the notations used in this figure. The CVX optimal objective is denoted as $l^\star_{\text{CVX}}$. Similarly, we use $l^\star_{\text{ADMM}}$ to denote the objective that ADMM converges to as the number of iterations k goes to infinity. Note that there are several methods to calculate the training cost obtained by ADMM. For fair comparisons among ADMM, CVX, and back-propagation, we use (4) to recover the neural network weights $(u_j, \alpha_j)_{j=1}^m$ from the ADMM optimization variables $(v_h^k, w_h^k)_{h=1}^{P_s}$, and use $(u_j, \alpha_j)_{j=1}^m$ to calculate the true non-convex training loss (1). The loss at each

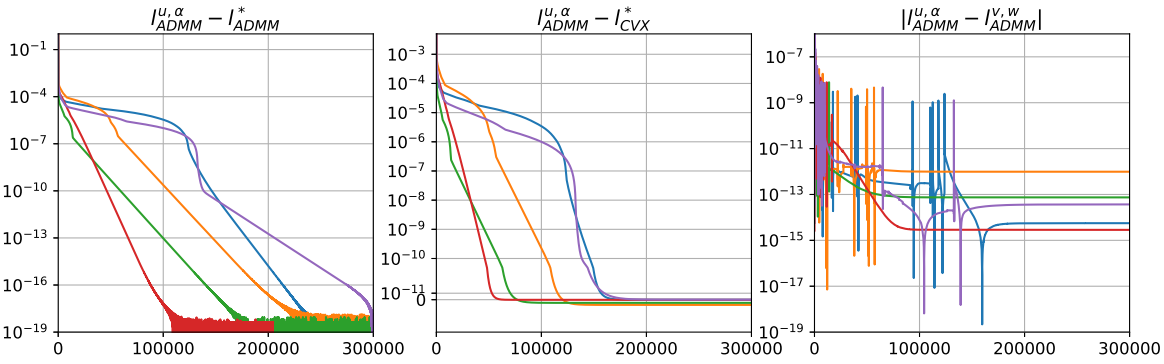

Figure 2: Gap between the cost returned by ADMM at each iteration and the true optimal cost for five independent runs.
Left: $l^{u,\alpha}_{\text{ADMM}} - l^{\star}_{\text{ADMM}}$; middle: $l^{u,\alpha}_{\text{ADMM}} - l^{\star}_{\text{CVX}}$; right: $\left|l^{u,\alpha}_{\text{ADMM}} - l^{v,w}_{\text{ADMM}}\right|$. **FIXIT**

iteration calculated via this method is denoted as $l^{u,\alpha}_{\text{ADMM}}$. The ADMM solution $l^{\star}_{\text{ADMM}}$ is also calculated via this method. Throughout the optimization process, we also directly calculate the convex objective of (2) using $(v_h^k, w_h^k)_{h=1}^{P_s}$. The loss at each iteration calculated via this method is denoted as $l^{v,w}_{\text{ADMM}}$. When the constraints of (2) are satisfied, it holds that $l^{u,\alpha}_{\text{ADMM}} = l^{v,w}_{\text{ADMM}}$. When some of the constraints are violated, then $l^{u,\alpha}_{\text{ADMM}}$ may be different from $l^{v,w}_{\text{ADMM}}$. Since ADMM uses dual variables to enforce the constraints, the sequence of optimization variables generated by Algorithm 2 may not always satisfy the constraints (the ADMM solution is feasible, but the intermediate iterations may not be feasible). For this reason, the gap between $l^{u,\alpha}_{\text{ADMM}}$ and $l^{v,w}_{\text{ADMM}}$ indirectly characterizes the feasibility of the ADMM intermediate solutions. When this gap is small, $(v_h^k, w_h^k)_{h=1}^{P_s}$ should be almost feasible. When this gap is large, the constraints may have been severely violated.

FIXITThe left plot of Figure 2 shows that the ADMM training loss converges to a stationary value at a linear rate, verifying the findings of Theorem 3. **FIXIT**The middle plot shows that ADMM converges towards the CVX ground truth, verifying the correctness of the ADMM solution. Note that $l^{\star}_{\text{ADMM}}$ is often slightly (on the order of 10^{-12}) lower than $l^{\star}_{\text{CVX}}$. This discrepancy between the two solutions is likely due to the inherent inaccuracy of the interior point method used by CVX: the IPM reforms the constraints as log barrier functions, resulting in strictly feasible (overly conservative) trajectories and objectives that are slightly higher than the true optimal. We observe that relaxing the accuracy setting of CVX increases this optimality gap, confirming this reasoning. **FIXIT**The right plot of Figure 2 shows that $l^{v,w}_{\text{ADMM}}$ and $l^{u,\alpha}_{\text{ADMM}}$ are close throughout the ADMM iterations, implying that v_i and w_i violate the constraints of (2) insignificantly at every step. Together, these figures confirm that the ADMM algorithm optimizes (1) effectively as designed.

While it can take about **FIXIT**3×10^5 iterations for ADMM to converge to machine precision, an approximate solution is usually sufficient for ANN training. As shown in Figure 3, which is a zoomed-in version of Figure 2, an accuracy of 10^{-3} can be achieved within 25 iterations. Moreover, **FIXIT**the right figure shows that the solution after 25 iterations violates the constraints insignificantly. This behavior of "converging rapidly in the first several steps and

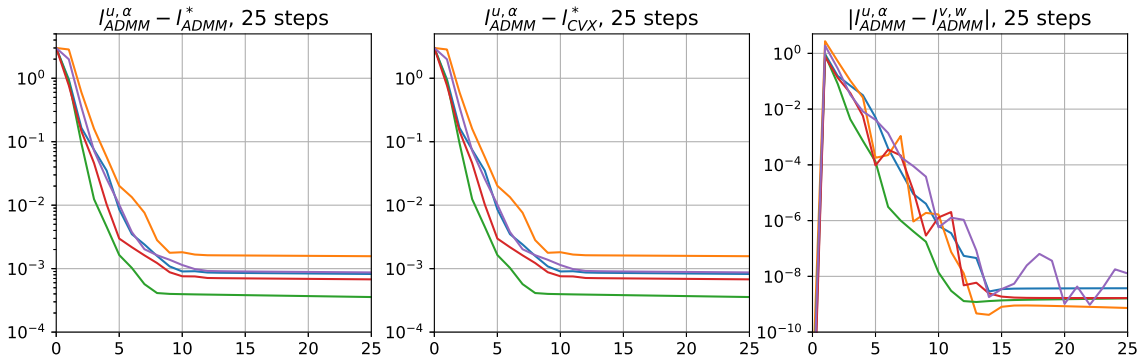

Figure 3: Gap between the cost returned by ADMM for the first 25 iterations and the true optimal cost for the five independent runs. **FIXIT**Left: $l^{u,\alpha}_{\text{ADMM}} - l^{\star}_{\text{ADMM}}$; middle: $l^{u,\alpha}_{\text{ADMM}} - l^{\star}_{\text{CVX}}$; right: $|l^{u,\alpha}_{\text{ADMM}} - l^{v,w}_{\text{ADMM}}|$.

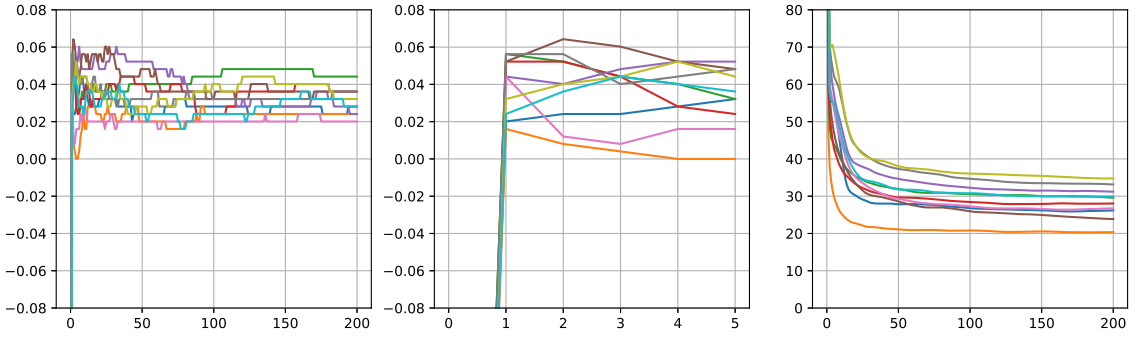

Figure 4: **FIXIT**Left: $\text{Accuracy}_{\text{ADMM}} - \text{Accuracy}_{\text{CVX}}$ (positive means the ADMM solution outperforms the CVX solution); middle: zoomed in to the first five iterations; right: $l^{u,\alpha}_{\text{ADMM}} - l^{\star}_{\text{CVX}}$. Ten independent runs are shown.

slowing down (to a linear rate) afterward" is typical for the ADMM algorithm. As will be shown next, a medium-accuracy solution returned by only running a few ADMM iterations can achieve a better prediction performance than the CVX solution.

To visualize how the prediction performance achieved by the model changes as the ADMM iteration progresses, we ran the ADMM iterations on the "mammographic masses" dataset from the UCI Machine Learning Repository (Dua and Graff, 2017), and recorded the prediction accuracy on the test set at each iteration. The exact global optimum of (2) was found via CVX as a baseline. 70% of the dataset was randomly selected as the training set, and the other 30% was used as the test set. Figure 4 plots the difference between the ADMM accuracies and the CVX accuracies at each iteration. Note that all v_i and w_i are initialized to be zero, and therefore the initial accuracy at the zeroth iteration is zero.

All ten runs achieved superior test accuracies throughout the first 200 iterations compared with the CVX baseline, and even the first five iterations outperformed the baseline, with the best run outperforming the baseline by 6%. After about 80 iterations, the accuracies stabilize

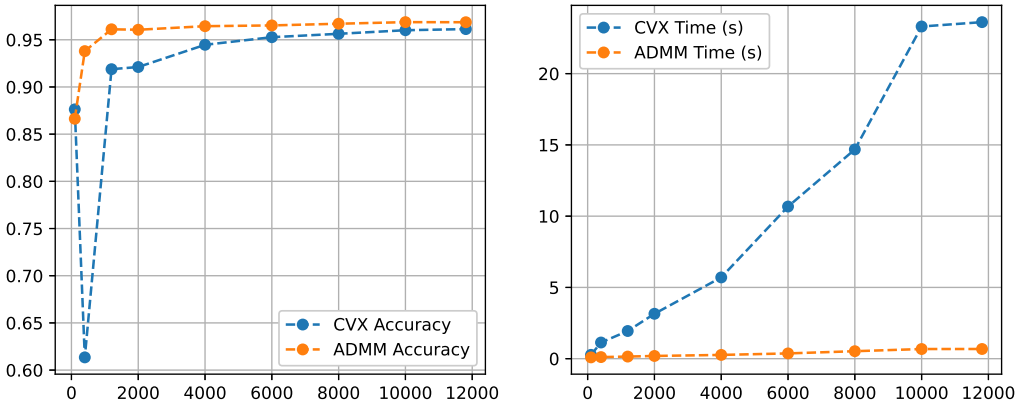

Figure 5: **FIXIT**Left: average test accuracy for each n; right: average CPU time for each n.

at around 2% to 4% better than the CVX baseline. On the other hand, the optimality gap between the ADMM solution after 20 iterations and the global optimum is around 30. Specifically, the average true optimal objective over the ten runs is 96.63, and the average ADMM objective is 125.0. In conclusion, the prediction performance of the classifiers trained by ADMM is superior even when only a few iterations are run, and an approximate solution with a slightly higher training cost may perform better.

6.2.2 Squared loss (closed form u updates) – complexity

To demonstrate the computational complexity of the proposed ADMM method, we used the ADMM method to train ANNs on a downsampled version of the MNIST handwritten digits database with $d = 100$. The task was to perform binary classification between digits "2" and "8". We first fixed $P_s = 8$ and varied n from 100 to 11809, the total number of 2's and 8's in the training set. The experiment was independently repeated five times for each n setting, and the results were averaged and shown in Figure 5. ADMM is allowed to run six iterations for each run. For all choices of n except $n = 100$, the networks trained with ADMM attained higher accuracies than those trained with CVX. More importantly, the CPU time required for CVX grows much faster than ADMM's execution time as n increases, verifying that the ADMM execution time increases linearly over n.

Similarly, we fixed $n = 1000$ and varied P_s from 4 to 50. The averaged results are shown in Figure 6. Once again, the proposed ADMM algorithm achieved a higher accuracy for each P_s, and the average CPU time required for ADMM grows much slower than the CVX CPU time. Figure 6 also shows that the CPU time scales quadratically with P_s, which supports our claim of an $\mathcal{O}(nP_s + d^2P_s^2)$ overall per-iteration complexity.

6.2.3 Squared loss (closed form u updates) – MNIST prediction

Finally, we demonstrate the effectiveness of the proposed ADMM algorithm on the full MNIST dataset (still for binary classification, but "full" in the sense of including all images

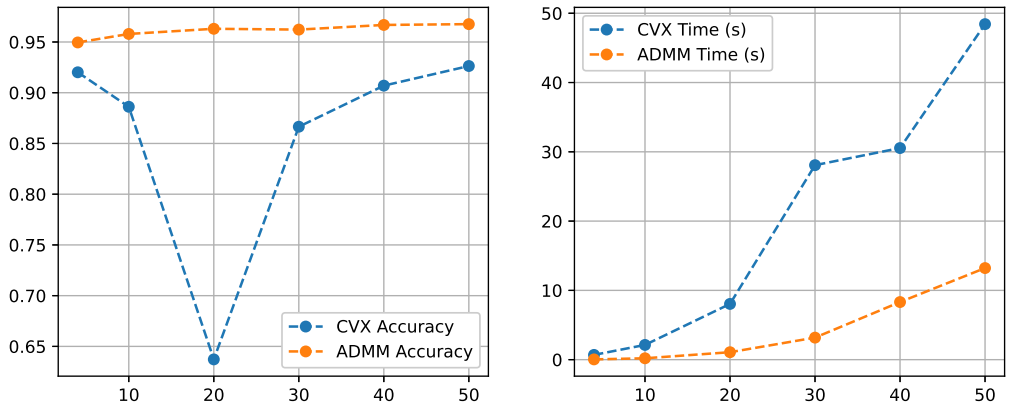

Figure 6: Left: average test accuracy for each P_s; right: average CPU time for each P_s.

Method	Accuracy	CPU Time (s)	Training Cost	Global Convergence
Back-propagation	98.86 %	74.09	422.4	No
CVX	70.99 %	14879	1.146	Yes
ADMM	98.90 %	802.2	223.2	Yes

Table 2: Average test accuracy, training cost, and CPU time achieved with the squared loss on the MNIST dataset over five independent runs.

of "2" and "8" and not downsampling), with $n = 11809$ and $d = 784$. The parameter P_s was chosen to be 24, corresponding to a network width of at most 48, the regularization strength β was chosen as 0.001, and the ADMM step size γ_a was chosen as 0.1. The ADMM method was allowed to run ten iterations. The prediction accuracy on the test set, the returned training cost, and the CPU time are shown in Table 2. The "CVX" method corresponds to using CVX to globally optimize the ANN by directly solving (2). This method globally optimizes the neural network and is thus considered the baseline. The solution returned by CVX is regarded as the true global optimum. "Back-propagation" is the conventional method that performs a local search on the non-convex cost function (1).

