SIAM J. MATH. DATA SCI.
Vol. 5, No. 2, pp. 446–474

Efficient Global Optimization of Two-Layer ReLU Networks: Quadratic-Time Algorithms and Adversarial Training*

Yatong Bai†, Tanmay Gautam‡, and Somayeh Sojoudi§

Abstract. The nonconvexity of the artificial neural network (ANN) training landscape brings optimization difficulties. While the traditional back-propagation stochastic gradient descent algorithm and its variants are effective 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 a ReLU-activated ANN 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 an 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 multipliers. It can solve both the exact convex formulation and the approximate counterpart, and it generalizes to a family of convex training formulations. Linear global convergence is achieved, and the initial several iterations often yield a solution with high prediction accuracy. When solving the approximate formulation, the per-iteration time complexity is quadratic. The second algorithm, based on the "sampled convex programs" theory, is simpler to implement. It solves unconstrained convex formulations and converges to an approximately globally optimal classifier. The nonconvexity 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 inputs. Our analysis explicitly focuses on one-hidden-layer fully connected ANNs, but can extend to more sophisticated architectures.

Key words. robust optimization, convex optimization, adversarial training, neural networks

MSC codes. 68Q25, 82C32, 49M29, 46N10, 62M45

DOI. 10.1137/21M1467134

1. Introduction.

The artificial neural network (ANN) is one of the most powerful and popular machine learning tools. Optimizing a typical ANN with nonlinear activation functions and a finite width requires solving nonconvex optimization problems. Traditionally, training ANNs relies on stochastic gradient descent (SGD) back-propagation [45]. Despite its tremendous empirical success, this algorithm is only guaranteed to converge to a local minimum when applied to the nonconvex ANN training objective. While SGD back-propagation

*Received by the editors January 6, 2022; accepted for publication (in revised form) October 24, 2022; published electronically June 1, 2023.
https://doi.org/10.1137/21M1467134
Funding: This work was supported by grants from ONR and NSF.

†Department of Mechanical Engineering, University of California, Berkeley, Berkeley, CA 94720 USA (yatong_bai@berkeley.edu).
‡Department of Electrical Engineering and Computer Science, University of California, Berkeley, Berkeley, CA 94720 USA (tgautam23@berkeley.edu).
§Department of Mechanical Engineering and Department of Electrical Engineering and Computer Science, University of California, Berkeley, Berkeley, CA 94720 USA (sojoudi@berkeley.edu).

can converge to a global optimizer for one-hidden-layer "rectified linear unit (ReLU)" activated networks when the considered network is wide enough [35, 17] or when the inputs follow a Gaussian distribution [14], spurious local minima can exist in general applications. Moreover, the nonconvexity of the training landscape and the properties of back-propagation SGD cause the issues listed below:

- **Poor interpretability:** With SGD, 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, and the local minimum may be spurious.
- **High sensitivity to hyperparameters:** Back-propagation SGD has several important hyperparameters to tune. Every parameter is crucial to the performance, but selecting the parameters can be difficult. SGD is also sensitive to the initialization [27].
- **Vanishing/exploding gradients:** With back-propagation, the gradient at shallower layers can be tiny (or huge) if the deeper layer weights are tiny (or huge).

While more advanced back-propagation optimizers such as Adam [32] can alleviate the above issues, avoiding them entirely can be hard. Since convex programs possess the desirable property that all local minima are global, the existing works have considered convexifying the ANN training problem [12, 8, 6]. More recently, Pilanci and Ergen proposed "convex training" and derived a convex optimization problem with the same global minimum as the nonconvex cost function of a one-hidden-layer fully connected ReLU ANN, enabling global ANN optimization [42]. The favorable properties of convex optimization make convex training immune to the deficiencies of back-propagation. Convex training also extends to more complex ANNs such as convolutional neural networks [21], deeper networks [20], and vector-output networks [46]. This work begins with one-hidden-layer ANNs for simplicity and extends to a family of convex ANN training formulations, including the results for two-hidden-layer subnetworks [20, 22] and one-hidden-layer networks with batch normalization [23]. Due to space restrictions, the extensions are presented in section SM1. Moreover, [11] designed a layerwise training scheme that concatenates one-hidden-layer ANNs into a deep network, where each layer provably reduces the training error. This approach can be combined with this work, ultimately leading toward training deep networks with convex optimization.

Unfortunately, the $\mathcal{O}(d^3r^3(\frac{n}{r})^{3r})$ computational complexity of the convex training formulation introduced in [42] is prohibitively high. This complexity arises due to the following two reasons:

- The size of the convex program grows exponentially in the training data matrix rank r. This exponential relationship is inherent due to the large number of possible ReLU activation patterns and thus can be hard to reduce. Fortunately, this problem is not a deal-breaker in practice: [42] has shown that a heuristic approximation that forms much smaller convex optimizations performs 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 data points n.
- The convex training formulation is constrained. The naive choice of algorithm for solving a general constrained convex optimization is often the interior-point method (IPM), which has a cubic per-step computational complexity. This paper develops more efficient algorithms that exploit the problem structure and achieve lower

Table 1

Comparisons between the proposed ANN training methods and related methods. The middle column is the per-epoch complexity 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. †: Toward the theoretically minimum loss—further increasing network width will not reduce the training loss; §: Toward a fixed desired level of suboptimality in the sense defined in Theorem 2.2; ‡: For an arbitrary network width m. Since there exists a globally optimal neural network with no more than $n+1$ active hidden-layer neurons [35], the $\mathcal{O}(mnd)$ bound for SGD back-propagation evaluates to $\mathcal{O}(n^2d)$.

Method	Complexity	Global convergence
IPM [42]	$\mathcal{O}\left(d^3r^3(\frac{n}{r})^{3r}\right)^{\dagger}$	Superlinear to the global optimum.
ADMM (exact)	$\mathcal{O}\left(d^2r^2(\frac{n}{r})^{2r}\right)^{\dagger}$	Rapid to a moderate accuracy; linear to the global optimum.
ADMM (approximate)	$\mathcal{O}(n^2d^2)^{\S}$	Rapid to a moderate accuracy; linear to an approximate global optimum.
SCP	$\mathcal{O}(n^2)^{\S}$	Toward an approximate global optimum; $\mathcal{O}(1/T)$ rate for weakly convex loss; linear for strongly convex loss.
SGD back-propagation	$\mathcal{O}(mnd)^{\ddagger}/\mathcal{O}(n^2d)^{\dagger}$	No spurious valleys if $m \geq 2n+2$; no general results.

computational cost. Specifically, an algorithm based on the alternating direction method of multipliers (ADMM) with a quadratic per-iteration complexity as well as an algorithm based on a sampled convex program (SCP) 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 [42], and back-propagation SGD are presented in Table 1. While the IPM can converge to a highly accurate solution with fewer iterations, ADMM can rapidly reach a medium-precision solution, which is often sufficient for machine learning tasks. Compared with SGD back-propagation, ADMM has a higher theoretical complexity but is guaranteed to converge linearly to a global optimum, enabling efficient global optimization.

Prior literature has considered applying the ADMM method to ANN training [48, 49]. These works used ADMM to separate the activations and the weights of each layer, enabling parallel computing. While the ADMM algorithm in [49] converges at an $\mathcal{O}(1/t)$ rate (t is the number of iterations) to a critical point of the augmented Lagrangian of the training formulation, there is no guarantee that this critical point is a global optimizer of the ANN training loss. In contrast, this paper uses ADMM as an efficient convex optimization algorithm and introduces an entirely different splitting scheme based on the convex formulations conceived in [42]. More importantly, our ADMM algorithm provably converges to a globally optimal classifier.

A parallel line of work has focused on making convex training more efficient. Specifically, [20, 22] use linear penalty functions to derive unconstrained formulations for convex training. When the strengths of the penalizations are chosen appropriately, the formulations are exact. However, the penalization strengths can be difficult to select, since a good choice depends on the optimization landscape of the problem. Note that the solutions found via this penalty method can be used to initialize our ADMM algorithm. After our work was submitted for

review, [39] proposed a parallel method aiming to accelerate convex training. [39] considers two approaches, one using an unconstrained relaxation to the constrained convex training formulation and the other directly tackling the constrained formulation. Similar to this work, [39] also proposes to reformulate the constraints into an augmented Lagrangian. However, the separation scheme in [39] is different. Specifically, we separate the group-sparse regularization in addition to the constraints, whereas [39] only separates the constraints and thus needs to use the FISTA algorithm for the primal update step. In comparison, our ADMM separation allows the primal update subproblem (3.3a) to be solved in closed form for the case of squared loss. For general losses, our separation embeds strong convexity into the subproblem (3.3a), allowing the randomized block coordinate descent (RBCD) subroutine to converge linearly. Furthermore, our ADMM algorithm also achieves linear convergence, whereas [39] claims a slower $\mathcal{O}(\frac{1}{\epsilon\delta})$ dual convergence rate.