As Table 2 shows, the ADMM algorithm achieved a higher test accuracy than both CVX and back-propagation. While ADMM and CVX solve the same problem and the CVX solution achieves a lower loss, the CVX solution suffers from overfitting and cannot generalize well to the test data as a result. The training cost returned by ADMM is higher than the true optimal cost but lower than the back-propagation solution. The training time of ADMM is considerably shorter than the time required by CVX. Specifically, assembling the matrix $I + \frac{1}{\gamma_a} F^\top F + G^\top G$ required 22% of the time, and the Cholesky decomposition needed 34% of the time, while each ADMM iteration only took 4.4% of the time. Thus, running more ADMM iterations will not considerably increase the training time. If allowed more iterations, the ADMM algorithm will converge to the global optimum of (2). In contrast, back-propagation does not have this guarantee due to the non-convexity of (1). Moreover, back-propagation is very sensitive to the initializations and hyperparameters and may fail in

Method	Accuracy	CPU Time (s)	Training Cost	Global Convergence
Back-propagation	98.91 %	62.06	437.6	No
CVX	98.21 %	14217	1.007	Yes
ADMM-RBCD	98.89 %	555.8	310.3	Yes

Table 3: Average test accuracy, training cost, and CPU time achieved with the binary cross-entropy loss on the MNIST dataset over five independent runs.

some instances. While ADMM also requires a pre-specified step size γ_a, it is much more stable: its convergence to a primal optimum does not depend on the step size (Boyd et al., 2011, Appendix A). An optimal step size speeds up the training, but a suboptimal step size is also acceptable.

6.2.4 Binary cross-entropy loss (iterative u updates) – MNIST prediction

To verify the efficacy and efficiency of using the RBCD method to numerically solve (9a), we similarly experimented on the MNIST handwritten digits dataset with the binary cross-entropy loss. For this experiment, a tanh output activation is coupled with the binary cross-entropy loss:

$$\ell(p, y) = \sum_{k=1}^{n} -y_k \log(p_k) - (1 - y_k) \log(1 - p_k), \quad y_k \in \{1, 0\},$$
$$p_k = \frac{1}{2}\big(\tanh(\widehat{y}_k) + 1\big) = \frac{1}{2}\Big(\frac{e^{2\widehat{y}_k} - 1}{e^{2\widehat{y}_k} + 1} + 1\Big) = \frac{1}{1 + e^{-2\widehat{y}_k}}, \quad \widehat{y}_k = \sum_{j=1}^{m} (x_k^\top u_j)_+ \alpha_j, \quad \forall k \in [n].$$

where the subscript k specifies the k^{th} entry of a vector.

Substituting each p_k into $\ell(\cdot)$ and expanding the hyperbolic tangent function yields:

$$\ell(\widehat{y}, y) = -2\widehat{y}^\top y + \mathbf{1}^\top \log(e^{2\widehat{y}} + 1), \tag{35}$$

which is a convex loss function with the gradient given by:

$$\nabla_{\widehat{y}} \ell(\widehat{y}, y) = -2y + \frac{2}{1 + e^{-2\widehat{y}}},$$

where the operations in the second term are performed element-wise to each entry of $\widehat{y}$.

Since the RBCD subroutine finds an approximate solution to the subproblem (9a), the ADMM outer loop was allowed to run more iterations (34 iterations) with a smaller step size ($\gamma_a = 0.01$) to compensate. The regularization parameter β was chosen to be .001. Since the value of the full augmented Lagrangian gradient in the stopping condition of Algorithm 3 is difficult to obtain, we use the amount of objective improvement as a surrogate. In other words, the RBCD iterations terminate when the objective of (9a) decreases slow enough. Other experiment settings were the same as the squared loss experiment discussed in Section 6.2.3.

The experiment results with the binary cross-entropy loss are shown in Table 3. The ADMM-RBCD algorithm achieved a high test accuracy while requiring a training time 94.6% shorter than the time of globally optimizing the cost function (2) with CVX. The ADMM-RBCD method requires less time to reach a comparable accuracy than the closed-form ADMM method with the squared loss. On the other hand, ADMM-RBCD is still slower than traditional back-propagation, trading the training speed for the global convergence guarantee. The extremely slow pace of CVX training hindered the application of convex training to even medium-scaled problems. The ADMM-RBCD algorithm has made convex training much more practical by providing a balance between efficiency and optimality.

6.2.5 Choosing the ADMM step size γ_a

The proposed ADMM algorithm has a hyperparameter γ_a, which is the step size. In fact, γ_a controls the level of infeasibility of v and w. Note that while ADMM guarantees to converge to an optimal feasible solution, the optimization variables may be infeasible in intermediate steps. The feasibility of v_i and w_i to (2) is emphasized when γ_a is large, while a low objective value is emphasized when γ_a is small. For the purpose of finding optimal u_j and α_j that minimize (1), a balance between feasibility and low objective is required. In practice, if there exists a significant gap between the objective of (2) and the training cost (1), then γ_a needs to be increased. If the objective of (2) struggles to reduce, then γ_a needs to be decreased.

6.3 The SCP convex training formulation

In this subsection, we demonstrate the efficacy of the SCP relaxed training using the one-shot random sampling approach to choose $u_1, \ldots, u_N$ and explore the effect of the number of sampled weights N. We independently sampled different numbers of hidden-layer-weights and used the SCP training formulation (14) to train ANNs on the "mammographic masses" dataset (Dua and Graff, 2017). We removed instances containing NaNs and randomly selected 70% of the data for the training set and 30% for the test set, resulting in $n = 581$ and $d = 5$. We used two different regularization strengths: $\beta = 10^{-4}$ and $\beta = 10^{-2}$. The training cost and the test accuracy of each N setting are plotted in Figure 7. The neural network training process is stochastic due to the randomly generated hidden-layer weights u_j and the random splitting of training and test sets. To reduce the effect of randomness, we performed 20 independent trials for each N. Since the problem scale is small, we used CVXPY (Diamond and Boyd, 2016) and the MOSEK solver (ApS, 2019) to solve the underlying optimization problem (14).

For both regularization strength settings, adding more sampled hidden layer weights decreased the training cost. This phenomenon is expected since more sampled weights make the SCP approximation more refined. When the regularization strength β is 10^{-4}, the test accuracy increases, peaks, and then decreases as N increases. The accuracy drops when N is large, possibly because of overfitting. As a comparison, training ANNs using Algorithm 1 with P_s set to 120 achieved an average accuracy of 79.80% and an average training loss of 0.2428 on the same dataset. Directly optimizing the non-convex cost function (1) using gradient descent back-propagation with the width m set to $2P_s = 240$ achieved a 81.14% average

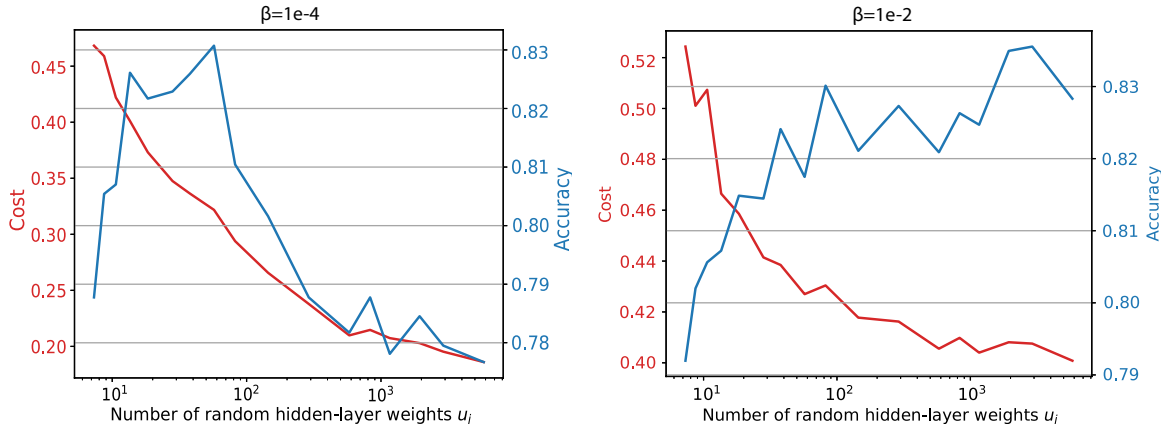

Figure 7: **FIXIT**Average accuracy and average cost with different choices of N for two different selections of the regularization strength β.

test accuracy and a 0.3560 average cost. So, with a proper choice of N, the prediction performance of the SCP convex training approach is on par with Algorithm 1 and traditional back-propagation SGD. When the regularization strength β is 10^{-2}, the test accuracy of the ANNs trained with the SCP method generally increases with N.

To verify the performance of the proposed training approach on larger-scale data, we used the SCP method to train ANNs on the MNIST handwritten digits database (LeCun et al., 2010) for binary classification between digits "2" and "8" using the binary cross-entropy loss, where $d = 784$ and $n = 11809$. With the number of sampled weights N set to 39365 (corresponding to an optimality level of $\xi\psi = 0.3$), the SCP formulation (14) was able to achieve an accuracy of 99.45%. Since the dimension of the MNIST dataset is much larger than the dimension of the previous mammographic masses dataset, the interior point method used by CVX became extremely slow. Therefore, in this experiment, we solved the SCP training optimization (14) using the ISTA algorithm, a first-order method. Compared with the ADMM approach discussed in Section 3, we were able to train much wider ANNs using the SCP approach using a similar amount of computational power. Specifically, P_s was chosen to be 24 in the ADMM experiments, corresponding to a network width of $m \leq 2P_s = 48$, whereas $m = N = 39365$ in the SCP experiment. In summary, this result demonstrates the performance and efficiency advantage of the SCP formulation (14) for medium or large machine learning problems.

6.4 Convex adversarial training

6.4.1 Hinge loss convex adversarial training – 2D illustration

To analyze the decision boundaries obtained from convex adversarial training, we ran Algorithm 1 and Algorithm 5 on 34 random points in 2-dimensional space for binary classification. The algorithms were run with the parameters $\beta = 10^{-9}$, $P_s = 360$ and $\epsilon = 0.08$. A bias term was included by concatenating a column of ones to the data matrix X.

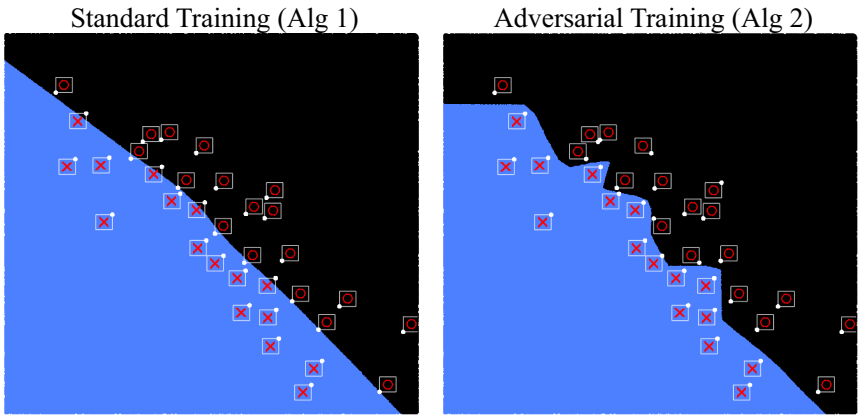

Figure 8: Visualization of the binary decision boundaries in a 2-dimensional space. The red crosses are positive training points while the red circles are negative points. The region classified as positive is in blue, whereas the negative region is in black. The white box around each training data is the ℓ_∞ perturbation bound. The white dot at a vertex of each box is the worst-case perturbation. Algorithm 5 fitted the perturbation boxes, while the standard training fitted the points.

The decision boundaries shown in Figure 8 confirm that Algorithm 5 fits the perturbation boxes as designed, coinciding with the theoretical prediction (Madry et al., 2018, Figure 3).

The decision boundaries obtained from various methods with different regularization strengths are shown in Figure 9. The two standard training methods (Algorithm 1 and GD-std) learned decision boundaries that separated the training points but failed to separate the perturbation boxes. Note that Algorithm 1 learned slightly more sophisticated boundaries while GD-std learned near-linear boundaries that were very close to one of the positive training points $\times$.

The convex adversarial training method given by Algorithm 5 learned boundaries that separated all perturbation boxes when β was 10^{-3}, 10^{-6}, or 10^{-9}. This behavior matches the theoretical illustration of adversarial training (Madry et al., 2018, Figure 3), and verifies that Algorithm 5 works as intended. When the regularization is too strong ($\beta = 10^{-2}$), the robust boundary becomes smoothed out and very similar to the standard training boundaries. The traditional adversarial training method GD-PGD learned boundaries that separated most perturbation boxes. However, the boundaries cut through the box at around $(1, -1)$ when β is 10^{-3}, 10^{-6}, or 10^{-9}. This behavior is likely caused by GD-PGD's worse convergence due to the non-convexity. When β is too large, the GD-PGD boundary also becomes smoothed out.

6.4.2 Hinge loss convex adversarial training – the optimization landscape

This subsection shows that the convex landscape and the non-convex landscape overlap for an ℓ_∞ norm bounded perturbation δ added upon a training point x_k, and thereby verify that the convex objective (21a) provides an exact certification of the non-convex loss function.

The visualizations are based on the 2-dimensional experiment described in Section 6.4.1. We use Algorithm 5 to train a robust neural network on the 2-dimensional dataset with

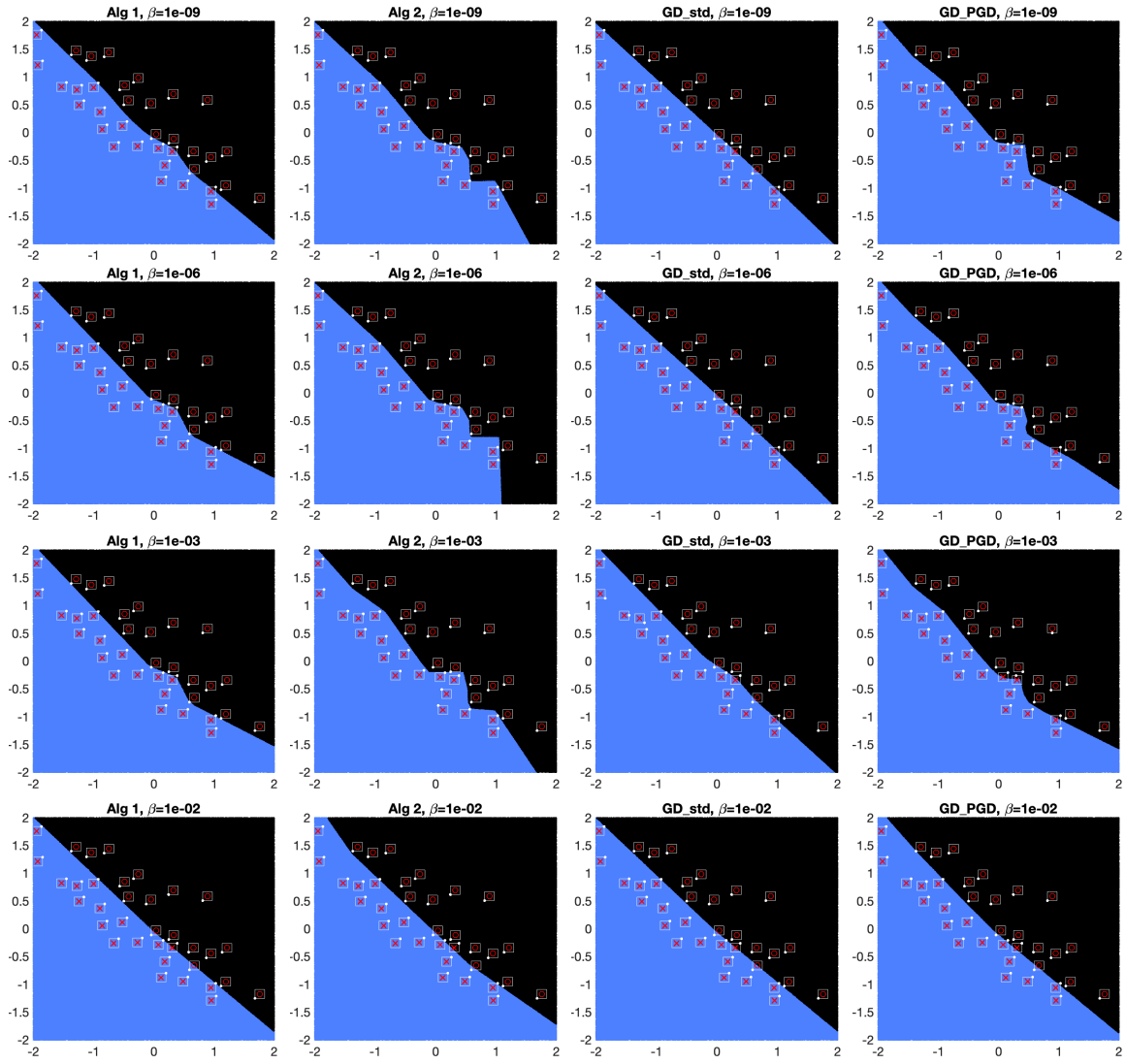

Figure 9: Decision boundaries obtained from various methods with β set to 10^{-9}, 10^{-6}, 10^{-3}, and 10^{-2}.