Combining the SCP analysis and the convex training framework leads to a further simplified convex training program that solves unconstrained convex optimization problems. This SCP-based method converges to an approximate global optimum. The scale of the SCP convex training formulation can be larger than the convex problem solved in the ADMM algorithm. However, the unconstrained nature enables the use of gradient methods, whose per-iteration complexities are lower than ADMM. The similarities between the SCP-based algorithm and extreme learning machines (ELMs) [29, 24] show that the training of a sparse ELM can be regarded as a convex relaxation of the training of an ANN, providing insights into the hidden sparsity of neural networks. Due to space restrictions, this result is presented in section SM2.

Neural networks can be vulnerable to adversarial attacks. Such a vulnerability has been observed in the field of computer vision [47, 40, 25] and controls [31]. While there have been studies on robustness certification [2, 37, 4, 9] and achieving certified robustness at test time via "randomized smoothing" [15, 3], efficiently achieving robustness via training remains an important topic. To this end, "adversarial training" [34, 25, 30] is one of the most effective methods to train robust classifiers, compared with other methods such as obfuscated gradients [7]. Specifically, adversarial training replaces the standard loss function with an "adversarial loss" and solves a bilevel min-max optimization to train robust ANNs.

When adversarial training is considered, the aforementioned issues of SGD backpropagation become worse: adversarial training can be highly unstable in practice, and convergence properties are pessimistic. Therefore, extending convex training to adversarial training is crucial. In our conference version [10], 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 (subsections 4.5 and SM5.2).

Previously, researchers have applied convex relaxation techniques to adversarial training. These works obtain convex certifications [44, 50] that upper-bound the inner maximization of the adversarial training formulation and use weak duality to develop robust loss functions. Note that while these works use convex relaxation, the resulting training formulations are still nonconvex. In contrast, we apply robust optimization techniques to the entire min-max adversarial training formulation and obtain convex training problems.

The main contributions of this work are summarized below:

- a theoretical evaluation of a relaxation that enables tractable convex training (section 2);
- efficient algorithms that accelerate convex (standard) training (section 3 and SM1);
- an extension of the convex adversarial training formulation for one-hidden-layer scalar-output ReLU networks (section 4).

1.1. Notation. Throughout this work, we focus on fully connected ANNs with one ReLU-activated hidden layer and a scalar output, defined as

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

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 ANN. We denote the target output used for training as $y \in \mathbb{R}^n$. The vectors $u_1, \dots, u_m \in \mathbb{R}^d$ are the weights of the m neurons in the hidden layer while the scalars $\alpha_1, \dots, \alpha_m \in \mathbb{R}$ are the weights of the output layer. $b_1 \dots, b_m \in \mathbb{R}$ 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. The n-dimensional identity matrix is denoted by I_n.

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 and the notation $\|\cdot\|_p$ denotes the ℓ_p norm. For a matrix A, the max norm $\|A\|_{\max}$ is defined as $\max_{ij} |a_{ij}|$, where a_{ij} is the (i, j)th entry. For a set $\mathcal{A}$, the notation $|\mathcal{A}|$ denotes its cardinality, and $\Pi_{\mathcal{A}}(\cdot)$ denotes the projection onto $\mathcal{A}$. 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, and $\operatorname{Unif}(\mathcal{S}^{n-1})$ denotes the uniform distribution on a $(n-1)$-sphere. 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 integers.

2. Practical convex ANN 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

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

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. The simplified training problem is then

$$\min_{(u_j, \alpha_j)_{j=1}^m} \ell\left(\sum_{j=1}^{m} (Xu_j)_+ \alpha_j, y \right) + \frac{\beta}{2} \sum_{j=1}^{m} (\|u_j\|_2^2 + \alpha_j^2). \tag{2.1}$$

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 [42]. Intuitively, P can be regarded as the number of possible ReLU activation patterns associated with X.

Consider the convex optimization problem

$$\begin{aligned} \min_{(v_i, w_i)_{i=1}^P} \ & \ell\Big(\sum_{i=1}^P D_i X(v_i - w_i), y\Big) + \beta \sum_{i=1}^P \Big(\|v_i\|_2 + \|w_i\|_2\Big) \\ \text{s.t.} \quad & (2D_i - I_n) X v_i \geq 0,\ (2D_i - I_n) X w_i \geq 0 \quad \forall i \in [P] \end{aligned} \tag{2.2}$$

and its dual formulation

$$\max_v -\ell^*(v) \qquad \text{s.t.} \quad |v^\top (Xu)_+| \leq \beta \ \forall u : \|u\|_2 \leq 1, \tag{2.3}$$

where $\ell^*(v) = \max_z z^\top v - \ell(z, y)$ is the Fenchel conjugate function. Note that (2.3) is a convex semi-infinite program. The next theorem, borrowed from Pilanci and Ergen's paper [42], explains the relationship between the nonconvex training problem (2.1), the convex problem (2.2), and the dual problem (2.3) when the ANN is sufficiently wide.

Theorem 2.1 (see [42]). *Let $(v_i^\star, w_i^\star)_{i=1}^P$ denote a solution of (2.2) and define $m^\star$ as $|\{i : v_i^\star \neq 0\}| + |\{i : w_i^\star \neq 0\}|$. Suppose that the ANN width m is at least $m^\star$, where $m^\star$ is upper-bounded by $n+1$. If the loss function $\ell(\cdot, y)$ is convex, then (2.1), (2.2), and (2.3) share the same optimal objective. The optimal network weights $(u_j^\star, \alpha_j^\star)_{j=1}^m$ can be recovered using the formulas*

$$\begin{aligned} (u_{j_{1i}}^\star, \alpha_{j_{1i}}^\star) &= \Big(\frac{v_i^\star}{\sqrt{\|v_i^\star\|_2}}, \sqrt{\|v_i^\star\|_2}\Big) \qquad \text{if } v_i^\star \neq 0; \\ (u_{j_{2i}}^\star, \alpha_{j_{2i}}^\star) &= \Big(\frac{w_i^\star}{\sqrt{\|w_i^\star\|_2}}, -\sqrt{\|w_i^\star\|_2}\Big) \quad \text{if } w_i^\star \neq 0, \end{aligned} \tag{2.4}$$

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

The worst-case computational complexity of solving (2.2) for the case of squared loss is $\mathcal{O}(d^3 r^3 (\frac{n}{r})^{3r})$ using standard interior-point solvers [42]. 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 $\min\{2^n, 2r(\frac{e(n-1)}{r})^r\}$ [42].

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 2.1, which approximately solves the training problem and can train ANNs with widths much less than $m^\star$. Algorithm 2.1 is an instance of the approximation described in [42, Remark 3.3], but [42] did not provide theoretical insights regarding its level

Algorithm 2.1 Practical convex training.

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

2: Solve

$$p_{s1}^\star = \min_{(v_h, w_h)_{h=1}^{P_s}} \ell\Big(\sum_{h=1}^{P_s} D_h X(v_h - w_h), y\Big) + \beta \sum_{h=1}^{P_s} (\|v_h\|_2 + \|w_h\|_2) \quad \text{s.t.} \quad (2D_h - I_n)Xv_h \geq 0,\ (2D_h - I_n)Xw_h \geq 0 \quad \forall h \in [P_s]; \tag{2.5}$$

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 (2.5) using (2.4).

of suboptimality. The following theorem bridges the gap by providing a probabilistic bound on the suboptimality of the ANN trained with Algorithm 2.1.

Theorem 2.2. *Consider an additional diagonal matrix D_{P_s+1} sampled uniformly, and construct*

$$p_{s2}^\star = \min_{(v_h, w_h)_{h=1}^{P_s+1}} \ell\Big(\sum_{h=1}^{P_s+1} D_h X(v_h - w_h), y\Big) + \beta \sum_{h=1}^{P_s+1} (\|v_h\|_2 + \|w_h\|_2) \quad \text{s.t.} \quad (2D_h - I_n)Xv_h \geq 0,\ (2D_h - I_n)Xw_h \geq 0\ \forall h \in [P_s + 1]. \tag{2.6}$$

It holds that $p_{s2}^\star \leq p_{s1}^\star$. Furthermore, if $P_s \geq \min\{\frac{n+1}{\psi\xi} - 1, \frac{2}{\xi}(n + 1 - \log\psi)\}$, 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 2.2 is presented in subsection SM6.1. Intuitively, Theorem 2.2 shows that sampling an additional D_{P_s+1} matrix will not reduce the training loss with high probability when P_s is large. One can recursively apply this bound T times to show that the solution with P_s matrices is close to the solution with $P_s + T$ matrices for an arbitrary number T. Thus, while 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 ANN is nearly optimal.

Compared with the exponential relationship between P and r, a satisfactory value of P_s is linear in n and is independent from r. Thus, when r is large, solving the approximated formulation (2.5) is significantly (exponentially) more efficient than solving the exact formulation (2.2). On the other hand, Algorithm 2.1 is no longer deterministic due to the stochastic sampling of the D_h matrices and yields solutions that upper-bound those of (2.2).

Since the confidence constants ψ and ξ are no greater than one, Theorem 2.2 only applies to overparameterized ANNs, where $P_s \geq n$. Although [42] has shown that there exists a globally optimal neural network whose width is at most $n + 1$ and Theorem 2.2 seems loose by this comparison, our theorem bounds a different quantity and is meaningful. Specifically, the bound in [42] does not provide a method that scales linearly: while a globally optimal neural network narrower than $n + 1$ exists, finding such an ANN requires solving a convex program with an exponential number of constraints. In contrast, Theorem 2.2 characterizes

the optimality of a convex optimization with a manageable number of constraints. In practice, selecting P_s is equivalent to choosing the ANN width. While Theorem 2.2 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 in subsection 5.1 show that even when P_s is much less than n (which is much less than P), Algorithm 2.1 still reliably trains high-performance classifiers.

3. An ADMM algorithm for global ANN training. The convex ReLU ANN training program (2.2) may be solved with the 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 ADMM, breaking down the optimization problem (2.2) into smaller subproblems that are easier to solve. Moreover, 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. In section SM1, we show that the ADMM algorithm extends to a family of convex training formulations.

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, and 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 $q = (q_1, \ldots, q_n) \in \mathbb{R}^n$, 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 (2.2) can be reformulated as a convex optimization problem with positive quadrant indicator functions and linear equality constraints:

$$\begin{aligned} \min_{(v_i,w_i,s_i,t_i,u_i,z_i)_{i=1}^P} \ & \ell\left(\sum_{i=1}^P F_i(u_i - z_i), y\right) + \beta\sum_{i=1}^P \|v_i\|_2 + \beta\sum_{i=1}^P \|w_i\|_2 + \sum_{i=1}^P \mathbb{I}_{\geq 0}(s_i) + \sum_{i=1}^P \mathbb{I}_{\geq 0}(t_i) \\ \text{(3.1)} \quad \text{s.t.} \quad & G_i u_i - s_i = 0, \quad G_i z_i - t_i = 0, \quad v_i - u_i = 0, \quad w_i - z_i = 0 \quad \forall i \in [P]. \end{aligned}$$

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

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

where $\operatorname{blkdiag}(\cdot, \ldots, \cdot)$ denotes the block diagonal matrix formed by the submatrices in parentheses. The formulation (3.1) is then equivalent to the compact notation

$$\text{(3.2)} \qquad \min_{v,s,u} \ell(Fu, y) + \beta\|v\|_{2,1} + \mathbb{I}_{\geq 0}(s) \quad \text{s.t.} \quad \begin{bmatrix} I_{2dP} \\ G \end{bmatrix} u - \begin{bmatrix} v \\ s \end{bmatrix} = 0,$$

Algorithm 3.1 An ADMM algorithm for the convex ANN training problem.

1: **repeat**

2: Primal update

$$u^{k+1} = \arg\min_{u} \ell(Fu, y) + \frac{\rho}{2}\|u - v^k + \lambda^k\|_2^2 + \frac{\rho}{2}\|Gu - s^k + \nu^k\|_2^2 \tag{3.3a}$$

3: Primal update

$$\begin{bmatrix} v^{k+1} \\ s^{k+1} \end{bmatrix} = \arg\min_{v,s} \beta\|v\|_{2,1} + \mathbb{I}_{\geq 0}(s) + \frac{\rho}{2}\|u^{k+1-v+\lambda^k}\|_2^2 + \frac{\rho}{2}\|Gu^{k+1} - s + \nu^k\|_2^2 \tag{3.3b}$$

4: Dual update:

$$\begin{bmatrix} \lambda^{k+1} \\ \nu^{k+1} \end{bmatrix} = \begin{bmatrix} \lambda^k + \frac{\gamma_a}{\rho}(u^{k+1} - v^{k+1}) \\ \nu^k + \frac{\gamma_a}{\rho}(Gu^{k+1} - s^{k+1}) \end{bmatrix} \tag{3.3c}$$

5: **end repeat**

where $\|\cdot\|_{2,1}$ denotes the ℓ_1-ℓ_2 mixed norm group sparse regularization and I_{2dP} is the idendity matrix in $\mathbb{R}^{2dP\times 2dP}$. The corresponding augmented Lagrangian of (3.2) is

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

where $\lambda := [\lambda_{11} \ \cdots \ \lambda_{1P} \ \ \lambda_{21} \ \cdots \ \lambda_{2P}]^\top \in \mathbb{R}^{2dP}$ and $\nu := [\nu_{11} \ \cdots \ \nu_{1P} \ \ \nu_{21} \ \cdots \ \nu_{2P}]^\top \in \mathbb{R}^{2nP}$ are dual variables, and $\rho > 0$ is a fixed penalty parameter [28].

We can apply the ADMM iterations described in Algorithm 3.1 to globally optimize (3.2).[1] Here, $\gamma_a > 0$ is a step-size constant. As will be shown next, (3.3b) and (3.3c) have simple closed-form solutions. The update (3.3a) has a closed-form solution when $\ell(\cdot)$ is the squared loss, and it can be efficiently solved numerically for general convex loss functions. When we apply ADMM to solve the approximated convex training formulation (2.5), Algorithm 3.1 becomes a subalgorithm of Algorithm 2.1. The following theorem certifies the linear convergence of the ADMM algorithm, with the proof provided in subsection SM6.2.

Theorem 3.1. *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 3.1 converges linearly to an optimal primal-dual solution for (3.2), provided that the step size γ_a is sufficiently small.*

Many popular loss functions satisfy the conditions of Theorem 3.1. Examples include the squared loss (for regression) and the binary cross-entropy loss coupled with the tanh or the sigmoid output activation (for binary classification).

[1] The ADMM algorithm is presented in the scaled dual form [13].

3.1. s and v updates. The update step (3.3b) can be separated for v^{k+1} and s^{k+1} as

$$v^{k+1} = \arg\min_v \beta\|v\|_{2,1} + \frac{\rho}{2}\|u^{k+1} - v + \lambda^k\|_2^2; \tag{3.4a}$$

$$s^{k+1} = \arg\min_s \mathbb{I}_{\geq 0}(s) + \|Gu^{k+1} - s + \nu^k\|_2^2 = \arg\min_{s\geq 0}\|Gu^{k+1} - s + \nu^k\|_2^2. \tag{3.4b}$$

Note that (3.4a) can be separated for each v_i and w_i (allowing parallelization) and solved analytically using the formulas

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

where $\mathtt{prox}_{\frac{\beta}{\rho}\|\cdot\|_2}$ denotes the proximal operation on the function $f(\cdot) = \frac{\beta}{\rho}\|\cdot\|_2$. The computational complexity of finding v_i and w_i is $\mathcal{O}(d)$. Similarly, (3.4b) can also be separated for each s_i and t_i and solved analytically using the formulas

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

where $\Pi_{\geq 0}$ denotes the projection onto the nonnegative quadrant. The computational complexity of finding s_i and t_i is $\mathcal{O}(n)$. The updates (3.4a) and (3.4b) 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 $\ell(\widehat{y}, y) = \frac{1}{2}\|\widehat{y} - y\|^2$ 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, (3.3a) amounts to

$$u^{k+1} = \arg\min_u \left\{\|Fu - y\|^2 + \frac{\rho}{2}\|u - v^k + \lambda^k\|_2^2 + \frac{\rho}{2}\|Gu - s^k + \nu^k\|_2^2\right\}. \tag{3.5}$$

Setting the gradient with respect to u to zero yields that

$$(I + \tfrac{1}{\rho}F^\top F + G^\top G)u^{k+1} = \tfrac{1}{\rho}F^\top y + v^k - \lambda^k + G^\top s^k - G^\top \nu^k. \tag{3.6}$$

Therefore, the u update can be performed by solving the linear system (3.6) 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 (3.6), a quadratic per-iteration complexity can be achieved. Specifically, the matrix $I + \frac{1}{\rho}F^\top F + G^\top G$ 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 following procedure:

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

Throughout the ADMM iterations, the first step only needs to be performed once, while the second and third steps are required for every iteration. Since the dimension of the matrix $(I+\frac{1}{\rho}F^\top F+G^\top G)$ 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 different for each iteration, and thus each iteration has a cubic complexity. Therefore, the proposed ADMM method achieves a notable speed improvement over IPM.

In the case when the approximated formulation (2.5) 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 $\mathcal{O}(d^3r^3(\frac{n}{r})^{3r})$ per-iteration complexity of [42]. The total computational complexity for reaching a point u^k satisfying $\|u^k - u^\star\| \le \epsilon_a$ is $\mathcal{O}(d^2n^2\log(1/\epsilon_a))$, where $u^\star$ is an optimal value of u and $\epsilon_a > 0$ is a predefined precision threshold. In subsection 5.2, we use numerical experiments to demonstrate that the improved efficiency of the ADMM algorithm enables the application of convex ANN training to image classification tasks, which was not possible before. Moreover, our experiments show that a favorable prediction accuracy may only require a moderate optimization precision, which can be reached with a few ADMM iterations.