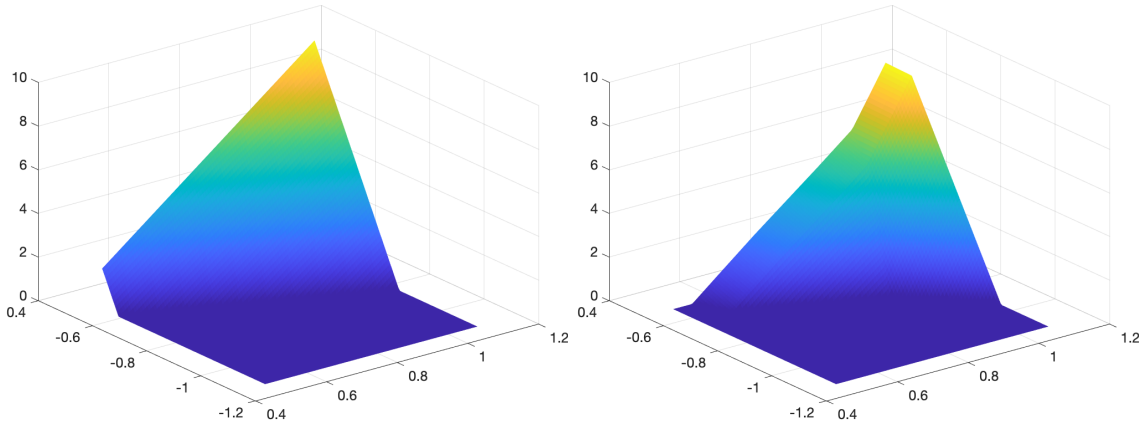

Figure 10: **FIXIT**Left: the loss landscape of the convex objective ℓ_{convex} for $\|\delta\|_\infty \leq 0.3$; right: the loss landscape of the non-convex objective $\ell_{\text{nonconvex}}$ for $\|\delta\|_\infty \leq 0.3$.

$\epsilon = 0.08$, $P_s = 360$, and $\beta = 10^{-9}$. We then randomly select one of the training points x_k and plot the loss around x_k for the convex objective (21a) and the non-convex objective (18). Specifically, for $\|\delta\|_\infty \leq 0.3$, we plot

$$\ell_{\text{convex}} = \left(1 - y_k \cdot \sum_{i=1}^{P} d_{ik}(x_k+\delta)^\top (v_i^\star - w_i^\star)\right) \text{ and } \ell_{\text{nonconvex}} = \left(1 - y_k \cdot \sum_{j=1}^{m} \left((x_k+\delta)^\top u_j^\star\right)_+ \alpha_j^\star\right),$$

where d_{ik} is the k^{th} entry of D_i, y_k is the training label corresponding to x_k. Moreover, $v_i^\star$, $w_i^\star$ are the optimizers returned by Algorithm 5 and $u_j^\star$ and $\alpha_j^\star$ are the neural network weights recovered from $v_i^\star$ and $w_i^\star$ with (4). The plots are shown in Figure 10.

For a clearer visualization, we plot $\ell_{\text{convex}} - \ell_{\text{nonconvex}}$ in Figure 11, and zoom in to the ℓ_∞ norm ball with radius $\epsilon = 0.08$. When $\ell_{\text{convex}} - \ell_{\text{nonconvex}}$ is zero, the convex objective provides an exact certificate for the non-convex loss function. **FIXIT**The right figure shows that for $\|\delta\|_\infty \leq 0.08$, the difference is zero, supporting the finding that for neural networks trained with Algorithm 5, the convex objective offers an exact certificate around the training points.

6.4.3 Hinge loss convex adversarial training – image classification

We now verify the real-world performance of the proposed convex training methods on a subset of the CIFAR-10 image classification dataset (Krizhevsky, 2012) for binary classification between the second class and the eighth class. The subset consists of 600 images downsampled to $d = 147$. The parameters are $\epsilon = 10$, $\beta = 10^{-4}$, and $P_s = 36$, corresponding to neural network widths of at most 72. We used the FGSM and PGD methods to generate adversarial examples and used both clean data and adversarial data to compare the performances of Algorithm 1, Algorithm 5, the traditional standard training method (standard back-propagation; abbreviated as GD-std in the tables), and the widely-used adversarial training method: use FGSM or PGD to solve for the inner maximum of (25) and use gradient descent

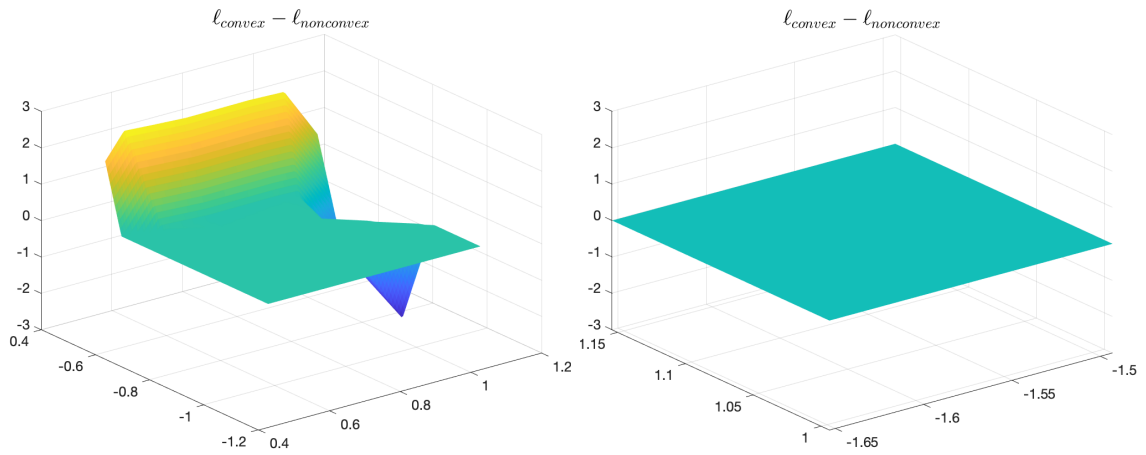

Figure 11: $\ell_{\text{convex}} - \ell_{\text{nonconvex}}$ for $\|\delta\|_\infty \leq 0.3$ (left) and zoomed in to $\|\delta\|_\infty \leq 0.08$ (right).

Table 4: Average optimal objective and accuracy on clean and adversarial data over seven runs on the CIFAR-10 database. The numbers in the parentheses are the standard deviations over the seven runs.

Method	Clean	FGSM adv.	PGD adv.	Objective
GD-std	79.56% (.4138%)	47.09% (.4290%)	45.60% (.4796%)	.3146 (.01101)
GD-FGSM	75.30% (3.104%)	61.03% (4.763%)	60.99% (4.769%)	.8370 (6.681×10^{-2})
GD-PGD	76.56% (.6038%)	62.48% (.2215%)	62.44% (.1988%)	.8220 (3.933×10^{-3})
Algorithm 1	81.01% (.8090%)	.4857% (.1842%)	.3571% (.1239%)	6.910×10^{-3} (3.020×10^{-4})
Algorithm 5	78.36% (.3250%)	66.95% (.4564%)	66.81% (0.4862%)	.6511 (6.903×10^{-3})

back-propagation to solve the outer minimization (abbreviated as GD-FGSM and GD-PGD in the tables).

Hinge loss has a flat part that has a zero gradient. To generate adversarial examples even in this part, we treat it as "leaky hinge loss" via the model $\max(\zeta(1 - \hat{y} \cdot y), 1 - \hat{y} \cdot y)$, where $\zeta \to 0^+$. Hence, the FGSM calculation (19) evaluates to

$$\tilde{x} = x - \epsilon \cdot \operatorname{sgn}\Big(y \cdot \textstyle\sum_{j:\ x^\top u_j \geq 0} (u_j \alpha_j)\Big).$$

Similarly, the PGD method (20) evaluates to

$$\tilde{x}^{t+1} = \Pi_{\mathcal{X}}\Big(\tilde{x}^t - \gamma_p \cdot \operatorname{sgn}\big(y \cdot \textstyle\sum_{j:\ x^\top u_j \geq 0} (u_j \alpha_j)\big)\Big), \quad \tilde{x}^0 = x.$$

where the projection step can be performed by clipping the coordinates that deviate more than ϵ from x. In the following experiments, we use $\gamma_p = \epsilon/30$ and run PGD for 40 steps.

The results on the CIFAR-10 subset are provided in Table 4. Convex standard training (Algorithm 1) achieved a slightly higher clean accuracy compared with GD-std and returned a much lower training cost. Such a behavior supports the findings of Theorem 2. The

convex adversarial training algorithm (Algorithm 5) achieved better accuracies on clean data and adversarial data compared with GD-FGSM and GD-PGD. While Algorithm 5 solves the upper-bound problem (27), it returned a lower training objective compared with GD-FGSM and GD-PGD, showing that the back-propagation methods failed to find an optimal network. Moreover, the back-propagation methods are very sensitive to initializations and hyperparameter choices. In contrast, since Algorithm 1 and Algorithm 5 solve convex programs, they are much less sensitive to initializations and are guaranteed to converge to their global optima.

We also experimentally compare the aforementioned SDP relaxation adversarial training method (Raghunathan et al., 2018) and the LP relaxation method (Wong and Kolter, 2018) against our work on the CIFAR-10 subset. We observe that an iteration of the LP or the SDP method is faster than a GD-PGD iteration. However, the ANNs trained with the LP or SDP method achieve worse accuracies and robustness than those trained with Algorithm 5: the LP method achieves a 74.05% clean accuracy and a 58.65% PGD accuracy, whereas the SDP method achieves 73.35% on clean data and 40.45% on PGD adversaries. For SDP, the robustness parameter is chosen as $\lambda = .04$, and larger λ causes the algorithm to fail. These results support the speculation that Algorithm 5 trains more robust ANNs and that the LP and SDP relaxations can be extremely loose. The LP and SDP formulations are also significantly less stable than Algorithm 5, and training often fails. As discussed before, while (Raghunathan et al., 2018; Wong and Kolter, 2018) applies the convex relaxation method to the adversarial training problem, their resulted training formulations are non-convex.

Furthermore, the presence of an ℓ_1 norm term in the upper-bound formulations (27) and (31) indicates that adversarial training with a small ϵ has a regularizing effect, which can improve generalization, supporting the finding of (Kurakin et al., 2017). In the above experiments, Algorithm 5 outperforms Algorithm 1 on adversarial data, highlighting the contribution of Algorithm 5: a novel efficient convex adversarial training procedure that reliably trains robust neural networks. Compared with Algorithm 1, Algorithm 5 retains the advantage in the absence of spurious local minima while achieving adversarial robustness.

6.4.4 Squared loss convex adversarial training

The performance of the proposed robust optimization problem (31) is compared with the standard training problem (2) on a contrived 1-dimensional dataset. Figure 12 shows the true relationship between the data vector X and the target output y. Throughout this experiment, training data are constructed by uniformly sampling eight points from this distribution, and test data are similarly constructed by uniformly sampling 100 points. A bias term is included by concatenating a column of ones to X.

The training and test procedure are repeated for 100 trials with convex standard training (Algorithm 1). For convex adversarial training (Algorithm 5), we varied the perturbation radius $\epsilon = 0.1, \ldots, 0.9$. The training and test procedure was carried out for ten trials for each ϵ. Figure 13 reports the average test mean square error (MSE) for each setup.

The adversarial training procedure outperforms standard training for all ϵ choices. We further observe that the average MSE is the lowest at $\epsilon \approx 0.3$. This behavior arises as the

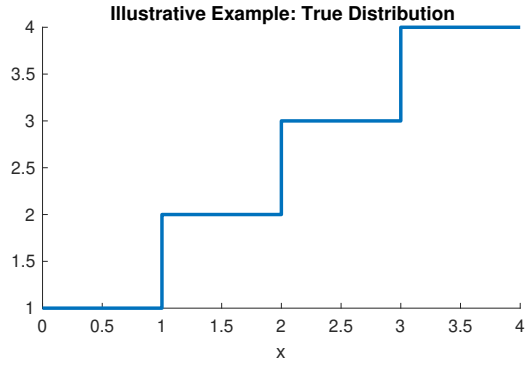

Figure 12: The true relationship between the data x and the targets y used in the illustrative example in Section 6.4.4. The training ($n = 8$ points) and test ($n = 100$ points) sets are uniformly sampled from the distribution.

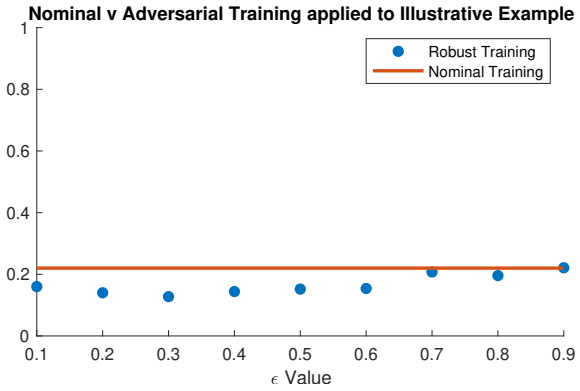

Figure 13: The robust training approach (31) outperforms the standard approach for different $\epsilon \in \{0.1, ..., 0.9\}$ on the dataset studied in Section 6.4.4.

robust problem attempts to account for all points within the uncertainty interval around the sampled training points. When ϵ is too small, the robust problem approaches the standard training problem. Larger values of ϵ cause the uncertainty interval to overestimate the constant regions of the true distribution, increasing the MSE.

7. Concluding Remarks

We used the sampled convex program theory to characterize the quality of the solution obtained from an approximation heuristic, providing theoretical insights into practical convex training. We then showed that a separating scheme enables the application of the ADMM algorithm to the convex training formulation, achieving a quadratic per-iteration computational complexity and a linear convergence rate when combined with the approximation scheme. We also introduced a simpler convex training formulation based on SCP relaxation and characterized its solution quality. This simpler formulation solves more efficient unconstrained convex programs, and show that ELMs are in fact convex relaxations to ANNs. Compared to the traditional back-propagation algorithms, our proposed training algorithms have theoretical convergence rate guarantees. Compared with naively solving the convex training formulation using general-purpose solvers, the improved computational complexities of our algorithms make a significant step towards convex training in practice.

We also used the robust convex optimization analysis to derive convex programs that train adversarially robust ANNs. Compared with traditional adversarial training methods, including GD-FGSM and GD-PGD, the favorable properties of convex optimization endow convex adversarial training with the following advantages:

- **Global convergence to an upper-bound:** For the case of hinge loss and squared loss, convex adversarial training provably converges to an upper-bound to the global optimum cost, offering superior interpretability.

- **Guaranteed adversarial robustness on training data:** As shown in Theorem 7, the inner maximization over the robust loss function is solved exactly.
- **Hyperparameter-free:** In practice, Algorithm 5 can automatically determine its step size with line search, not requiring any preset parameters.
- **Immune to vanishing / exploding gradients:** The convex training method avoids this problem completely because it does not rely on back-propagation.