3.2.2. General convex loss functions. When a general convex loss function $\ell(\widehat{y}, y)$ is considered, a closed-form solution to (3.3a) does not always exist and one may need to use iterative methods to solve (3.3a). 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 (3.3a) and propose an application of the RBCD method. The details of RBCD are presented in Algorithm 3.2. The superscript $^+$ denotes the updated quantities for each iteration, and the notation γ_r is the step size. Steps 5 and 6 of Algorithm 3.2 are derived via the chain rule of differentiation. It can be verified that (3.3a) is always strongly convex because its second term is strongly convex while the first and third terms are convex. [36, Theorem 1] has shown that when minimizing strongly convex functions, RBCD converges linearly. The theoretical convergence rate is higher when the convexity of (3.3a) is stronger and P is smaller.

In practice, the RBCD step size γ_r can be adaptively chosen via the backtracking line search. While Algorithm 3.2 updates one block in each iteration, it is possible to update

Algorithm 3.2 Randomized block coordinate descent.

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;
  $u_i^+ \leftarrow u_i - \gamma_r F_i^\top \tilde{y} - \gamma_r \rho(u_i - v_i + \lambda_{1i} + G_i^\top G_i u_i - \tilde{s}_i)$;
  $z_i^+ \leftarrow z_i + \gamma_r F_i^\top \tilde{y} - \gamma_r \rho(z_i - w_i + \lambda_{2i} + G_i^\top G_i z_i - \tilde{t}_i)$;
  $\widehat{y}^+ \leftarrow \widehat{y} + F_i((u_i^+ - z_i^+) - (u_i + z_i))$;
until $\|\nabla_u L(u, v, s, \nu, \lambda)\|_2 \le \dfrac{\varphi}{\max\{\tau, \|u\|\}}$.

multiple blocks at once by sampling multiple indices if desired. Moreover, while each iteration utilizes the gradient associated with the entire dataset in Algorithm 3.2, when the dataset consists of a large number of examples, a random portion of the dataset can be used as a surrogate.

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 assembled in advance, and there is no need to compute $G_i^\top G_i$ in each iteration. The most expensive calculations per RBCD update thus have the following complexities:

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

While it can be costly to solve (3.3a) to a high accuracy using iterative methods, especially during the early iterations of ADMM, [19, Proposition 6] has shown that even when (3.3a) 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 (3.2). Each iterative solution of the u-update subproblem can also take advantage of warm-starting by initializing at the result of the previous ADMM iteration. As a result, we alternate between an ADMM update and several RBCD updates in a delicate manner.

4. Convex adversarial training.

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

4.1. 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 [25]. The perturbation set can then be defined as

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

In this work, we consider the "white box" setting, where the adversary has complete knowledge about the ANN. One common method for training robust classifiers is to minimize the maximum loss within the perturbation set by solving the following min-max problem [38]:

$$\min_{(u_j,\alpha_j)_{j=1}^m} \left(\max_{\Delta: X+\Delta \in \mathcal{X}} \ell\Big(\sum_{j=1}^m ((X+\Delta)u_j)_+ \alpha_j, y \Big) + \frac{\beta}{2} \sum_{j=1}^m (\|u_j\|_2^2 + \alpha_j^2) \right). \tag{4.1}$$

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, the fast gradient sign method (FGSM) and projected gradient descent (PGD) are commonly used to numerically solve the inner maximization of (4.1) and generate adversarial examples [38]. Specifically, PGD generates adversarial examples $\tilde{x}$ by running the iterations

$$\tilde{x}^{t+1} = \Pi_{\mathcal{X}} \Big(\tilde{x}^t + \gamma_p \cdot \operatorname{sgn}\Big(\nabla_x \ell\Big(\sum_{j=1}^m (x^\top u_j)_+ \alpha_j, y \Big) \Big) \Big) \tag{4.2}$$

for $t = 0, 1, \ldots, T$, where $\tilde{x}^t$ is the perturbed data vector at the tth iteration, $\gamma_p > 0$ is the step size, and $T \geq 1$ is the number of iterations. The initial vector $\tilde{x}^0$ is the unperturbed data x. FGSM can be regarded as a special case of PGD where $T = 1$.

4.2. 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 are generally nonconcave over the perturbation Δ, there is no guarantee that PGD will find the true worst-case adversary. Furthermore, traditional adversarial training solves complicated bilevel min-max optimization problems, exacerbating the instability of nonconvex ANN training. Our experiments show that back-propagation gradient methods can struggle to converge when solving (4.1). Moreover, solving the bilevel optimization (4.1) requires an algorithm with a computationally cumbersome nested loop structure. To conquer such difficulties, we leverage Theorem 2.1 to recharacterize (4.1) 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 nonconvex ANN loss function holds only when the linear constraints in (2.2) 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 (2.2) reflect the ReLU patterns of X, these matrices can change when X is perturbed. Therefore, we include all distinct diagonal matrices $\operatorname{diag}([(X+\Delta)u \geq 0])$ 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 (2.2), 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 $P \leq 2r(\frac{e(n-1)}{r})^r$ as discussed above.

Finally, we replace the objective of the convex standard training formulation (2.2) with its robust counterpart, giving rise to the optimization problem

$$\min_{(v_i,w_i)_{i=1}^{\widehat{P}}} \left(\max_{\Delta: X+\Delta\in\mathcal{U}} \ell\left(\sum_{i=1}^{\widehat{P}} D_i(X+\Delta)(v_i-w_i), y \right) + \beta \sum_{i=1}^{\widehat{P}} (\|v_i\|_2 + \|w_i\|_2) \right) \tag{4.3a}$$

$$\text{s.t.} \quad \min_{\Delta: X+\Delta\in\mathcal{U}} (2D_i - I_n)(X+\Delta)v_i \geq 0, \quad \min_{\Delta: X+\Delta\in\mathcal{U}} (2D_i - I_n)(X+\Delta)w_i \geq 0 \ \ \forall i \in [\widehat{P}], \tag{4.3b}$$

where $\mathcal{U}$ is any convex additive perturbation set. The next theorem shows that (4.3) is an upper bound to the robust loss function (4.1), with the proof provided in subsection SM6.5.

Theorem 4.1. *Let* $(v^\star_{\text{rob}_i}, w^\star_{\text{rob}_i})_{i=1}^{\widehat{P}}$ *denote a solution of* (4.3) *and define* $\widehat{m}^\star$ *as* $|\{i : v^\star_{\text{rob}_i} \neq 0\}| + |\{i : w^\star_{\text{rob}_i} \neq 0\}|$. *When the ANN width* m *satisfies* $m \geq \widehat{m}^\star$, *the optimization problem* (4.3) *provides an upper bound on the nonconvex adversarial training problem* (4.1). *The robust ANN weights* $(u^\star_{\text{rob}_j}, \alpha^\star_{\text{rob}_j})_{j=1}^{\widehat{m}}$ *can be recovered using* (2.4).

When the perturbation set is zero, Theorem 4.1 reduces to Theorem 2.1. In light of Theorem 4.1, we use optimization (4.3) as a surrogate for the optimization (4.1) to train the ANN. Since (4.3) includes all D_i matrices in (2.2), 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 $P \leq 2r(\frac{e(n-1)}{r})^r$ as discussed above. As will be shown in subsection 4.3, an approximation to (4.3) can be applied to train ANNs with width much less than $\widehat{m}^\star$.

The robust constraints in (4.3b) force all points within the perturbation set to be feasible. Intuitively, for every $j \in [\widehat{m}^\star]$, (4.3b) forces the ReLU activation pattern $\text{sgn}((X+\Delta)u^\star_{\text{rob}_j})$ to stay the same for all Δ such that $X+\Delta \in \mathcal{U}$. Moreover, if $\Delta^\star_{\text{rob}}$ denotes a solution to the inner maximization in (4.3a), then $X + \Delta^\star_{\text{rob}}$ corresponds to the worst-case adversarial inputs for the recovered ANN.

Corollary 4.2. *For the perturbation set* $\mathcal{X}$, *the constraints in* (4.3b) *are equivalent to*

$$(2D_i - I_n)Xv_i \geq \epsilon\|v_i\|_1, \quad (2D_i - I_n)Xw_i \geq \epsilon\|w_i\|_1 \quad \forall i \in [\widehat{P}]. \tag{4.4}$$

The proof of Corollary 4.2 is provided in subsection SM6.6. Note that the left side of each inequality in (4.4) 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.

We will show that the new problem can be efficiently solved in important cases. Specifically, (4.3) reduces to a classic convex optimization problem when $\ell(\widehat{y}, y)$ is the hinge loss, the squared loss, or the binary cross-entropy loss. Due to space restrictions, the result for the squared loss is presented in subsection SM5.1.

4.3. Practical algorithm for convex adversarial training. Since Theorem 2.2 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 (4.3). Therefore, an approximation to (4.3) can be applied to train robust ANNs with widths much less than $\widehat{m}^\star$. Similar to the strategy rendered in Algorithm 2.1, 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

Algorithm 4.1 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

$$\begin{aligned} (4.5) \qquad \min_{(v_i, w_i)_{i=1}^{\widehat{P}}} & \left(\max_{\Delta : X+\Delta \in \mathcal{U}} \ell\left(\sum_{h=1}^{P_s} D_h(X+\Delta)(v_h - w_h), y \right) + \beta \sum_{h=1}^{P_s} (\|v_h\|_2 + \|w_h\|_2) \right) \\ \text{s.t.} \quad & \min_{\Delta : X+\Delta \in \mathcal{U}} (2D_h - I_n)(X+\Delta) v_h \geq 0 \quad \forall h \in [P_s], \\ & \min_{\Delta : X+\Delta \in \mathcal{U}} (2D_h - I_n)(X+\Delta) w_h \geq 0 \quad \forall h \in [P_s]. \end{aligned}$$

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 (4.5) using (2.4).

perturbations, leading to Algorithm 4.1. P_a and S are preset parameters that determine the number of random weight samples with $P_a \times S \geq P_s$.