Overall, the analysis of this work makes it easier and more efficient to train interpretable and robust ANNs with global convergence guarantees, potentially facilitating the application of ANNs in safety-critical applications.

References

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

Brendon G. Anderson, Ziye Ma, Jingqi Li, and Somayeh Sojoudi. Tightened convex relaxations for neural network robustness certification. In *59th IEEE Conference on Decision and Control*, 2020.

MOSEK ApS. *The MOSEK optimization toolbox for MATLAB manual. Version 9.0*, 2019.

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

Anish Athalye, Nicholas Carlini, and David Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. In *Proceedings of the 35th International Conference on Machine Learning*, 2018.

Francis Bach. Breaking the curse of dimensionality with convex neural networks. *Journal of Machine Learning Research*, 18(19), 2017.

Yatong Bai, Tanmay Gautam, Yu Gai, and Somayeh Sojoudi. Practical convex formulation of robust one-hidden-layer neural network training. *CoRR*, 2021.

A. Beck and M. Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM Journal on Imaging Sciences*, 2, 2009.

Eugene Belilovsky, Michael Eickenberg, and Edouard Oyallon. Greedy layerwise learning can scale to ImageNet. In *Proceedings of the 36th International Conference on Machine Learning*, 2019.

Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. In *Proceedings of the Twentieth Annual Conference on Neural Information Processing Systems*, 2006a.

Yoshua Bengio, Nicolas Roux, Pascal Vincent, Olivier Delalleau, and Patrice Marcotte. In *Advances in Neural Information Processing Systems*, 2006b.

Yoshua Bengio, Nicolas Roux, Pascal Vincent, Olivier Delalleau, and Patrice Marcotte. Convex neural networks. In *Advances in Neural Information Processing Systems*, 2006c.

Avrim Blum, Travis Dick, Naren Manoj, and Hongyang Zhang. Random smoothing might be unable to certify ℓ_∞ robustness for high-dimensional images. *Journal of Machine Learning Research*, 21(211), 2020.

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

Stephen P. Boyd, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, 3(1), 2011.

Alon Brutzkus and Amir Globerson. Globally optimal gradient descent for a convnet with gaussian inputs. In *Proceedings of the 34th International Conference on Machine Learning, ICML*, 2017.

Giuseppe Calafiore and M. C. Campi. Uncertain convex programs: randomized solutions and confidence levels. *Mathematical Programming*, 102(1), 2005.

Marco C. Campi, Simone Garatti, and Maria Prandini. The scenario approach for systems and control design. *Annual Reviews in Control*, 33(2), 2009.

Jeremy Cohen, Elan Rosenfeld, and Zico Kolter. Certified adversarial robustness via randomized smoothing. In *Proceedings of the 36th International Conference on Machine Learning*, 2019.

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

Simon S. Du, Xiyu Zhai, Barnabas Poczos, and Aarti Singh. Gradient descent provably optimizes over-parameterized neural networks. In *International Conference on Learning Representations*, 2019.

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

Jonathan Eckstein and Wang Yao. Approximate ADMM algorithms derived from lagrangian splitting. *Computational Optimization and Applications*, 68(2), 2017.

Tolga Ergen and Mert Pilanci. Implicit convex regularizers of {cnn} architectures: Convex optimization of two- and three-layer networks in polynomial time. In *International Conference on Learning Representations*, 2021a.

Tolga Ergen and Mert Pilanci. Global optimality beyond two layers: Training deep relu networks via convex programs. In *Proceedings of the 38th International Conference on Machine Learning*, 2021b.

Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *3rd International Conference on Learning Representations*, 2015.

Michael Grant and Stephen Boyd. CVX: Matlab software for disciplined convex programming, version 2.1, March 2014.

Magnus R. Hestenes. Multiplier and gradient methods. *Journal of Optimization Theory and Applications*, 4(5), 1969.

Mingyi Hong and Zhi-Quan Luo. On the linear convergence of the alternating direction method of multipliers. *Mathematical Programming*, 162(1-2), 2017.

Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2), 1991.

Ruitong Huang, Bing Xu, Dale Schuurmans, and Csaba Szepesvári. Learning with a strong adversary. *CoRR*, 2015.

Sandy H. Huang, Nicolas Papernot, Ian J. Goodfellow, Yan Duan, and Pieter Abbeel. Adversarial attacks on neural network policies. In *5th International Conference on Learning Representations*, 2017.

Mahesh Jangid and Sumit Srivastava. Handwritten devanagari character recognition using layer-wise training of deep convolutional neural networks and adaptive gradient methods. *Journal of Imaging*, 4(2), 2018.

Alex Krizhevsky. Learning multiple layers of features from tiny images. *University of Toronto*, 05 2012.

Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. Adversarial machine learning at scale. In *5th International Conference on Learning Representations*, 2017.

Jonathan Lacotte and Mert Pilanci. All local minima are global for two-layer relu neural networks: The hidden convex optimization landscape. *CoRR*, 2020.

Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. *ATT Labs [Online]*, 2, 2010.

Zhaosong Lu and Lin Xiao. On the complexity analysis of randomized block-coordinate descent methods. *Mathematical Programming*, 152(1-2), 2015.

Ziye Ma and Somayeh Sojoudi. Strengthened SDP verification of neural network robustness via non-convex cuts. *CoRR*, 2020.

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations*, 2018.

Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: A simple and accurate method to fool deep neural networks. In *2016 IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

Mert Pilanci and Tolga Ergen. Neural networks are convex regularizers: Exact polynomial-time convex optimization formulations for two-layer networks. In *Proceedings of the 37th International Conference on Machine Learning*, 2020.

Aditi Raghunathan, Jacob Steinhardt, and Percy Liang. Certified defenses against adversarial examples. In *International Conference on Learning Representations*, 2018.

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088), 1986.

Arda Sahiner, Tolga Ergen, John M. Pauly, and Mert Pilanci. Vector-output re{lu} neural network problems are copositive programs: Convex analysis of two layer networks and polynomial-time algorithms. In *International Conference on Learning Representations*, 2021.

Maurice Sion. On general minimax theorems. *Pacific Journal of Mathematics*, 8(1), 1958.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *2nd International Conference on Learning Representations*, 2014.

Gavin Taylor, Ryan Burmeister, Zheng Xu, Bharat Singh, Ankit Patel, and Tom Goldstein. Training neural networks without gradients: A scalable admm approach. In *Proceedings of The 33rd International Conference on Machine Learning*, 2016.

Luca Venturi, Afonso S. Bandeira, and Joan Bruna. Spurious valleys in one-hidden-layer neural network optimization landscapes. *Journal of Machine Learning Research*, 20(133), 2019.

Junxiang Wang, Fuxun Yu, Xiang Chen, and Liang Zhao. ADMM for efficient deep learning with global convergence. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019.

Eric Wong and Zico Kolter. Provable defenses against adversarial examples via the convex outer adversarial polytope. In *Proceedings of the 35th International Conference on Machine Learning*, 2018.

Table 5: Average optimal objective, CPU time, and prediction accuracy on clean and adversarial test data over seven runs on the CIFAR-10 dataset.

$P_s = 24$ and $m = 48$				
Method	Clean	FGSM adv.	PGD adv.	Objective
GD-std	81.40 %	54.72 %	54.66 %	.1486
GD-FGSM	77.75 %	64.53 %	64.46 %	.7038
GD-PGD	76.49 %	64.70 %	64.64 %	.7363
Algorithm 1	80.51 %	.2500 %	.1357 %	.007516
Algorithm 5	78.54 %	66.91 %	66.75 %	.7123
$P_s = 18$ and $m = 36$				
Method	Clean	FGSM adv.	PGD adv.	Objective
GD-std	81.04 %	54.86 %	54.82 %	.1550
GD-FGSM	77.29 %	64.69 %	64.56 %	.7131
GD-PGD	76.44 %	64.76 %	64.74 %	.7365
Algorithm 1	79.71 %	.3571 %	.2714 %	.008953
Algorithm 5	78.71 %	63.89 %	63.67 %	.8049

Appendix A. Additional Experiments

A.1 Adversarial training on the CIFAR-10 dataset with different P_s

In this section, we repeat the experiments in Section 6.4.3 on the CIFAR-10 dataset with different numbers of sampled D_h matrices. Compared with Table 4, which used $\epsilon = 10$, $\beta = 10^{-4}$, and $P_s = 36$, these additional experiments keep the same ϵ and β settings but reduce P_s to 24 and 18. For fair comparisons, we set the ANN width m equal to $2P_s$ for back-propagation implementations in all experiments. Each experiment was repeated seven times and the results are shown in Table 5.

Table 5 shows that the effect of the neural network width on the prediction performance is not significant for all methods, but Algorithm 1 and Algorithm 5 are affected more: when P_s is 36 or 24, Algorithm 5 outperforms GD-FGSM and GD-PGD, but when P_s is 18, Algorithm 5 achieves a lower accuracy than FGSM and PGD. Our explanation is that the constraints in the convex training formulations become more restrictive when P_s is small, worsening the suboptimalities of the solutions. Therefore, Algorithm 1 and Algorithm 5 are more suitable for ANN that are not too narrow.

Appendix B. Extensions

B.1 Extending the analysis to CNNs

The paper (Ergen and Pilanci, 2021a) shows that the convex ANN training approach extends to various convolutional neural network (CNN) architectures. Taking advantage of this

result, the convex adversarial training formulations similarly generalize. In this part of the appendix, we change our notations to align with (Ergen and Pilanci, 2021a). For example, the robust counterpart of the average pooling two-layer CNN convex training formulation (cf. Equations (4) and (26) in (Ergen and Pilanci, 2021a)) is: **FIXIT**

$$\begin{aligned}\min_{\{c_i,c_i'\}_{i=1}^{P_{conv}}} &\left(\max_{\mathbf{X}_k\in\mathcal{X}_k} \ell\Big(\sum_{i=1}^{P_{conv}} \sum_{k=1}^{K} \mathbf{D}(S_i^k)\mathbf{X}_k(c_i'-c_i), \mathbf{y} \Big) + \beta \sum_{i=1}^{P_{conv}} \big(\|c_i\|_2 + \|c_i'\|_2\big) \right) \\ \text{s.t.} \quad & \min_{\mathbf{X}_k\in\mathcal{X}_k} \big(2\mathbf{D}(S_i^k)-\mathbf{I}_n\big)\mathbf{X}_k c_i' \geq 0, \quad \min_{\mathbf{X}_k\in\mathcal{X}_k} \big(2\mathbf{D}(S_i^k)-\mathbf{I}_n\big)\mathbf{X}_k c_i \geq 0, \ \ \forall i,k,\end{aligned}$$

where $\mathcal{X}_k$ is the corresponding perturbation set of the patch $\mathbf{X}_k$.

The next step would be to show that the above formulation is equivalent to a classic convex optimization. Note that each robust constraint is an LP subproblem that can be solved in closed form, which means that the robust constraints can be cast as equivalent classic constraints. When $\ell(\cdot)$ is the squared loss, the above equation becomes a robust second-order cone program (SOCP), which is known to be a convex optimization (similar to (30) of this work). Otherwise, if $\ell(\cdot)$ is monotonously increasing or decreasing in the CNN output $\widehat{\mathbf{y}}$ (examples include the hinge loss and the binary cross-entropy loss), the inner maximization problem

$$\arg\max_{\mathbf{X}_k\in\mathcal{X}_k} \ell\Big(\sum_{i=1}^{P_{conv}} \sum_{k=1}^{K} \mathbf{D}(S_i^k)\mathbf{X}_k(c_i'-c_i), \mathbf{y} \Big)$$

reduces to

$$\arg\max_{\mathbf{X}_k\in\mathcal{X}_k} \sum_{i=1}^{P_{conv}} \sum_{k=1}^{K} \mathbf{D}(S_i^k)\mathbf{X}_k(c_i'-c_i) \quad \text{or} \quad \arg\min_{\mathbf{X}_k\in\mathcal{X}_k} \sum_{i=1}^{P_{conv}} \sum_{k=1}^{K} \mathbf{D}(S_i^k)\mathbf{X}_k(c_i'-c_i),$$

which are LPs that can be solved in closed form. Substituting the closed-form solution yields the desired convex adversarial training formulations.

Similarly, for max pooling two-layer CNNs, the robust counterpart becomes (cf. Equation (7) of (Ergen and Pilanci, 2021a)):

$$\begin{aligned}\min_{\{c_i,c_i'\}_{i=1}^{P_{conv}}} &\left(\max_{\mathbf{X}_k\in\mathcal{X}_k} \ell\Big(\sum_{i=1}^{P_{conv}} \sum_{k=1}^{K} \mathbf{D}(S_i^k)\mathbf{X}_k(c_i'-c_i), \mathbf{y} \Big) + \beta \sum_{i=1}^{P_{conv}} \big(\|c_i\|_2 + \|c_i'\|_2\big) \right) \\ \text{s.t.} \quad & \min_{\mathbf{X}_k\in\mathcal{X}_k} \big(2\mathbf{D}(S_i^k)-\mathbf{I}_n\big)\mathbf{X}_k c_i' \geq 0, \quad \min_{\mathbf{X}_k\in\mathcal{X}_k} \big(2\mathbf{D}(S_i^k)-\mathbf{I}_n\big)\mathbf{X}_k c_i \geq 0, \ \ \forall i,k. \\ & \min_{\mathbf{X}_k\in\mathcal{X}_k} \mathbf{D}(S_i^k)\mathbf{X}_k c_i \geq \max_{\mathbf{X}_j\in\mathcal{X}_j} \mathbf{D}(S_i^k)\mathbf{X}_j c_i, \ \forall i,j,k, \\ & \min_{\mathbf{X}_k\in\mathcal{X}_k} \mathbf{D}(S_i^k)\mathbf{X}_k c_i' \geq \max_{\mathbf{X}_j\in\mathcal{X}_j} \mathbf{D}(S_i^k)\mathbf{X}_j c_i', \ \forall i,j,k.\end{aligned}$$

where each additional robust constraint is an LP subproblem that can be solved in closed form.

The same robust optimization techniques can be applied to three-layer CNNs (cf. Equation (11) in (Ergen and Pilanci, 2021a)) and derive corresponding convex adversarial training

formulations. In general, the convex standard training formulations for different NNs / CNNs share very similar structures. Therefore, many convex standard training formulations can be "robustified" by recasting as mini-max formulations. Whether these mini-max formulations can be reformed into classic convex optimizations depends on the specific structures of the problems. For CNNs with two or three layers considered in (Ergen and Pilanci, 2021a), such classic convex formulations can be derived.

Similarly, the ADMM splitting scheme, discussed in Section 3, also applies to the above CNN formulations. The CNN training formulations can be similarly split into loss function terms, regularization terms, and linear inequality constraints.

B.2 The vector-output counterpart of Theorem 4

For vector-output networks, the output $\widehat{Y}$ becomes a matrix in $\mathbb{R}^{n\times c}$. Specifically, $\widehat{Y} = \sum_{j=1}^{m}(Xu_j)_+\alpha_j^\top$, where $\alpha_j \in \mathbb{R}^c$ for all $j \in [m]$ and c is the dimension of the output (the number of classes). The target (label) matrix Y also has the dimension of $n \times c$.