4.4. Convex hinge loss adversarial training. While the inner maximization of the robust problem (4.3) 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 $y \in \{-1, 1\}^n$.[2]

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

$$(4.6) \qquad \min_{(u_j, \alpha_j)_{j=1}^m} \left(\frac{1}{n} \cdot \mathbf{1}^\top \left(\mathbf{1} - y \odot \sum_{j=1}^m (Xu_j)_+ \alpha_j \right)_+ + \frac{\beta}{2} \sum_{j=1}^m (\|u_j\|_2^2 + \alpha_j^2) \right).$$

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

$$(4.7) \qquad \min_{(u_j, \alpha_j)_{j=1}^m} \left(\max_{\Delta : X+\Delta \in \mathcal{X}} \frac{1}{n} \cdot \mathbf{1}^\top \left(\mathbf{1} - y \odot \sum_{j=1}^m ((X+\Delta)u_j)_+ \alpha_j \right)_+ + \frac{\beta}{2} \sum_{j=1}^m (\|u_j\|_2^2 + \alpha_j^2) \right).$$

[2] Other ℓ_p norm-bounded additive perturbation sets can be similarly analyzed, as shown in subsection SM5.3. It is also straightforward to extend the analysis in this section to any convex piecewise-affine loss functions.

Applying Theorem 4.1 and Corollary 4.2 leads to the following formulation as an upper bound on (4.7):

$$\min_{(v_i,w_i)_{i=1}^{\widehat{P}}} \left(\max_{\Delta: X+\Delta\in\mathcal{X}} \frac{1}{n}\cdot \mathbf{1}^\top \left(\mathbf{1} - y \odot \sum_{i=1}^{\widehat{P}} D_i(X+\Delta)(v_i - w_i) \right)_+ + \beta \sum_{i=1}^{\widehat{P}} (\|v_i\|_2 + \|w_i\|_2) \right)$$

$$\text{(4.8)} \quad \text{s.t.} \quad (2D_i - I_n)Xv_i \geq \epsilon \|v_i\|_1, \;\; (2D_i - I_n)Xw_i \geq \epsilon \|w_i\|_1 \;\; \forall i \in [\widehat{P}].$$

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 4.3. Solving the inner maximization of (4.8) in closed form leads to the next theorem, whose proof is provided in subsection SM6.7.

Theorem 4.3. *For the binary classification problem, the inner maximum of* (4.8) *is attained at* $\Delta^\star_{\text{hinge}} = -\epsilon \cdot \operatorname{sgn}(\sum_{i=1}^{\widehat{P}} D_i y (v_i - w_i)^\top)$, *and the bilevel optimization problem* (4.8) *is equivalent to the classic optimization problem*

$$\min_{(v_i,w_i)_{i=1}^{\widehat{P}}} \frac{1}{n} \sum_{k=1}^{n} \left(1 - y_k \sum_{i=1}^{\widehat{P}} d_{ik} x_k^\top (v_i - w_i) + \epsilon \left\| \sum_{i=1}^{\widehat{P}} d_{ik}(v_i - w_i) \right\|_1 \right)_+ + \beta \sum_{i=1}^{\widehat{P}} (\|v_i\|_2 + \|w_i\|_2)$$

$$\text{(4.9)} \quad \text{s.t.} \quad (2D_i - I_n)Xv_i \geq \epsilon \|v_i\|_1, \;\; (2D_i - I_n)Xw_i \geq \epsilon \|w_i\|_1 \;\; \forall i \in [\widehat{P}],$$

where d_{ik} *denotes the* kth *diagonal element of* D_i.

The problem (4.9) is a finite-dimensional convex program that upper-bounds (4.7), the robust counterpart of (4.6). We can thus solve (4.9) to robustly train the ANN.

4.5. 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 $\ell(\widehat{y}, y) = -2\widehat{y}^\top y + \mathbf{1}^\top \log(e^{2\widehat{y}} + 1)$. The nonconvex adversarial training formulation considering the ℓ_∞-bounded data uncertainty set $\mathcal{X}$ is then

$$\text{(4.10)} \quad \min_{(u_j,\alpha_j)_{j=1}^{m}} \left(\max_{\|\Delta\|_{\max}\leq\epsilon} \frac{1}{n} \sum_{k=1}^{n} \Big(-2\widehat{y}_k y_k + \log(e^{2\widehat{y}_k} + 1) \Big) \right) + \frac{\beta}{2} \sum_{j=1}^{m} (\|u_j\|_2^2 + \alpha_j^2),$$
$$\text{where} \;\; \widehat{y} := \sum_{j=1}^{m} ((X+\Delta)u_j)_+ \alpha_j.$$

Applying Theorem 4.1 and Corollary 4.2 leads to the following optimization problem as an upper bound on (4.10):

$$\min_{(v_i,w_i)_{i=1}^{\widehat{P}}} \left(\max_{\|\Delta\|_{\max}\leq\epsilon} \frac{1}{n} \sum_{k=1}^{n} \Big(-2\widehat{y}_k y_k + \log(e^{2\widehat{y}_k} + 1) \Big) \right) + \beta \sum_{i=1}^{\widehat{P}} \Big(\|v_i\|_2 + \|w_i\|_2 \Big)$$

$$\text{(4.11)} \quad \text{s.t.} \quad (2D_i - I_n)Xv_i \geq \epsilon \|v_i\|_1, \;\; (2D_i - I_n)Xw_i \geq \epsilon \|w_i\|_1 \quad \forall i \in [\widehat{P}],$$
$$\widehat{y}_k = \sum_{i=1}^{\widehat{P}} d_{ik} x_k^\top (v_i - w_i) + \sum_{i=1}^{\widehat{P}} d_{ik} \delta_k^\top (v_i - w_i).$$

Consider the convex optimization formulation

$$\begin{aligned}
&\min_{(v_i,w_i)_{i=1}^{\widehat{P}}} \frac{1}{n}\left(\sum_{k=1}^{n} f\circ g_k(\{v_i,w_i\}_{i=1}^{\widehat{P}})\right)+\beta\sum_{i=1}^{\widehat{P}}\left(\|v_i\|_2+\|w_i\|_2\right)\\
&\text{s.t.}\quad (2D_i-I_n)Xv_i\geq\epsilon\|v_i\|_1,\ (2D_i-I_n)Xw_i\geq\epsilon\|w_i\|_1\quad \forall i\in[\widehat{P}],\\
&\qquad f(u):=\log(e^{2u}+1),\\
&\qquad g_k(\{v_i,w_i\}_{i=1}^{\widehat{P}}):=(2y_k-1)\sum_{i=1}^{\widehat{P}} d_{ik}x_k^\top(v_i-w_i)+\epsilon\cdot\left\|\sum_{i=1}^{\widehat{P}} d_{ik}(v_i-w_i)\right\|_1\quad\forall k\in[n].
\end{aligned}\tag{4.12}$$

The next theorem establishes the equivalence between (4.12) and (4.11). The proof is provided in subsection SM6.9.

Theorem 4.4. *The optimization* (4.12) *is a convex program that is equivalent to the bilevel optimization* (4.11) *and can be used as a surrogate for* (4.10) *to train robust ANNs. The worst-case perturbation is* $\Delta^\star_{\mathrm{BCE}}=-\epsilon\cdot\operatorname{sgn}((2y-1)\sum_{i=1}^{\widehat{P}} D_i(v_i-w_i)^\top)$.

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 $D_1,\dots,D_{\widehat{P}}$ matrices, we again only need to enumerate all vertices of $\mathcal{X}$ instead of all points in $\mathcal{X}$.

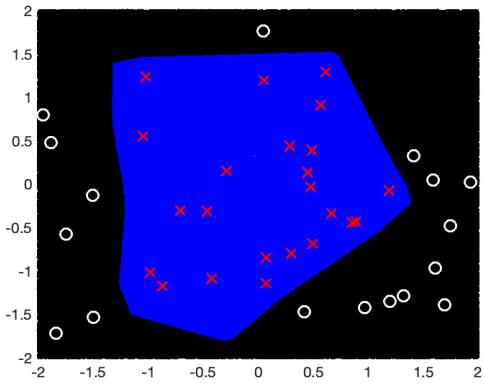
5. Numerical experiments. Due to space restrictions, we focus on binary classification with the hinge loss and defer the squared loss results to subsection SM3.5.