When an arbitrary convex loss function $\ell(\widehat{Y}, Y)$ is considered, the minimization of the non-convex training cost, expressed as

$$p^\star = \min_{(u_j,\alpha_j)_{j=1}^m} \ell\Big(\sum_{j=1}^{m}(Xu_j)_+\alpha_j^\top, Y\Big) + \frac{\beta}{2}\sum_{j=1}^{m}\Big(\|u_j\|_2^2 + \|\alpha_j\|_2^2\Big)$$

is equivalent to the following problem: (Sahiner et al., 2021, Lemma 4):

$$p^\star = \min_{(u_j,\alpha_j)_{j=1}^m} \ell\Big(\sum_{j=1}^{m}(Xu_j)_+\alpha_j^\top, Y\Big) + \beta\sum_{j=1}^{m}\|\alpha_j\|_2 \quad \text{s.t.} \quad \|u_j\|_2 \le 1,\ \forall j \in [m]. \tag{36}$$

When the width m is not smaller than $m^\star$, where $m^\star$ is defined in the same manner as for the scalar-output case, strong duality holds between (36) and the following dual formulation (Sahiner et al., 2021, Lemma 5 and Appendix 6):

$$d^\star = \max_{Z\in\mathbb{R}^{n\times c}} -\ell^*(Z) \quad \text{s.t.} \quad \|Z^\top(Xu)_+\|_2 \le \beta, \quad \forall \|u\|_2 \le 1,$$

where $\ell^*(\cdot)$ is again the Fenchel conjugate function of $\ell(\cdot)$.

Similarly, one can sample $u_1, \ldots, u_N \in \mathbb{R}^d$ on the unit Euclidean norm sphere via a uniform distribution and construct the following SCP as an approximation to $p^\star$:

$$p_{s3}^\star = \min_{(\alpha_i)_{i=1}^N} \ell\Big(\sum_{i=1}^{N}(Xu_i)_+\alpha_i^\top, Y\Big) + \beta\sum_{i=1}^{N}\|\alpha_i\|_2, \tag{37}$$

which has a strong dual

$$d^\star = \max_{Z\in\mathbb{R}^{n\times c}} -\ell^*(Z) \quad \text{s.t.} \quad \|Z^\top(Xu_i)_+\|_2 \le \beta, \quad \forall i \in [N].$$

The rest of the proof of Theorem 4 readily applies to characterize the level of subuptimality of the SCP (37) compared with the UCP (36), and (37) is a finite-dimensional convex optimization that can be solved to train vector-output ReLU networks. Note that the group sparse regularization is used for vector-output networks in place of the ℓ_1 regularization for the scalar-output case.

B.3 ℓ_p norm-bounded perturbation set for hinge loss

Theorem 7 can be extended to the following ℓ_p norm-bounded perturbation set:

$$\widetilde{\mathcal{X}} = \left\{ X + \Delta \in \mathbb{R}^{n\times d} \;\middle|\; \Delta = [\delta_1 \;\cdots\; \delta_n]^\top,\; \|\delta_k\|_p \le \epsilon,\; \forall k \in [n] \right\}$$

.

In the case of performing binary classification with a hinge-lossed neural network, the convex adversarial training problem then becomes:

$$\begin{aligned} \min_{(v_i,w_i)_{i=1}^{\widehat{P}}} \left(\frac{1}{n} \sum_{k=1}^{n} \left(1 - y_k \sum_{i=1}^{\widehat{P}} d_{ik} x_k^\top (v_i - w_i) + \epsilon \cdot \left\| \sum_{i=1}^{\widehat{P}} d_{ik}(v_i - w_i) \right\|_{p*} \right)_+ \right. \\ \left. + \beta \textstyle\sum_{i=1}^{\widehat{P}} \left(\|v_i\|_2 + \|w_i\|_2 \right) \right) \\ \text{s.t.} \quad (2D_i - I_n) X v_i \ge \epsilon \|v_i\|_{p*}, \quad (2D_i - I_n) X w_i \ge \epsilon \|w_i\|_{p*}, \quad \forall i \in [\widehat{P}] \end{aligned} \tag{38}$$

where $D_1, \ldots, D_{\widehat{P}}$ are all distinct diagonal matrices associated with $\mathrm{diag}([Xu \ge 0])$ for all possible $u \in \mathbb{R}^d$ and all $X + \Delta$ at the *boundary* of $\widetilde{\mathcal{X}}$. Note that $\|\cdot\|_{p*}$ is the dual norm of $\|\cdot\|_p$.

Appendix C. Proofs

C.1 Proof of Theorem 2

We start by recasting the semi-infinite constraint of the dual formulation (3) as $\max_{\|u\|_2\le 1} |v^\top (Xu)_+| \le \beta$, and obtain

$$\max_{\|u\|_2\le 1} \left| v^\top (Xu)_+ \right| = \max_{\|u\|_2\le 1} \left| v^\top \mathrm{diag}([Xu \ge 0]) Xu \right| = \max_{i\in[P]} \left(\max_{\substack{\|u\|_2\le 1 \\ (2D_i - I_n)Xu \ge 0}} \left| v^\top D_i X u \right| \right),$$

where the last equality holds by the definition of the D_i matrices: $D_1 \ldots, D_P$ are all distinct matrices that can be formed by $\mathrm{diag}([Xu \ge 0])$ for some $u \in \mathbb{R}^d$. The constraint $(2D_i - I_n)Xu \ge 0$ is equivalent to $D_i X u \ge 0$ and $(I_n - D_i)Xu \le 0$, which forces $D_i = \mathrm{diag}([Xu \ge 0])$ to hold.

Therefore, the dual formulation (3) can be recast as

$$\max_v -\ell^*(v) \qquad \text{s.t.} \quad \max_{\substack{\|u\|_2\le 1 \\ (2D_i - I_n)Xu \ge 0}} \left| v^\top D_i X u \right| \le \beta,\; \forall i \in [P]. \tag{39}$$

To form a tractable convex program that provides an approximation to (39), one can independently sample a subset of the diagonal matrices. One possible sampling procedure is presented in Algorithm 1. The sampled matrices, denoted as $D_1, \ldots, D_{P_s}$, can be used to construct the relaxed problem:

$$d_{s1}^\star = \max_v -\ell^*(v) \qquad \text{s.t.} \max_{\substack{\|u\|_2\le 1 \\ (2D_h - I_n)Xu \ge 0}} \left|v^\top D_h X u\right| \le \beta,\ \forall h \in [P_s]. \tag{40}$$

The optimization problem (40) is convex with respect to v. (Pilanci and Ergen, 2020) has shown that (39) has the same optimal objective as its dual problem (2). By following precisely the same derivation, it can be shown that (40) has the same optimal objective as (5) and $p_{s1}^\star = d_{s1}^\star$. Moreover, if an additional diagonal matrix D_{P_s+1} is independently randomly sampled to form (6), then we also have $p_{s2}^\star = d_{s2}^\star$, where

$$d_{s2}^\star = \max_v -\ell^*(v) \qquad \text{s.t.} \max_{\substack{\|u\|_2\le 1 \\ (2D_h - I_n)Xu \ge 0}} \left|v^\top D_h X u\right| \le \beta,\ \forall h \in [P_s+1].$$

Thus, the level of suboptimality of (40) compared with (39) is the level of suboptimality of (5) compared with (2). Notice that by introducing a slack variable $w \in \mathbb{R}$, (39) can be represented as an instance of the uncertain convex program (UCP) with $n+1$ optimization variables, defined in (Calafiore and Campi, 2005):

$$\max_{v,w:\ w \le -\ell^*(v)} w \qquad \text{s.t.} \max_{\substack{\|u\|_2\le 1 \\ (2D_i - I_n)Xu \ge 0}} \left|v^\top D_i X u\right| \le \beta,\ \forall i \in [P].$$

The relaxed problem (40) can be regarded as a corresponding SCP. Suppose that $w^\star, v^\star$ is a solution to the sampled convex problem (40). It can be concluded from (Calafiore and Campi, 2005, Theorem 1) and (Campi et al., 2009, Theorem 1) that if $P_s \ge \min\left\{\frac{n+1}{\psi\xi} - 1, \frac{2}{\xi}(n+1-\log\psi)\right\}$, then $v^\star$ satisfies the original constraints of the UCP (39) with high probability. Specifically, with probability no smaller than $1-\xi$,

$$\mathbb{P}\Big\{D \in \mathcal{D}\ :\ \max_{\substack{\|u\|_2\le 1 \\ (2D - I_n)Xu \ge 0}} \left|v^{\star\top} D X u\right| > \beta\Big\} \le \psi.$$

where $\mathcal{D}$ denotes the set of all diagonal matrices that can be formed by $\mathrm{diag}([Xu \ge 0])$ for some $u \in \mathbb{R}^d$, which is the set formed by $D_1, \ldots, D_P$.

Since D_{P_s+1} is randomly sampled from $\mathcal{D}$, we have

$$\mathbb{P}\Big\{D \in \mathcal{D}\ :\ \max_{\substack{\|u\|_2\le 1 \\ (2D - I_n)Xu \ge 0}} \left|v^{\star\top} D X u\right| > \beta\Big\} = \mathbb{P}\Big\{\max_{\substack{\|u\|_2\le 1 \\ (2D_{P_s+1} - I_n)Xu \ge 0}} \left|v^{\star\top} D_{P_s+1} X u\right| > \beta\Big\}$$

Thus, with probability no smaller than $1-\xi$, it holds that

$$\mathbb{P}\Big\{\max_{\substack{\|u\|_2\le 1 \\ (2D_{P_s+1} - I_n)Xu \ge 0}} \left|v^{\star\top} D_{P_s+1} X u\right| > \beta\Big\} \le \psi.$$

Moreover, $d_{s2}^\star < d_{s1}^\star$ if and only if $\left|v^{\star\top} D_{P_s+1} X u\right| > \beta$ with $d_{s2}^\star = d_{s1}^\star$ otherwise. The proof is completed by noting that $p_{s1}^\star = d_{s1}^\star$ and $p_{s2}^\star = d_{s2}^\star$. □

C.2 Proof of Theorem 3

We start by rewriting (8) as

$$\min_{v,s,u:\ s\geq 0} f_1(u) + f_2(v,s) \quad \text{s.t.} \quad E_1 u - E_2 \begin{bmatrix} v \\ s \end{bmatrix} = 0, \tag{41}$$

where $f_1(u) = \ell(Fu, y)$, $f_2(v,s) = \beta\|v\|_{2,1}$, $E_1 = \begin{bmatrix} I \\ G \end{bmatrix}$, and $E_2 = I$.

Furthermore, let $L(u, v, s, \nu, \lambda)$ denote the augmented Lagrangian:

$$L(u,v,s,\nu,\lambda) := \\ f_1(u) + \beta\|v\|_{2,1} + \mathbb{I}_{\geq 0}(s) + \frac{\gamma_a}{2}\Big(\|u - v + \lambda\|_2^2 - \|\lambda\|_2^2\Big) + \frac{\gamma_a}{2}\Big(\|Gu - s + \nu\|_2^2 - \|\nu\|_2^2\Big)$$

(Hong and Luo, 2017, Theorem 3.1) shows that the ADMM algorithm converges linearly when the objective satisfies seven conditions. We show that these conditions are all satisfied for (41) given the assumptions of Theorem 3:

(a) It can be easily shown that (41) attains a global solution because the feasible set of the equivalent problem (2) is non-empty.

(b) We can then decompose $f_1(u)$ into $g_1(Fu) =: \ell(Fu, y)$ and $h_1(u) =: 0$ and define $h_2(\cdot) =: f_2(\cdot)$. When the loss $\ell(\widehat{y}, y)$ is convex with respect to $\widehat{y}$, the functions $g_1(\cdot), h_1(\cdot), h_2(\cdot)$ are all convex and continuous.

(c) When $\ell(\widehat{y}, y)$ is strictly convex and continuously differentiable with a uniform Lipschitz continuous gradient with respect to $\widehat{y}$, the function $g_1(\cdot)$ is strictly convex and continuously differentiable with a uniform Lipschitz continuous gradient.

(d) The epigraph of $h_1(\cdot) = 0$ is a polyhedral set. Moreover, $h_2(v,s) = \|v\|_{2,1} = \sum_{i=1}^{P}(\|v_i\|_2 + \|w_i\|_2)$ by definition.

(e) The constant function $h_1(\cdot)$ is trivially finite. Furthermore, for all u, v, s that make $L(u, v, s, \nu, \lambda)$ finite, it must hold that $f_1(u) < +\infty$, $v < +\infty$, and $s \geq 0$. Therefore, $h_2(\cdot)$ must be finite.

(f) E_1 and E_2 both have full column rank since the identity matrix has full column rank.

(g) When $u \to \infty$, we have $L(u, v, s, \nu, \lambda) \to \infty$. Hence, the solution to (9a) must be finite as long as the initial points $u^0, v^0, s^0, \lambda^0, \nu^0$ are finite. The solutions to (9b) and (9c) are also finite, since the closed-form solutions are derived in Section 3.1. Therefore, the sequence $\{(u^k, v^k, s^k, \lambda^k, \nu^k)\}$ is finite. Thus, there exist finite $u_{\max}, v_{\max}, s_{\max}$ such that (41) is equivalent to the formulation below:

$$\begin{aligned} \min_{v,s,u} \quad & f_1(u) + f_2(v,s) \\ \text{s.t.} \quad & E_1 u - E_2 \begin{bmatrix} v \\ s \end{bmatrix} = 0, \ \ \|u\|_\infty \leq u_{\max}, \ \ \|v\|_\infty \leq v_{\max}, \ \ \|s\|_\infty \leq s_{\max}. \end{aligned} \tag{42}$$

Furthermore, the ADMM algorithm that solves (42) is equivalent to Algorithm 2. The feasible set of (42) is a compact polyhedral set formed by the ℓ_∞ norm constraints.

Thus, by the application of (Hong and Luo, 2017, Theorem 3.1), the desired result holds true when the step size γ_a is sufficiently small **FIXIT**. □

C.3 Proof of Theorem 4

As discussed in Section 4.2, strong duality holds between (13) and (16), as well as between (14) and (17). Here, we introduce a slack variable w and cast (16) as a canonical uncertain convex program with $n+1$ optimization variables and a linear objective, where n is the number of training data points:

$$\begin{aligned} &\min_{(v,w)\in\mathcal{F}} w \\ \text{s.t.}\quad & f(v,w,u) := |v^\top (Xu)_+| - \beta \le 0,\ \forall u \in \mathcal{G} \\ & \mathcal{F} = \left\{ v \in \mathbb{R}^n, w \in \mathbb{R} \,\middle|\, \|y - v\|_2^2 - 2w \le 0 \right\} \\ & \mathcal{G} = \left\{ u \,\middle|\, \|u\|_2 = 1 \right\}. \end{aligned}$$

By leveraging (Calafiore and Campi, 2005, Theorem 1) and (Campi et al., 2009, Theorem 1), we can conclude that if $N \ge \min\left\{ \frac{n+1}{\psi\gamma} - 1, \frac{2}{\gamma}(n+1-\log\psi) \right\}$, then with probability no smaller than $1-\gamma$, the solution $v^\star$ to the randomized problem (17) satisfies $\mathbb{P}\{u : \|u\|_2 = 1, |v^{\star\top}(Xu)_+| > \beta\} \le \psi$. Since u_{N+1} is randomly generated on the Euclidean norm sphere via a uniform distribution, it holds that $\mathbb{P}\{|v^{\star\top}(Xu_{N+1})_+| > \beta\} \le \psi$.

Consider the following dual formulation with the newly sampled hidden neuron u_{N+1} included:

$$d_{s4}^\star = \max_{v\in\mathbb{R}^n} -\ell^*(v) \quad \text{s.t.} \quad |v^\top (Xu_i)_+| \le \beta,\ \ \forall i \in [N+1]. \tag{43}$$

Since (43) and (17) share the same objective, it holds that $d_{s4}^\star < d_{s3}^\star$ if and only if $|v^{\star\top}(Xu_{N+1})_+| > \beta$ with $d_{s4}^\star = d_{s3}^\star$ otherwise. The proof is completed by recalling that $p_{s3}^\star = d_{s3}^\star$ and $p_{s4}^\star = d_{s4}^\star$ due to strong duality. □

C.4 Details about the strong duality between (17) and (14)

C.4.1 General loss functions

In this part of the appendix, we explicitly derive the relationship between the optimal solutions $(\alpha_i^\star)_{i=1}^N$ and $v^\star$ for the purpose of recovering the dual optimizers from the primal optimizers.