5.1. Approximated convex standard training. In this subsection, we use numerical experiments to demonstrate the efficacy of practical standard training (Algorithm 2.1) and to show the level of suboptimality of the ANN trained using Algorithm 2.1.[3] The experiment was performed on a randomly generated dataset with $n=40$ and $d=2$ shown in Figure 1(a). The upper bound on the number of ReLU activation patterns is $4(\frac{e(39)}{2})^2=11239$. We ran Algorithm 2.1 to train ANNs using the hinge loss with the number of D_h matrices equal to $4,8,16,\dots,2048$ and compared the optimized loss.[4] We repeated this experiment 15 times for each setting and plotted the loss in Figure 1(b). 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 2.1 yields consistent and favorable results. Further increasing the number of D matrices does not produce a significantly lower loss. By Theorem 2.2, $P_s=128$ corresponds to $\psi\xi=0.318$.

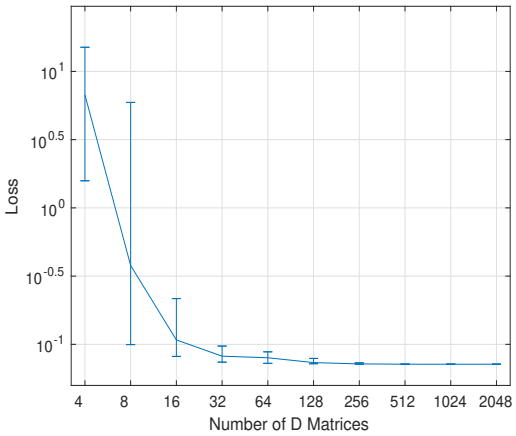
5.2. The ADMM convex training algorithm. We now present the experiment results with the ADMM training algorithm. We use Algorithm 3.1 to solve the approximate convex training formulation (2.5) with the sampled D_h matrices. The hyperparameter settings for the experiments are discussed in subsection SM4.1, where we also present guidelines on selecting the ADMM hyperparameters.

[3]For all experiments in this section, CVX [26] and CVXPY [1, 16] with the MOSEK [5] solver were used for solving optimization on a laptop computer, unless otherwise stated. Off-the-shelf solvers supported by CVX and CVXPY often treat the convex training problem as a general second order cone program. Among all solvers that we experimented on the convex training formulation, MOSEK is the most efficient.

[4]To reliably sample P_s matrices, $P_a\cdot S$ in Algorithm 4.1 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 to be 10^{-4}.



(a) A randomized 2-dimensional dataset used in this experiment. 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.



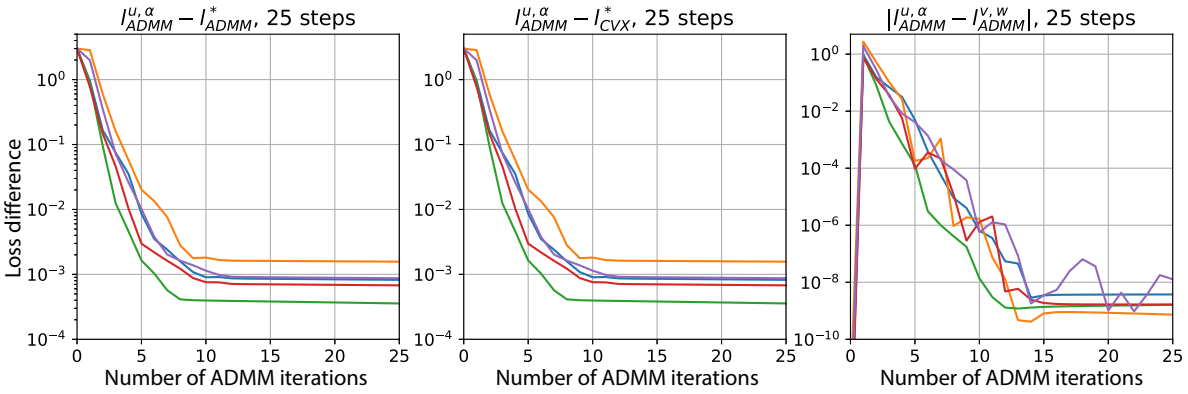
(b) The optimized training loss for each P_s. When P_s reaches 128, the mean and variance of the optimized loss become very small.

Figure 1. *Analyzing the effect of P_s on convex standard training.*

5.2.1. Squared loss (closed-form u updates)—convergence. For the case of the squared loss, the closed-form solution (3.6) is used for the u updates. We first demonstrate the convergence of the proposed ADMM algorithm using illustrative random data with dimensions $n = 6, d = 5, P_s = 8$. CVX [26] with the IPM-based MOSEK solver [5] was used to solve the optimal objective of (2.2) as the ground truth.

We first explain the notation used in the figures. We use $l^\star_{\text{CVX}}$ to denote the CVX optimal objective and use $l^\star_{\text{ADMM}}$ to denote the objective that ADMM converges to as the number of iterations k goes to infinity. There are several methods to calculate the training loss obtained by ADMM. For fair comparisons among ADMM, CVX, and SGD, we use (2.4) to recover the ANN 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 nonconvex training loss (2.1). The loss at each iteration calculated via this method is denoted as $l^{u,\alpha}_{\text{ADMM}}$, and the ADMM solution $l^\star_{\text{ADMM}}$ is also calculated via this method. At each iteration, we also compute the convex objective of (2.2) using $(v_h^k, w_h^k)_{h=1}^{P_s}$, denoted as $l^{v,w}_{\text{ADMM}}$. Since ADMM uses dual variables to enforce the constraints, while the ADMM solution is feasible as k goes to infinity, the intermediate iterations may not be feasible. When the constraints in (2.2) are satisfied, it holds that $l^{u,\alpha}_{\text{ADMM}} = l^{v,w}_{\text{ADMM}}$. Otherwise, $l^{u,\alpha}_{\text{ADMM}}$ may be different from $l^{v,w}_{\text{ADMM}}$. 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.

While it can be expensive for ADMM to converge to a high precision (note that the algorithm is guaranteed to linearly converge to a global minimum given an ample computation time according to Theorem 3.1), an approximate solution is usually sufficient for achieving a high validation accuracy since decreasing the training loss excessively could induce overfitting. Therefore, when performing the experiments, we apply early stopping [43], a common training



(a) $l^{u,\alpha}_{\text{ADMM}} - l^{\star}_{\text{ADMM}}$ (b) $l^{u,\alpha}_{\text{ADMM}} - l^{\star}_{\text{CVX}}$ (c) $\left|l^{u,\alpha}_{\text{ADMM}} - l^{v,w}_{\text{ADMM}}\right|$

Figure 2. *Gap between the cost returned by ADMM for the first* 25 *iterations and the true optimal cost for the* 5 *independent runs.*

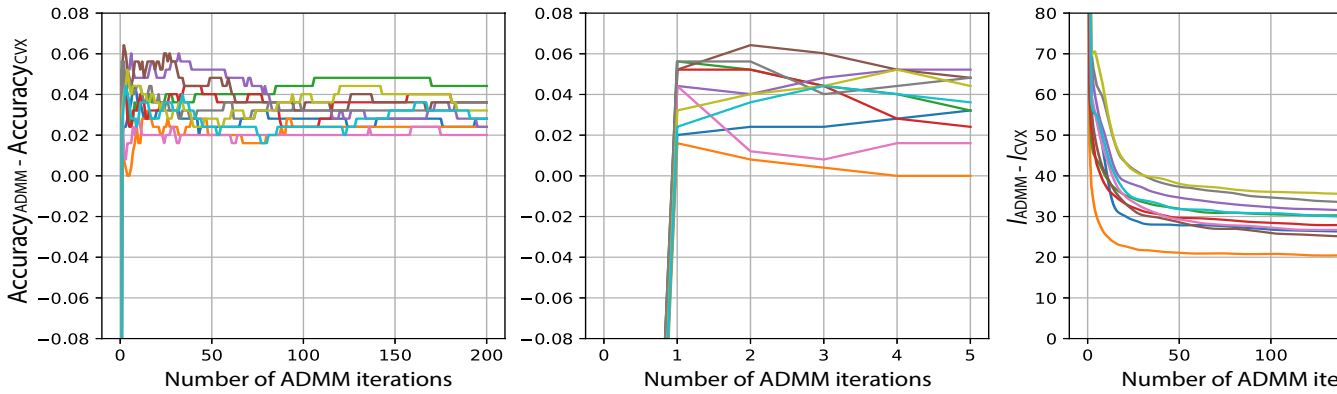
technique that improves generalization. Figures 2(a) and 2(b) show that a precision of 10^{-3} can be achieved within 25 iterations. Moreover, Figure 2(c) shows that the solution after 25 iterations violates the constraints insignificantly. This behavior of "converging rapidly in the first several steps and 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 a few ADMM iterations can achieve a better prediction performance than the CVX solution. In subsection SM3.1, we present empirical results that demonstrate the asymptotic convergence properties of ADMM.

To visualize how the prediction performance achieved by the model changes as the ADMM iteration progresses, we run the ADMM iterations on the "mammographic masses" dataset from the UCI Machine Learning Repository [18] and record the prediction accuracy on the validation set at each iteration. 70% of the dataset is randomly selected as the training set, and the other 30% is used as the validation set. Figure 3 plots the difference between the ADMM accuracy and the CVX accuracy at each iteration. In all experiments, all variables in the ADMM algorithm are initialized to be zero.

All 10 runs achieve superior validation accuracy throughout the first 200 iterations compared with the CVX baseline, and even the first 5 iterations outperform the baseline, with the best run outperforming CVX by 6%. After about 80 iterations, the accuracy stabilizes at around 2% to 4% better than the CVX baseline. In conclusion, the prediction performance of the classifiers trained by ADMM is superior even when only a few iterations are run.

5.2.2. 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 MNIST handwritten digits dataset with $d = 100$. The task was to perform binary classification between digits "2" and "8." We first fix $P_s = 8$ and vary n from 100 to 11809.[5] We independently repeat the experiment five times for each n setting and present the average results in Figures 4(a) and 4(b). In each experiment, ADMM is allowed

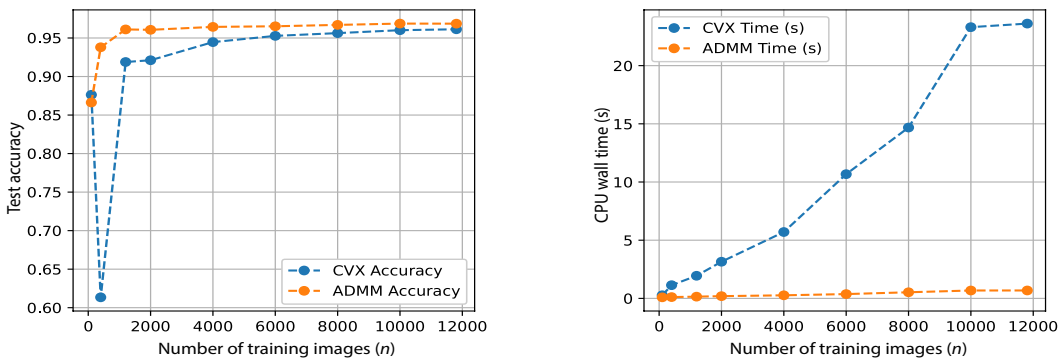
[5] 11809 is the the total number of 2's and 8's in the training set.



(a) $\text{Accuracy}_{\text{ADMM}} - \text{Accuracy}_{\text{CVX}}$ (positive means the ADMM solution outperforms CVX).

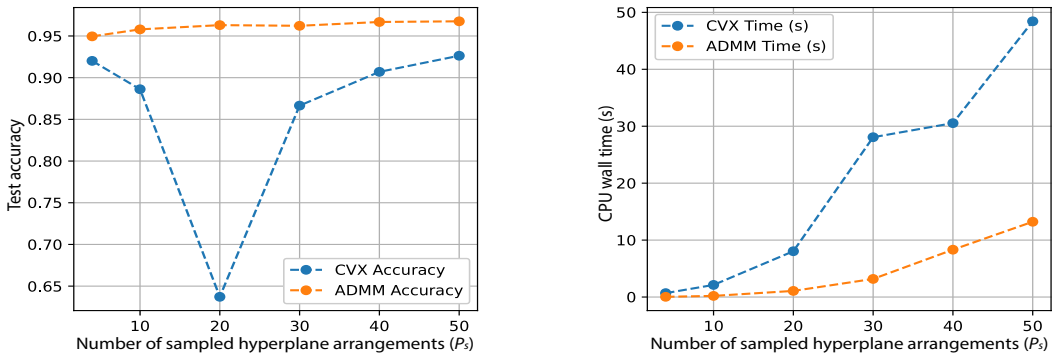
(b) 3a zoomed-in to the first five iterations.

Figure 3. *Comparing the ANNs trained with ADMM and with CVX over* 10 *independent runs on the mammographic masses dataset.*



(a) Average validation accuracy for each n.

(b) Average CPU wall time for each n.



(c) Average validation accuracy for each P_s.

(d) Average CPU wall time for each P_s.

Figure 4. *Analyzing the effect of* n *and* P_s *on ADMM convex training with the MNIST dataset.*

to run six iterations, which is sufficient to train an accurate ANN. For all choices of n except $n = 100$, the ANNs trained with ADMM attain higher accuracy than CVX networks. This is because while ADMM and CVX solve the same problem, the medium-precision solution from ADMM generalizes better than the high-precision CVX solution. More importantly, as n increases, the CPU time required for CVX grows much faster than ADMM's execution time, which increases linearly in n. While it is also theoretically possible to run the IPM to a medium precision, even a few IPM iterations become too expensive when n is large. Moreover, since the IPM uses barrier functions to approximate the constraints, a medium-precision solution produced by the IPM may have feasibility issues, while the ADMM solution sequence generally has good feasibility, as illustrated in Figure 2.

Similarly, we fix $n = 1000$ and vary P_s from 4 to 50. The average results over five runs are shown in Figures 4(c) and 4(d). Once again, the proposed ADMM algorithm achieves a higher accuracy for each P_s, and the average CPU time of ADMM grows much slower than the CVX CPU time. When P_s is 20, all five CVX runs achieve low validation accuracy, possibly because the structure of the true underlying distribution cannot be well approximated with a combination of 20 linear classifiers. Figures 4(c) and 4(d) also show that the CPU time scales quadratically with P_s, confirming our theoretical analysis of the $\mathcal{O}(nP_s + d^2P_s^2)$ per-iteration complexity.

5.2.3. Squared loss (closed-form u updates)—MNIST, Fashion MNIST, and CIFAR-10. We now demonstrate the effectiveness of the proposed ADMM algorithm on all images of "2" and "8" in the MNIST dataset without downsampling ($n = 11809$ and $d = 784$). The parameter P_s was chosen to be 24, corresponding to a network width of at most 48. The prediction accuracy on the validation set, the training loss, and the CPU time are shown in Table 2. The baseline method "CVX" corresponds to using CVX to globally optimize the ANN by solving (2.2), while "Back-prop" denotes the conventional method that performs a SGD local search on the nonconvex cost function (2.1).

Table 2 shows that the training loss returned by ADMM is higher than the true optimal cost but lower than the back-propagation solution. Note that the difference between the ADMM training loss and the CVX loss is due to the early-stopping strategy applied to ADMM. ADMM will converge to the true global optimal with a sufficient computation time, but we prematurely terminate the algorithm once the validation accuracy becomes satisfactory so that the rapid initial convergence of ADMM can be fully exploited. In contrast, back-propagation does not have this guarantee due to the nonconvexity of (2.1). Moreover, back-propagation is highly sensitive to the initialization and the hyperparameters. While ADMM also requires a prespecified step size γ_a, it is much more stable: its convergence to a primal optimum does

Table 2

Average experiment results with the squared loss on the MNIST dataset over five independent runs. We run 10 *ADMM iterations for each setting.*

Method	Validation accuracy	CPU time (s)	Training loss	Global convergence
Back-prop	98.86 %	74.09	422.4	No
CVX	70.99 %	14879	1.146	Yes
ADMM	98.90 %	802.2	223.2	Yes

Table 3

Average experiment results with the squared loss over five independent runs.

Fashion MNIST (42 ADMM iterations, P_s set to 18)			
Method	Validation accuracy	CPU time (s)	Training loss
Back-prop	99.04% (.0735%)	183.6	175.1 (4.246)
ADMM	98.73% (.0200%)	167.1	129.7 (13.24)
Back-prop (DS)	98.34% (.0917%)	18.31	433.0 (10.40)
ADMM (DS)	98.80% (.0585%)	6.840	380.1 (17.74)
Downsampled CIFAR-10 (30 ADMM iterations, P_s set to 18)			
Method	Validation accuracy	CPU time (s)	Training loss
Back-prop (DS)	90.90% (.305%)	122.7	991.5 (11.68)
ADMM (DS)	86.89% (.132%)	118.6	607.6 (10.76)

"DS" denotes downsampling with a stride of 2. The numbers in parentheses are the standard deviations over five runs. Note that the ADMM algorithm is theoretically guaranteed to converge to an approximate global minimum, whereas back-propagation does not have this property.

not depend on the step size [13, Appendix A]. An optimal step size speeds up the training, but a suboptimal step size is also acceptable.

ADMM achieves a higher validation accuracy than both CVX and back-propagation SGD. Once again, while ADMM and CVX solve the same problem, the CVX solution suffers from overfitting and thus cannot generalize well to the validation data.

The training time of ADMM is considerably shorter than CVX. Specifically, assembling the matrix $I+\frac{1}{\rho}F^\top F+G^\top G$ required 22% of the time, and the Cholesky decomposition needed 34% of the time, while each ADMM iteration took only 4.4% of the time. Thus, running more ADMM iterations will not considerably increase the training time.

We also compare ADMM with back-propagation on the harder and more challenging Fashion MNIST [51] and CIFAR-10 datasets. For Fashion MNIST, we perform binary classification between the "pullover" and the "bag" classes on both full data ($n = 12000$, $d = 784$) and downsampled data ($n = 12000$, $d = 196$). For CIFAR-10, we perform binary classification between "birds" and "ships" and downsample the images to $16 \times 16 \times 3$. The results are presented in Table 3, and we plot the training loss with respect to time in Figure 5. The results shows that ADMM converges faster and achieves a lower loss within the same amount of time, even though it requires some preprocessing before the iterations start. However, on these datasets, the classifiers learned via back-propagation generalize better to the validation set. Gradient descent is known to have favorable properties when it comes to machine learning problems, where solutions with similar losses can have vastly different properties. For applications where training data is abundant, ADMM is a well-suited method to use since the generalization gap would be small for these scenarios.