The SCP training formulation (14) is equivalent to the following constrained optimization:

$$\min_{r,(\alpha_i)_{i=1}^N} \ell(r,y) + \beta \sum_{i=1}^N |\alpha_i| \qquad \text{s.t.} \quad r = \sum_{i=1}^N \left(Xu_i\right)_+ \alpha_i, \tag{44}$$

and a solution to (14) is also optimal for (44). The optimization (44) is then equivalent to the minimax problem

$$\min_{r,(\alpha_i)_{i=1}^N} \Big(\max_v \ell(r,y) + \beta \sum_{i=1}^N |\alpha_i| + v^\top \Big(\sum_{i=1}^N (Xu_i)_+ \alpha_i - r \Big) \Big). \tag{45}$$

The outer minimization is convex over r and $(\alpha_i)_{i=1}^N$, while the inner maximization is concave over v. Thus, by the Sion's minimax theorem (Sion, 1958), the optimization (45) is equivalent to:

$$\begin{aligned}
&\max_v \Big(\min_r \Big(\ell(r,y) - v^\top r \Big) + \min_{(\alpha_i)_{i=1}^N} \Big(\beta \sum_{i=1}^N |\alpha_j| + v^\top \sum_{i=1}^N (Xu_j)_+ \alpha_j \Big) \Big) \\
=&\max_v \Big(- \max_r \Big(v^\top r - \ell(r,y) \Big) \quad \text{s.t.} \quad \big| v^\top (Xu_i)_+ \big| \le \beta, \ \forall i \in [N] \Big) \\
=&\max_v -\ell^*(v) \quad \text{s.t.} \quad \big| v^\top (Xu_i)_+ \big| \le \beta, \ \forall i \in [N],
\end{aligned}$$

which is (17). The first equality holds because

$$\min_{(\alpha_i)_{i=1}^N} \Big(\beta \sum_{i=1}^N |\alpha_j| + v^\top \sum_{i=1}^N (Xu_j)_+ \alpha_j \Big) = \begin{cases} 0, & \big| v^\top (Xu_i)_+ \big| \le \beta, \ \forall i \in [N], \\ \infty, & \text{otherwise.} \end{cases}$$

Therefore, with the optimal $(\alpha_i^\star)_{i=1}^N$, one can calculate $r^\star$ via $r^\star = \sum_{i=1}^N \big(Xu_i\big)_+ \alpha_i^\star$, and recover $v^\star$ by solving the following LP:

$$v^\star = \arg\max_v -v^\top r^\star \quad \text{s.t.} \quad \big| v^\top (Xu_i)_+ \big| \le \beta, \ \forall i \in [N].$$

C.4.2 Squared loss

In this part, we prove the relationship between $(\alpha_i^\star)_{i=1}^N$ and $v^\star$ by deriving the Karush–Kuhn–Tucker (KKT) conditions for the special case when the squared loss is considered. In this case, the SCP training formulation (14) reduces to

$$\min_{(\alpha_i)_{i=1}^N} \frac{1}{2} \Big\| \sum_{i=1}^N \big(Xu_i\big)_+ \alpha_i - y \Big\|_2^2 + \beta \sum_{i=1}^N |\alpha_i|,$$

which is equivalent to

$$\min_{r,(\alpha_i)_{i=1}^N} \frac{1}{2}\|r\|_2^2 + \beta \sum_{i=1}^N |\alpha_i| \qquad \text{s.t.} \quad r = \sum_{i=1}^N \big(Xu_i\big)_+ \alpha_i - y. \tag{46}$$

By introducing a dual vector variable $v \in \mathbb{R}^n$, we can write the Lagrangian of (46) as:

$$\begin{aligned}
L_{\text{SCP}}\Big(v, r, (\alpha_i)_{i=1}^N\Big) =& \frac{1}{2}\|r\|_2^2 + \beta \sum_{i=1}^N |\alpha_i| + v^\top \Big(\sum_{i=1}^N \big(Xu_i\big)_+ \alpha_i - y - r \Big) \\
=& \Big(\frac{1}{2} r^\top + v^\top \Big) r + \Big(\beta \sum_{i=1}^N |\alpha_i| + v^\top \sum_{i=1}^N \big(Xu_i\big)_+ \alpha_i \Big) + v^\top y
\end{aligned}$$

$L_{\text{SCP}}\big(v, r, (\alpha_i)_{i=1}^N\big)$ is smooth with respect to r. Thus, by the Lagrangian stationarity condition, at optimum, we must have $\nabla_r L\big(v^\star, r^\star, (\alpha_i^\star)_{i=1}^N\big) = r^\star + v^\star = 0$. By the primal feasibility condition, we must have $r^\star = \sum_{i=1}^N \big(Xu_i\big)_+ \alpha_i^\star - y$. Thus, at the optimum, $v^\star = y - \sum_{i=1}^N \big(Xu_i\big)_+ \alpha_i^\star$.

C.5 Proof of Theorem 5

Before proceeding with the proof, we first present the following result borrowed from (Pilanci and Ergen, 2020).

Lemma 10 *For a given data matrix X and $(v_i, w_i)_{i=1}^P$, if $(2D_i - I_n)Xv_i \geq 0$ and $(2D_i - I_n)Xw_i \geq 0$ for all $i \in [P]$, then we can recover the corresponding neural network weights $(u_{v,w_j}, \alpha_{v,w_j})_{j=1}^{m^\star}$ using the formulas in (4), and it holds that*

$$\begin{aligned}&\ell\bigg(\sum_{i=1}^P D_i X(v_i - w_i), y\bigg) + \beta \sum_{i=1}^P \big(\|v_i\|_2 + \|w_i\|_2\big)\\ =&\ell\bigg(\sum_{j=1}^{m^\star} (Xu_{v,w_j})_+ \alpha_{v,w_j}, y\bigg) + \frac{\beta}{2}\sum_{j=1}^{m^\star}\big(\|u_{v,w_j}\|_2^2 + \alpha_{v,w_j}^2\big).\end{aligned} \tag{47}$$

Theorem 1 implies that the non-convex cost function (1) has the same objective value as the following finite-dimensional convex optimization problem:

$$\begin{aligned}q^\star = \min_{(v_i,w_i)_{i=1}^P} \ &\ell\bigg(\sum_{i=1}^P D_i X(v_i - w_i), y\bigg) + \beta \sum_{i=1}^P \big(\|v_i\|_2 + \|w_i\|_2\big)\\ \text{s.t.} \quad &(2D_i - I_n)Xv_i \geq 0,\ (2D_i - I_n)Xw_i \geq 0,\ \ \forall i \in [P]\end{aligned} \tag{48}$$

where $D_1, \ldots, D_P$ are all of the matrices in the set of matrices $\mathcal{D}$, which is defined as the set of all distinct diagonal matrices $\text{diag}([Xu \geq 0])$ that can be obtained for all possible $u \in \mathbb{R}^d$. We recall that the optimal neural network weights can be recovered using (4).

Consider the following optimization problem:

$$\begin{aligned}\widetilde{q}^\star = \min_{(v_i,w_i)_{i=1}^{\widetilde{P}}} \ &\ell\bigg(\sum_{i=1}^{\widetilde{P}} D_i X(v_i - w_i), y\bigg) + \beta \sum_{i=1}^{\widetilde{P}} \big(\|v_i\|_2 + \|w_i\|_2\big)\\ \text{s.t.} \quad &(2D_i - I_n)Xv_i \geq 0,\ (2D_i - I_n)Xw_i \geq 0,\ \ \forall i \in [\widetilde{P}]\end{aligned} \tag{49}$$

where additional D matrices, denoted as $D_{P+1}, \ldots, D_{\widetilde{P}}$, are introduced. These additional matrices are still diagonal with each entry being either 0 or 1, while they do not belong to $\mathcal{D}$. They represent "infeasible hyperplanes" that cannot be achieved by the sign pattern of Xu for any $u \in \mathbb{R}^d$.

Lemma 11 *It holds that $\widetilde{q}^\star = q^\star$, meaning that the optimization problem (49) has the same optimal objective as (48).*

The proof of Lemma 11 is given in Section C.10.

The robust minimax training problem (18) considers an uncertain data matrix $X + \Delta$. Different values of $X + \Delta$ within the perturbation set $\mathcal{U}$ can result in different D matrices. Now, we define $\widehat{\mathcal{D}} = \bigcup_{\Delta} \mathcal{D}_{\Delta}$, where $\mathcal{D}_{\Delta}$ is the set of diagonal matrices for a particular Δ such that $X + \Delta \in \mathcal{U}$. By construction, we have $\mathcal{D}_{\Delta} \subseteq \widehat{\mathcal{D}}$ for every Δ such that $X + \Delta \in \mathcal{U}$. Thus, if we define $D_1, \ldots, D_{\widehat{P}}$ as all matrices in $\widehat{\mathcal{D}}$, then for every Δ with the property $X + \Delta \in \mathcal{U}$, the optimization problem

$$\begin{aligned} \min_{(v_i, w_i)_{i=1}^{\widehat{P}}} \; & \ell\bigg(\sum_{i=1}^{\widehat{P}} D_i(X+\Delta)(v_i - w_i), y \bigg) + \beta \sum_{i=1}^{\widehat{P}} (\|v_i\|_2 + \|w_i\|_2) \\ \text{s.t.} \quad & (2D_i - I_n)(X+\Delta)v_i \geq 0, \; (2D_i - I_n)(X+\Delta)w_i \geq 0, \;\; \forall i \in [\widehat{P}] \end{aligned} \tag{50}$$

is equivalent to

$$\min_{(u_j, \alpha_j)_{j=1}^{m}} \ell\bigg(\sum_{j=1}^{m} ((X+\Delta)u_j)_+ \alpha_j, y \bigg) + \frac{\beta}{2} \sum_{j=1}^{m} \left(\|u_j\|_2^2 + \alpha_j^2 \right)$$

as long as $m \geq \widehat{m}^\star$ with $\widehat{m}^\star = |\{i : v_i^\star(\Delta) \neq 0\}| + |\{i : w_i^\star(\Delta) \neq 0\}|$, where $(v_i^\star(\Delta), w_i^\star(\Delta))_{i=1}^{\widehat{P}}$ denotes an optimal point to (50).

Now, we focus on the minimax training problem with a convex objective given by

$$\min_{(v_i, w_i)_{i=1}^{\widehat{P}} \in \mathcal{F}} \left(\begin{array}{l} \max\limits_{\Delta : X+\Delta \in \mathcal{U}} \ell\bigg(\sum\limits_{i=1}^{\widehat{P}} D_i(X+\Delta)(v_i - w_i), y \bigg) + \beta \sum\limits_{i=1}^{\widehat{P}} \left(\|v_i\|_2 + \|w_i\|_2 \right) \\ \text{s.t. } (2D_i - I_n)(X+\Delta)v_i \geq 0, \; (2D_i - I_n)(X+\Delta)w_i \geq 0, \; \forall i \in [\widehat{P}] \end{array} \right), \tag{51}$$

where $\mathcal{F}$ is defined as:

$$\left\{ (v_i, w_i)_{i=1}^{\widehat{P}} \;\middle|\; \begin{array}{c} \exists \Delta : X + \Delta \in \mathcal{U} \\ \text{s.t. } (2D_i - I_n)(X+\Delta)v_i \geq 0, \; (2D_i - I_n)(X+\Delta)w_i \geq 0, \; \forall i \in [\widehat{P}] \end{array} \right\}.$$

The introduction of the feasible set $\mathcal{F}$ is to avoid the situation where the inner maximization over Δ is infeasible and the objective becomes $-\infty$, leaving the outer minimization problem unbounded.

Moreover, consider the following problem:

$$\begin{aligned} \min_{(v_i, w_i)_{i=1}^{\widehat{P}}} \; & \left(\ell\bigg(\sum_{i=1}^{\widehat{P}} D_i(X + \Delta_{v,w}^\star)(v_i - w_i), y \bigg) + \beta \sum_{i=1}^{\widehat{P}} \left(\|v_i\|_2 + \|w_i\|_2 \right) \right) \\ \text{s.t.} \quad & (2D_i - I_n)(X + \Delta_{v,w}^\star)v_i \geq 0, \; (2D_i - I_n)(X + \Delta_{v,w}^\star)w_i \geq 0, \;\; \forall i \in [\widehat{P}] \end{aligned} \tag{52}$$

where $\Delta_{v,w}^\star$ is the optimal point for $\max\limits_{\Delta : X+\Delta \in \mathcal{U}} \ell\bigg(\sum\limits_{i=1}^{\widehat{P}} D_i(X+\Delta)(v_i - w_i), y \bigg)$. Note that the inequality constraints are dropped for the maximization here compared to (51).

The optimization problem (51) gives a lower bound on (52). To prove this, we first rewrite (52) as:

$$\min_{(v_i,w_i)_{i=1}^{\widehat{P}}} f\big((v_i,w_i)_{i=1}^{\widehat{P}}\big), \text{ where } f\big((v_i,w_i)_{i=1}^{\widehat{P}}\big) =$$
$$\begin{cases} \ell\Big(\sum_{i=1}^{\widehat{P}} D_i(X+\Delta_{v,w}^\star)(v_i-w_i), y\Big) & (2D_i-I_n)(X+\Delta_{v,w}^\star)v_i \geq 0,\ \forall i\in[\widehat{P}] \\ \qquad +\beta\sum_{i=1}^{\widehat{P}}\big(\|v_i\|_2+\|w_i\|_2\big), & (2D_i-I_n)(X+\Delta_{v,w}^\star)w_i \geq 0,\ \forall i\in[\widehat{P}] \\ +\infty, & \text{otherwise.} \end{cases}$$

Now, we analyze (51). Consider three cases:

Case 1: For some $(v_i,w_i)_{i=1}^{\widehat{P}}$, $\Delta_{v,w}^\star$ is optimal for the inner maximization of (51) and the inequality constraints are inactive. This happens whenever $\Delta_{v,w}^\star$ is feasible for the particular choice of $(v_i,w_i)_{i=1}^{\widehat{P}}$. In other words, $(2D_i-I_n)(X+\Delta_{v,w}^\star)v_i \geq 0$ and $(2D_i-I_n)(X+\Delta_{v,w}^\star)w_i \geq 0$ hold true for all $i\in[\widehat{P}]$. For these $(v_i,w_i)_{i=1}^{\widehat{P}}$, we have:

$$\begin{pmatrix} \max\limits_{\Delta: X+\Delta\in\mathcal{U}} \ell\Big(\sum\limits_{i=1}^{\widehat{P}} D_i(X+\Delta)(v_i-w_i), y\Big) + \beta\sum\limits_{i=1}^{\widehat{P}}\big(\|v_i\|_2+\|w_i\|_2\big) \\ \text{s.t. } (2D_i-I_n)(X+\Delta)v_i\geq 0,\ (2D_i-I_n)(X+\Delta)w_i\geq 0,\ \forall i\in[\widehat{P}] \end{pmatrix}$$
$$= \ell\Big(\sum_{i=1}^{\widehat{P}} D_i(X+\Delta_{v,w}^\star)(v_i-w_i), y\Big) + \beta\sum_{i=1}^{\widehat{P}}\big(\|v_i\|_2+\|w_i\|_2\big)$$

Case 2: For some $(v_i,w_i)_{i=1}^{\widehat{P}}$, $\Delta_{v,w}^\star$ is infeasible, while some Δ within the perturbation bound satisfies the inequality constraints. Suppose that among the feasible Δ's,

$$\widetilde{\Delta}_{v,w}^\star = \operatorname*{arg\,max}_{\Delta: X+\Delta\in\mathcal{U}} \ell\Big(\sum_{i=1}^{\widehat{P}} D_i(X+\Delta)(v_i-w_i), y\Big) + \beta\sum_{i=1}^{\widehat{P}}\big(\|v_i\|_2+\|w_i\|_2\big)$$
$$\text{s.t. } (2D_i-I_n)(X+\Delta)v_i\geq 0,\ (2D_i-I_n)(X+\Delta)w_i\geq 0,\ \forall i\in[\widehat{P}].$$

In this case,

$$\begin{pmatrix} \max\limits_{\Delta: X+\Delta\in\mathcal{U}} \ell\Big(\sum\limits_{i=1}^{\widehat{P}} D_i(X+\Delta)(v_i-w_i), y\Big) + \beta\sum\limits_{i=1}^{\widehat{P}}\big(\|v_i\|_2+\|w_i\|_2\big) \\ \text{s.t. } (2D_i-I_n)(X+\Delta)v_i\geq 0,\ (2D_i-I_n)(X+\Delta)w_i\geq 0,\ \forall i\in[\widehat{P}] \end{pmatrix}$$
$$= \ell\Big(\sum_{i=1}^{\widehat{P}} D_i(X+\widetilde{\Delta}_{v,w}^\star)(v_i-w_i), y\Big) + \beta\sum_{i=1}^{\widehat{P}}\big(\|v_i\|_2+\|w_i\|_2\big)$$

Case 3: For all other $(v_i,w_i)_{i=1}^{\widehat{P}}$, the objective value is $+\infty$ since they do not belong to $\mathcal{F}$.

Therefore, (51) can be rewritten as

$$\min_{(v_i,w_i)_{i=1}^{\widehat{P}}} g\big((v_i,w_i)_{i=1}^{\widehat{P}}\big), \text{ where } g\big((v_i,w_i)_{i=1}^{\widehat{P}}\big) =$$
$$\begin{cases} \ell\Big(\sum_{i=1}^{\widehat{P}} D_i(X+\Delta^\star_{v,w})(v_i-w_i), y\Big) & (2D_i-I_n)(X+\Delta^\star_{v,w})v_i \geq 0,\ \forall i\in[\widehat{P}] \\ \quad +\beta\sum_{i=1}^{\widehat{P}}\big(\|v_i\|_2+\|w_i\|_2\big), & (2D_i-I_n)(X+\Delta^\star_{v,w})w_i \geq 0,\ \forall i\in[\widehat{P}] \\ & \exists j: (2D_j-I_n)(X+\Delta^\star_{v,w})v_j<0 \\ \ell\Big(\sum_{i=1}^{\widehat{P}} D_i(X+\widetilde{\Delta}^\star_{v,w})(v_i-w_i), y\Big) & \text{ or } (2D_j-I_n)(X+\Delta^\star_{v,w})w_j<0 \\ \quad +\beta\sum_{i=1}^{\widehat{P}}\big(\|v_i\|_2+\|w_i\|_2\big), & \exists\Delta:\ (2D_i-I_n)(X+\Delta)v_i\geq 0,\ \forall i\in[\widehat{P}] \\ & \qquad (2D_i-I_n)(X+\Delta)w_i\geq 0,\ \forall i\in[\widehat{P}] \\ +\infty, & \text{otherwise} \end{cases}$$

Hence, $g((v_i,w_i)_{i=1}^{\widehat{P}}) = f((v_i,w_i)_{i=1}^{\widehat{P}})$ for all $(v_i,w_i)_{i=1}^{\widehat{P}}$ belonging to the first and the third cases. $g((v_i,w_i)_{i=1}^{\widehat{P}}) < f((v_i,w_i)_{i=1}^{\widehat{P}})$ for all $(v_i,w_i)_{i=1}^{\widehat{P}}$ belonging to the second case. Thus, $\min_{(v_i,w_i)_{i=1}^{\widehat{P}}} g((v_i,w_i)_{i=1}^{\widehat{P}}) \leq \min_{(v_i,w_i)_{i=1}^{\widehat{P}}} f((v_i,w_i)_{i=1}^{\widehat{P}})$. This concludes that (51) is a lower bound to (52).

Let $(v^\star_{\text{minimax}_i}, w^\star_{\text{minimax}_i})_{i=1}^{\widehat{P}}$ denote an optimal point for (52). It is possible that for some Δ : $X+\Delta\in\mathcal{U}$, the constraints $(2D_i-I_n)(X+\Delta)v^\star_{\text{minimax}_i}\geq 0$ and $(2D_i-I_n)(X+\Delta)w^\star_{\text{minimax}_i}\geq 0$ are not satisfied for all $i\in[\widehat{P}]$. In light of Lemma 10, at those Δ where such constraints are violated, the convex problem (52) does not reflect the cost of the neural network. For these infeasible Δ, the input-label pairs $(X+\Delta, y)$ can have a high cost in the neural network and potentially become the worst-case adversary. However, these Δ are ignored in (52) due to the infeasibility. Since adversarial training aims to minimize the cost over the worst-case adversaries generated upon the training data whereas (52) may sometimes miss the worst-case adversaries, (52) does not fully accomplish the task of adversarial training. In fact, by applying Theorem 1 and Lemma 11, it can be verified that (51) and (52) are lower bounds to (18) as long as $m\geq\widehat{m}^\star$:

$$\begin{aligned} &\min_{(u_j,\alpha_j)_{j=1}^m}\left(\max_{\Delta:X+\Delta\in\mathcal{U}}\ell\Big(\sum_{j=1}^m\big((X+\Delta)u_j\big)_+\alpha_j,y\Big)+\frac{\beta}{2}\sum_{j=1}^m\Big(\|u_j\|_2^2+\alpha_j^2\Big)\right) \\ &\quad\geq \min_{(u_j,\alpha_j)_{j=1}^m}\ell\Big(\sum_{j=1}^m\big((X+\Delta^\star_{v,w})u_j\big)_+\alpha_j,y\Big)+\frac{\beta}{2}\sum_{j=1}^m\Big(\|u_j\|_2^2+\alpha_j^2\Big) \\ &\quad= \begin{pmatrix}\min\limits_{(v_i,w_i)_{i=1}^{\widehat{P}}}\ell\Big(\sum\limits_{i=1}^{\widehat{P}}D_i(X+\Delta^\star_{v,w})(v_i-w_i),y\Big)+\beta\sum\limits_{i=1}^{\widehat{P}}\big(\|v_i\|_2+\|w_i\|_2\big)\\ \text{s.t. } (2D_i-I_n)(X+\Delta^\star_{v,w})v_i\geq 0,\ (2D_i-I_n)(X+\Delta^\star_{v,w})w_i\geq 0,\ \forall i\in[\widehat{P}]\end{pmatrix}. \end{aligned}$$

To address the feasibility issue, we can apply robust optimization techniques ((Boyd and Vandenberghe, 2004) section 4.4.2) and replace the constraints in (52) with robust convex

constraints, which will lead to (21). Let $\big((v^\star_{\mathrm{rob}_i}, w^\star_{\mathrm{rob}_i})_{i=1}^{\widehat{P}}, \Delta^\star_{\mathrm{rob}}\big)$ denote an optimal point of (21) and let $(u^\star_{\mathrm{rob}_j}, \alpha^\star_{\mathrm{rob}_j})_{j=1}^{\widehat{m}^\star}$ be the neural network weights recovered from $(v^\star_{\mathrm{rob}_i}, w^\star_{\mathrm{rob}_i})_{i=1}^{\widehat{P}}$ with (4), where $\widehat{m}^\star$ is the number of nonzero weights. In light of Lemma 10, since the constraints $(2D_i - I_n)(X+\Delta)v^\star_{\mathrm{rob}_i} \geq 0$ and $(2D_i - I_n)(X+\Delta)w^\star_{\mathrm{rob}_i} \geq 0$ for all $i \in [\widehat{P}]$ apply to all $X + \Delta \in \mathcal{U}$, all $X + \Delta \in \mathcal{U}$ satisfy the equality

$$\ell\Big(\sum_{i=1}^{\widehat{P}} D_i(X+\Delta)(v^\star_{\mathrm{rob}_i} - w^\star_{\mathrm{rob}_i}), y\Big) + \beta\sum_{i=1}^{\widehat{P}}\big(\|v^\star_{\mathrm{rob}_i}\|_2 + \|w^\star_{\mathrm{rob}_i}\|_2\big)$$
$$= \ell\Big(\sum_{j=1}^{\widehat{m}^\star}\big((X+\Delta)u^\star_{\mathrm{rob}_j}\big)_+\alpha^\star_{\mathrm{rob}_j}, y\Big) + \frac{\beta}{2}\sum_{j=1}^{\widehat{m}^\star}\big(\|u^\star_{\mathrm{rob}_j}\|_2^2 + \alpha^{\star 2}_{\mathrm{rob}_j}\big).$$

Thus, since

$$\Delta^\star_{\mathrm{rob}} = \underset{\Delta: X+\Delta\in\mathcal{U}}{\arg\max}\ \ell\Big(\sum_{i=1}^{\widehat{P}} D_i(X+\Delta)(v^\star_{\mathrm{rob}_i} - w^\star_{\mathrm{rob}_i}), y\Big) + \beta\sum_{i=1}^{\widehat{P}}\big(\|v^\star_{\mathrm{rob}_i}\|_2 + \|w^\star_{\mathrm{rob}_i}\|_2\big),$$

we have

$$\Delta^\star_{\mathrm{rob}} = \underset{\Delta: X+\Delta\in\mathcal{U}}{\arg\max}\ \ell\Big(\sum_{j=1}^{\widehat{m}^\star}\big((X+\Delta)u^\star_{\mathrm{rob}_j}\big)_+\alpha^\star_{\mathrm{rob}_j}, y\Big) + \frac{\beta}{2}\sum_{j=1}^{\widehat{m}^\star}\big(\|u^\star_{\mathrm{rob}_j}\|_2^2 + \alpha^{\star 2}_{\mathrm{rob}_j}\big),$$

giving rise to:

$$\ell\Big(\sum_{i=1}^{\widehat{P}} D_i(X+\Delta^\star_{\mathrm{rob}})(v^\star_{\mathrm{rob}_i} - w^\star_{\mathrm{rob}_i}), y\Big) + \beta\sum_{i=1}^{\widehat{P}}\big(\|v^\star_{\mathrm{rob}_i}\|_2 + \|w^\star_{\mathrm{rob}_i}\|_2\big)$$
$$= \ell\Big(\sum_{j=1}^{\widehat{m}^\star}\big((X+\Delta^\star_{\mathrm{rob}})u^\star_{\mathrm{rob}_j}\big)_+\alpha^\star_{\mathrm{rob}_j}, y\Big) + \frac{\beta}{2}\sum_{j=1}^{\widehat{m}^\star}\big(\|u^\star_{\mathrm{rob}_j}\|_2^2 + \alpha^{\star 2}_{\mathrm{rob}_j}\big)$$
$$= \max_{\Delta: X+\Delta\in\mathcal{U}} \ell\Big(\sum_{j=1}^{\widehat{m}^\star}\big((X+\Delta)u^\star_{\mathrm{rob}_j}\big)_+\alpha^\star_{\mathrm{rob}_j}, y\Big) + \frac{\beta}{2}\sum_{j=1}^{\widehat{m}^\star}\big(\|u^\star_{\mathrm{rob}_j}\|_2^2 + \alpha^{\star 2}_{\mathrm{rob}_j}\big)$$
$$\geq \min_{(u_j,\alpha_j)_{j=1}^{\widehat{m}^\star}} \left(\max_{\Delta: X+\Delta\in\mathcal{U}} \ell\Big(\sum_{j=1}^{\widehat{m}^\star}\big((X+\Delta)u_j\big)_+\alpha_j, y\Big) + \frac{\beta}{2}\sum_{j=1}^{\widehat{m}^\star}\big(\|u_j\|_2^2 + \alpha_j^2\big)\right)$$

Therefore, (21) is an upper bound to (18). □

C.6 Proof of Corollary 6

Define $E_i = 2D_i - I_n$ for all $i \in [\widehat{P}]$. Note that each E_i is a diagonal matrix, and its diagonal elements are either -1 or 1. Therefore, for each $i \in [\widehat{P}]$, we can analyze the robust constraint $\min_{\Delta: X+\Delta\in\mathcal{U}} E_i(X+\Delta)v_i \geq 0$ element-wise (for each data point). Let e_{ik} denote the k^{th}

diagonal element of E_i and $\delta_{ik}^\top$ denote the k^{th} element of Δ that appears in the i^{th} constraint. We then have:

$$\Big(\min_{\|\delta_{ik}\|_\infty \le \epsilon} e_{ik}(x_k^\top + \delta_{ik}^\top)v_i\Big) = \Big(e_{ik}x_k^\top v_i + \min_{\|\delta_{ik}\|_\infty \le \epsilon} e_{ik}\delta_{ik}^\top v_i\Big) \ge 0 \tag{53}$$

The minima of the above optimization problems are achieved at $\delta_{ik}^{\star\star} = \epsilon \cdot \text{sgn}(e_{ik}v_i) = \epsilon \cdot e_{ik} \cdot \text{sgn}(v_i)$.

Note that as ϵ approaches 0, $\delta_{ik}^{\star\star}$ and $\Delta_{\text{rob}}^\star$ in Theorem 5 both approach 0, which means that the gap between the convex robust problem (27) and the non-convex adversarial training problem (25) diminishes. Substituting $\delta_k^{\star\star}$ into (53) yields that

$$\Big(e_{ik}x_k^\top v_i - \epsilon\|e_{ik}v_i\|_1\Big) = \Big(e_{ik}x_k^\top v_i - \epsilon\|v_i\|_1\Big) \ge 0.$$

Vertically concatenating $e_{ik}x_k^\top v_i - \epsilon\|v_i\|_1 \ge 0$ for all $i \in [\widehat{P}]$ gives the vectorized representation $E_i X v_i - \epsilon\|v_i\|_1 \ge 0$, which leads to (22). Since the constraints on w are exactly the same, we also have that $\min_{\Delta: X+\Delta\in\mathcal{U}} E_i(X+\Delta)w_i \ge 0$ is equivalent to $E_iXw_i - \epsilon\|w_i\|_1 \ge 0$ for all $i \in [\widehat{P}]$.

C.7 Proof of Theorem 7

The regularization term is independent of Δ. Thus, it can be ignored for the purpose of analyzing the inner maximization. Note that each D_i is diagonal, and its diagonal elements are either 0 or 1. Therefore, the inner maximization of (26) can be analyzed element-wise (cost of each data point).

The maximization problem of the loss at each data point is:

$$\max_{\|\delta_k\|_\infty \le \epsilon} \Big(1 - y_k \sum_{i=1}^P d_{ik}(x_k^\top + \delta_k^\top)(v_i - w_i)\Big)_+ \tag{54}$$

where d_{ik} is the k^{th} diagonal element of D_i and $\delta_k^\top$ is the k^{th} row of Δ. One can write:

$$\begin{aligned}
&\max_{\|\delta_k\|_\infty \le \epsilon} \Big(1 - y_k \sum_{i=1}^P d_{ik}(x_k^\top + \delta_k^\top)(v_i - w_i)\Big)_+ \\
&\qquad = \Big(\max_{\|\delta_k\|_\infty \le \epsilon} 1 - y_k \sum_{i=1}^P d_{ik}(x_k^\top + \delta_k^\top)(v_i - w_i)\Big)_+ \\
&\qquad = \Big(1 - y_k \sum_{i=1}^P d_{ik}x_k^\top(v_i - w_i) - \min_{\|\delta_k\|_\infty \le \epsilon} \delta_k^\top y_k \sum_{i=1}^P d_{ik}(v_i - w_i)\Big)_+ .
\end{aligned}$$

The optimal solution to $\min_{\|\delta_k\|_\infty \le \epsilon} \delta_k^\top y_k \sum_{i=1}^P d_{ik}(v_i - w_i)$ is $\delta_{\text{hinge}_k}^\star = -\epsilon \cdot \text{sgn}\Big(y_k \sum_{i=1}^P d_{ik}(v_i - w_i)^\top\Big)$, or equivalently:

$$\Delta_{\text{hinge}}^\star = -\epsilon \cdot \text{sgn}\Big(\sum_{i=1}^P D_i y (v_i - w_i)^\top\Big).$$

By substituting $\delta^\star_{\text{hinge}_k}$ into (54), the optimization problem (54) reduces to:

$$\left(1 - y_k \sum_{i=1}^{P} d_{ik} x_k^\top (v_i - w_i) + \epsilon \left\| y_k \sum_{i=1}^{P} d_{ik}(v_i - w_i) \right\|_1 \right)_+ \\ = \left(1 - y_k \sum_{i=1}^{P} d_{ik} x_k^\top (v_i - w_i) + \epsilon |y_k| \left\| \sum_{i=1}^{P} d_{ik}(v_i - w_i) \right\|_1 \right)_+ .$$

Therefore, the overall loss function is:

$$\frac{1}{n} \sum_{k=1}^{n} \left(1 - y_k \sum_{i=1}^{P} d_{ik} x_k^\top (v_i - w_i) + \epsilon |y_k| \left\| \sum_{i=1}^{P} d_{ik}(v_i - w_i) \right\|_1 \right)_+ .$$

In the case of binary classification, $y = \{-1, 1\}^n$, and thus $|y_k| = 1$ for all $k \in [n]$. Therefore, the above is equivalent to

$$\frac{1}{n} \sum_{k=1}^{n} \left(1 - y_k \sum_{i=1}^{P} d_{ik} x_k^\top (v_i - w_i) + \epsilon \left\| \sum_{i=1}^{P} d_{ik}(v_i - w_i) \right\|_1 \right)_+ \tag{55}$$

which is the objective of (27). This completes the proof. □

C.8 Proof of Theorem 8

We first exploit the structure of (30) and reformulate it as the following robust second-order cone program (SOCP) by introducing a slack variable $a \in \mathbb{R}$:

$$\begin{aligned} \min_{(v_i, w_i)_{i=1}^{\widehat{P}}, a} \quad & a + \beta \sum_{i=1}^{\widehat{P}} (\|v_i\|_2 + \|w_i\|_2) \\ \text{s.t.} \quad & (2D_i - I_n) X v_i \geq \epsilon \|v_i\|_1, \;\; (2D_i - I_n) X w_i \geq \epsilon \|w_i\|_1, \quad \forall i \in [\widehat{P}] \\ & \max_{\Delta : X + \Delta \in \mathcal{X}} \left\| \begin{bmatrix} \sum_{i=1}^{\widehat{P}} D_i (X + \Delta)(v_i - w_i) - y \\ 2a - \frac{1}{4} \end{bmatrix} \right\|_2 \leq 2a + \tfrac{1}{4}, \quad \forall i \in [\widehat{P}]. \end{aligned} \tag{56}$$

Then, we need to establish the equivalence between (56) and (31). To this end, we consider the constraints of (56) and argue that these can be recast as the constraints given in (31).

One can write:

$$\max_{\Delta: X+\Delta\in\mathcal{X}} \left\| \begin{bmatrix} \sum_{i=1}^{\widehat{P}} D_i(X+\Delta)(v_i-w_i)-y \\ 2a-\frac{1}{4} \end{bmatrix} \right\|_2 \leq 2a+\frac{1}{4}$$

$$\iff \max_{\|\delta_k\|_\infty\leq\epsilon,\ \forall k\in[n]} \left\| \begin{bmatrix} \sum_{i=1}^{\widehat{P}} d_{i1}(x_1^\top-\delta_1^\top)(v_i-w_i)-y_1 \\ \sum_{i=1}^{\widehat{P}} d_{i2}(x_2^\top-\delta_2^\top)(v_i-w_i)-y_2 \\ \vdots \\ \sum_{i=1}^{\widehat{P}} d_{in}(x_n^\top-\delta_n^\top)(v_i-w_i)-y_n \\ 2a-\frac{1}{4} \end{bmatrix} \right\|_2 \leq 2a+\frac{1}{4}$$

$$\iff \max_{\|\delta_k\|_\infty\leq\epsilon,\ \forall k\in[n]} \left(\sum_{k=1}^{n} \Big(\sum_{i=1}^{\widehat{P}} d_{ik}(x_k^\top-\delta_k^\top)(v_i-w_i)-y_k \Big)^2 + \Big(2a-\frac{1}{4}\Big)^2 \right)^{\frac{1}{2}} \leq 2a+\frac{1}{4}$$

where d_{ik} is the k^{th} diagonal element of D_i and $\delta_k^\top$ is the k^{th} row of Δ. The above constraints can be rewritten by introducing slack variables $z\in\mathbb{R}^{n+1}$ as

$$\begin{aligned} &z_k \geq \Big| \textstyle\sum_{i=1}^{\widehat{P}} d_{ik}x_k^\top(v_i-w_i)-y_k \Big| + \epsilon \Big\| \textstyle\sum_{i=1}^{\widehat{P}} d_{ik}(v_i-w_i) \Big\|_1,\ \forall k\in[n] \\ &z_{n+1} \geq \big|2a-\tfrac{1}{4}\big|, \quad \|z\|_2 \leq 2a+\tfrac{1}{4}. \end{aligned}$$

□

C.9 Proof of Theorem 9

The inner maximization of (33) can be analyzed separately for each y_k. For every index k such that $y_k=0$, it holds that $\sum_{k=1}^n \big(-2\widehat{y}_k y_k+\log(e^{2\widehat{y}_k}+1)\big)$ monotonously increases with respect to $\widehat{y}_k$. Thus, we need to find δ_k that maximizes $\widehat{y}_k$ in order to maximize the objective. Therefore, the worst-case adversary $\delta_k^\star$ is

$$\delta^\star_{k:y_k=0} = \operatorname*{arg\,max}_{\|\delta_k\|_\infty\leq\epsilon} \left(\sum_{i=1}^{\widehat{P}} d_{ik}\delta_k^\top(v_i-w_i) \right) = \epsilon\cdot\operatorname{sgn}\Big(\sum_{i=1}^{\widehat{P}} d_{ik}(v_i-w_i)^\top\Big). \tag{57}$$

For each index k such that $y_k=1$, it holds that $\sum_{k=1}^n \big(-2\widehat{y}_k y_k+\log(e^{2\widehat{y}_k}+1)\big)$ monotonously decreases with respect to $\widehat{y}_k$. Thus, we need to minimize $\widehat{y}_k$. Therefore,

$$\delta^\star_{k:y_k=1} = \operatorname*{arg\,min}_{\|\delta_k\|_\infty\leq\epsilon} \left(\sum_{i=1}^{\widehat{P}} d_{ik}\delta_k^\top(v_i-w_i) \right) = -\epsilon\cdot\operatorname{sgn}\Big(\sum_{i=1}^{\widehat{P}} d_{ik}(v_i-w_i)^\top\Big). \tag{58}$$

The two cases can be combined as $\delta_k^\star = -\epsilon\cdot\operatorname{sgn}\Big((2y_k-1)\sum_{i=1}^{\widehat{P}} d_{ik}(v_i-w_i)^\top\Big)$. Concatenating $\delta_1^\star,\ldots,\delta_n^\star$ back into the matrix form yields the worst-case perturbation matrix $\Delta^\star_{\text{BCE}} = -\epsilon\cdot\operatorname{sgn}\Big((2y-1)\sum_{i=1}^{\widehat{P}} D_i(v_i-w_i)^\top\Big)$.

Moreover, notice that the objective is separable based on those k such that $y_k = 0$ and those k such that $y_k = 1$:

$$\begin{aligned}
&\sum_{k=1}^{n} \Big(-2\widehat{y}_k y_k + \log(e^{2\widehat{y}_k}+1)\Big) \\
&\quad= \sum_{k:y_k=1} \Big(-2\widehat{y}_k + \log(e^{2\widehat{y}_k}+1)\Big) + \sum_{k:y_k=0} \log\big(e^{2\widehat{y}_k}+1\big) \\
&\quad= \sum_{k:y_k=1} \log\Big(\frac{e^{2\widehat{y}_k}+1}{e^{2\widehat{y}_k}}\Big) + \sum_{k:y_k=0} \log\big(e^{2\widehat{y}_k}+1\big) \\
&\quad= \sum_{k:y_k=1} \log\big(e^{-2\widehat{y}_k}+1\big) + \sum_{k:y_k=0} \log\big(e^{2\widehat{y}_k}+1\big) \\
&\quad= \sum_{k:y_k=1} \log\Bigg(\exp\Big(-2\sum_{i=1}^{\widehat{P}} d_{ik} x_k^\top (v_i - w_i) + 2\epsilon \cdot \Big\| \sum_{i=1}^{\widehat{P}} d_{ik}(v_i - w_i)\Big\|_1\Big) + 1\Bigg) \quad (59) \\
&\qquad+ \sum_{k:y_k=0} \log\Bigg(\exp\Big(2\sum_{i=1}^{\widehat{P}} d_{ik} x_k^\top (v_i - w_i) + 2\epsilon \cdot \Big\| \sum_{i=1}^{\widehat{P}} d_{ik}(v_i - w_i)\Big\|_1\Big) + 1\Bigg) \quad (60) \\
&\quad= \sum_{k=1}^{n} \log\Bigg(\exp\Big(2\Big((2y_k-1)\sum_{i=1}^{\widehat{P}} d_{ik} x_k^\top (v_i - w_i) + \epsilon \cdot \Big\| \sum_{i=1}^{\widehat{P}} d_{ik}(v_i - w_i)\Big\|_1\Big)\Big) + 1\Bigg) \\
&\quad= \sum_{k=1}^{n} f \circ g_k\big(\{v_i, w_i\}_{i=1}^{\widehat{P}}\big)
\end{aligned}$$

where (59) and (60) are obtained by substituting in (57) and (58), and $f(\cdot)$, $g(\cdot)$ are defined in (34). Substituting the term $\sum_{k=1}^{n} \big(-2\widehat{y}_k y_k + \log(e^{2\widehat{y}_k}+1)\big)$ in (33) with the term $\sum_{k=1}^{n} f \circ g_k\big(\{v_i, w_i\}_{i=1}^{\widehat{P}}\big)$ yields the formulation (34). Since the function $f(\cdot)$ is convex non-decreasing and $g(\cdot)$ is convex, the optimization (34) is convex. □

C.10 Proof of Lemma 11

According to (Pilanci and Ergen, 2020), recovering the neural network weights by substituting (4) into (48) leads to

$$\begin{aligned}
q^\star &= \min_{(v_i,w_i)_{i=1}^{P}} \ell\Bigg(\sum_{i=1}^{P} D_i X (v_i - w_i), y\Bigg) + \beta \sum_{i=1}^{P} \Big(\|v_i\|_2 + \|w_i\|_2\Big) \\
&= \min_{(u_j,\alpha_j)_{j=1}^{m^\star}} \ell\Bigg(\sum_{j=1}^{m^\star} (Xu_j)_+ \alpha_j, y\Bigg) + \frac{\beta}{2} \sum_{j=1}^{m^\star} \Big(\|u_j\|_2^2 + \alpha_j^2\Big)
\end{aligned}$$

Similarly, we can recover the network weights from the solution $(\widetilde{v}_i^\star, \widetilde{w}_i^\star)_{i=1}^{\widetilde{P}}$ of (49) using

$$(\widetilde{u}_{j_{1i}}, \widetilde{\alpha}_{j_{1i}}) = \Bigg(\frac{\widetilde{v}_i^\star}{\sqrt{\|\widetilde{v}_i^\star\|_2}}, \sqrt{\|\widetilde{v}_i^\star\|_2}\Bigg), \;\; (\widetilde{u}_{j_{2i}}, \widetilde{\alpha}_{j_{2i}}) = \Bigg(\frac{\widetilde{w}_i^\star}{\sqrt{\|\widetilde{w}_i^\star\|_2}}, -\sqrt{\|\widetilde{w}_i^\star\|_2}\Bigg), \;\; \forall i \in [\widetilde{P}]. \quad (61)$$

Unlike in (4), zero weights are not discarded in (61). For simplicity, we use $\widetilde{u}_1, \ldots, \widetilde{u}_{\widetilde{m}^\star}$ to refer to the hidden layer weights and use $\widetilde{\alpha}_1, \ldots, \widetilde{\alpha}_{\widetilde{m}^\star}$ to refer to the output layer weights recovered using (61). Since $(\widetilde{v}_i^\star, \widetilde{w}_i^\star)_{i=1}^{\widetilde{P}}$ is a solution to (49), it satisfies $(2D_i - I_n)X\widetilde{v}_i^\star \geq 0$ and $(2D_i - I_n)X\widetilde{w}_i^\star \geq 0$ for all $i \in [\widetilde{P}]$. Thus, we can apply Lemma 10 to obtain:

$$\begin{aligned}
\widetilde{q}^\star =& \ell\bigg(\sum_{i=1}^{\widetilde{P}} D_i X(\widetilde{v}_i^\star - \widetilde{w}_i^\star), y\bigg) + \beta \sum_{i=1}^{\widetilde{P}} \Big(\|\widetilde{v}_i^\star\|_2 + \|\widetilde{w}_i^\star\|_2\Big) \\
=& \ell\bigg(\sum_{j=1}^{\widetilde{m}^\star} (X\widetilde{u}_j^\star)_+ \alpha_j, y\bigg) + \frac{\beta}{2} \sum_{j=1}^{\widetilde{m}^\star} \Big(\|\widetilde{u}_j^\star\|_2^2 + \widetilde{\alpha}_j^{\star 2}\Big) \\
\geq & \min_{(u_j, \alpha_j)_{j=1}^{\widetilde{m}^\star}} \ell\bigg(\sum_{j=1}^{\widetilde{m}^\star} (Xu_j)_+ \alpha_j, y\bigg) + \frac{\beta}{2} \sum_{j=1}^{\widetilde{m}^\star} \Big(\|u_j\|_2^2 + \alpha_j^2\Big)
\end{aligned}$$

Since $\widetilde{P} \geq P$, $m^\star \leq 2P$ and $\widetilde{m}^\star = 2\widetilde{P}$, we have $\widetilde{m}^\star \geq m^\star$. Therefore, according to Section 2 and Theorem 6 of (Pilanci and Ergen, 2020), we have:

$$\begin{aligned}
q^\star =& \min_{(u_j, \alpha_j)_{j=1}^{m^\star}} \ell\bigg(\sum_{j=1}^{m^\star} (Xu_j)_+ \alpha_j, y\bigg) + \frac{\beta}{2} \sum_{j=1}^{m^\star} \Big(\|u_j\|_2^2 + \alpha_j^2\Big) \\
=& \min_{(u_j, \alpha_j)_{j=1}^{\widetilde{m}^\star}} \ell\bigg(\sum_{j=1}^{\widetilde{m}^\star} (Xu_j)_+ \alpha_j, y\bigg) + \frac{\beta}{2} \sum_{j=1}^{\widetilde{m}^\star} \Big(\|u_j\|_2^2 + \alpha_j^2\Big) \leq \widetilde{q}^\star.
\end{aligned}$$

The above inequality $q^\star \leq \widetilde{q}^\star$ shows that a neural network with more than m neurons in the hidden layer will yield the same loss as the neural network with m neurons when optimized.

Note that (49) can always attain $q^\star$ by simply substituting in the optimal solution of (48) and assigning zeros to all other additional v_i and w_i, implying that $q^\star \geq \widetilde{q}^\star$. Since $q^\star$ is both an upper bound and a lower bound on $\widetilde{q}^\star$, we have $\widetilde{q}^\star = q^\star$, proving that as long as all matrices in $\mathcal{D}$ are included, the existence of redundant matrices does not change the optimal objective value. □