We also note that ADMM is extremely efficient on the downsampled Fashion MNIST dataset, since the faster convergence of ADMM overshadows the higher complexity associated with the decomposition when the data dimension is smaller. This result shows that ADMM is particularly suitable for data with a dimension of around 200.

5.2.4. Binary cross-entropy loss (iterative u updates)—MNIST. To verify the efficacy of using the RBCD method to numerically solve (3.3a), we similarly experiment with the

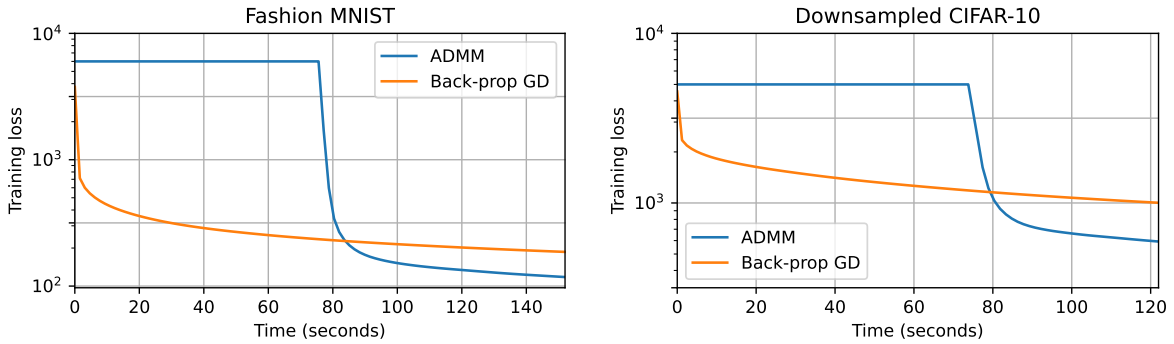


Figure 5. *The learning curves of the closed-form ADMM algorithm and back-propagation gradient descent. The flat parts of the ADMM curves represent the preprocessing time.*

Table 4

Average experiment results with the binary cross-entropy loss over five runs. The main advantage of ADMM-RBCD is its theoretically guaranteed global convergence.

MNIST (34 ADMM iterations, $P_s = 24$)			
Method	Validation accuracy	CPU time (s)	Training loss
Back-prop	98.91 %	62.06	437.6
CVX	98.21 %	14217	1.007
ADMM-RBCD	98.89 %	555.8	310.3

binary cross-entropy loss coupled with a tanh output activation. The resulting loss function is $\ell(\widehat{y}, y) = -2\widehat{y}^\top y + \mathbf{1}^\top \log(e^{2\widehat{y}} + 1)$. Since the value of the full augmented Lagrangian gradient in the stopping condition of Algorithm 3.2 is difficult to obtain, we use the amount of objective improvement as a surrogate.

The experiment results are shown in Table 4. On the MNIST dataset, the ADMM-RBCD algorithm achieves a high validation accuracy while requiring a training time 94.6% shorter than the time of globally optimizing the cost function (2.2) with CVX. ADMM-RBCD also 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 back-propagation local search, trading the training speed for the global convergence guarantee. The extremely slow pace of CVX forbids its application to even medium-scaled problems, while ADMM-RBCD makes convex training much more practical by balancing between efficiency and optimality.

5.2.5. GPU acceleration. The success of modern deep learning relies on the parallelized computing enabled by GPUs. Using GPUs to accelerate the proposed ADMM algorithm is straightforward. All operations required in the ADMM algorithm are implemented in existing deep learning libraries that support GPUs, such as PyTorch [41]. Consider each step of Algorithm 3.1. Note that (3.3c) consists of parallelizable algebraic operations, and we have shown that (3.3b) reduces to parallelizable elementwise operations. Now, consider (3.3a). If the RBCD algorithm is used to solve (3.3a), then all operations are again parallelizable (as is the case for traditional back-propagation gradient descent), and auto-differentiation can be used to obtain the closed-form gradients.

To verify the effectiveness of GPU acceleration and show that ADMM-RBCD scales to wider neural networks and higher dimensions with the help of GPUs, we use the method to train binary classifiers with P_s set to 120 on the CIFAR-10 dataset. The average validation accuracy over five runs is 91.23%. On a MacBook Pro laptop computer, this task takes 474.5 seconds on average. Repeating the experiment on an Nvidia V100 GPU only requires 24.64 seconds, which is a 19.25x speed-up.

5.2.6. Summary of ADMM experiment results. Based on the above experiment results, we summarize some advantages of our ADMM methods below:

1. While the closed-form ADMM algorithm has a higher theoretical complexity compared with back-propagation, it is guaranteed to linearly converge to a global optimum if allowed to run for a sufficiently long time, enabling efficient global optimization of neural networks. Back-propagation does not have this property.
2. The closed-form ADMM algorithm often converges rapidly in the first few iterations. Since a moderately accurate solution is sufficient for many machine learning tasks, this fast initial convergence is very advantageous.
3. For datasets with a relatively small number of dimensions, the closed-form ADMM algorithm is more efficient than back-propagation (as shown in Table 3), since the faster convergence outweighs the increased complexity.
4. Compared with closed-form ADMM, ADMM-RBCD applies to general convex loss functions, and scales better to wide ANNs, but is less efficient, as illustrated in Table 4. ADMM-RBCD is then a trade-off between CVX (high solution quality) and back-propagation (efficient), while maintaining the theoretically provable global convergence.

In summary, the proposed ADMM method is particularly suited for applications where

- abundant training data exists (a low empirical risk translates to a low true risk);
- accuracy is more important than computational efficiency;
- the number of dimensions is not too large.

5.3. Convex adversarial training. All experiment results in this section are obtained using CVX with the MOSEK solver based on the IPM.

5.3.1. Hinge loss convex adversarial training—two-dimensional illustration. To analyze the decision boundaries obtained from convex adversarial training, we ran Algorithms 2.1 and 4.1 on 34 random points in two-dimensional space for binary classification. The algorithms were run with the parameters $P_s = 360$ and $\epsilon = 0.08$. A bias term was included by concatenating a column of ones to the data matrix X. The decision boundaries shown in Figure 6 confirm that Algorithm 4.1 fits the perturbation boxes as designed, coinciding with the theoretical prediction [38, Figure 3]. In subsection SM3.4, we compare the decision boundaries of convex training methods and back-propagation methods and discuss how the regularization strength β affects the decision boundaries. Additionally, in subsection SM3.3, we compare the convex and the nonconvex optimization landscapes and demonstrate robust certifications around the training data points.

5.3.2. 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

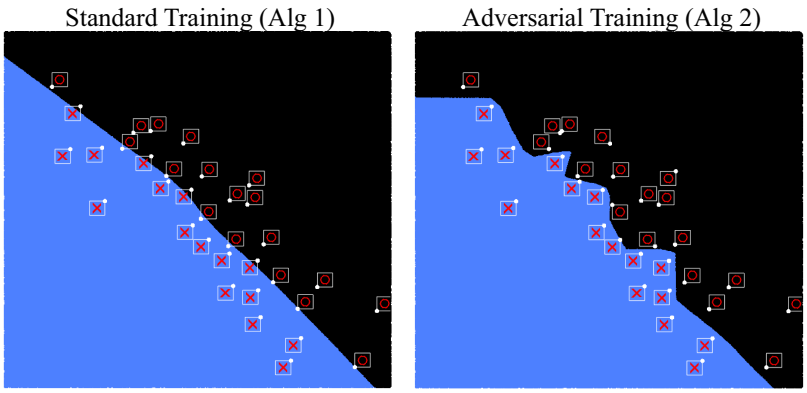


Figure 5. Visualization.

Figure 6. *Visualization of the binary decision boundaries in a two-dimensional space. Algorithm 4.1 fitted the perturbation boxes, while the standard training fitted the training points.*

image classification dataset [33] for binary classification between "birds" and "ships." The subset consists of 600 images downsampled to $d = 7 \times 7 \times 3 = 147$.[6] We use clean data and adversarial data generated with FGSM and PGD to compare the performances of Algorithm 2.1, Algorithm 4.1, traditional back-propagation standard training (abbreviated as GD-std), and the widely used adversarial training method: use FGSM or PGD to solve for the inner maximum of (4.7) and use back-propagation to solve the outer minimization (abbreviated as GD-FGSM and GD-PGD). The implementation details of FGSM and PGD are discussed in subsection SM4.2.

The results on the CIFAR-10 subset are provided in Table 5. Convex standard training (Algorithm 2.1) achieves a higher clean accuracy than GD-std and returns a much lower training loss, supporting the findings of Theorem 2.2. Note that the nonadversarially convex-trained model is highly sensitive to adversarial perturbations. Standard training has no control over the loss of the perturbed inputs, and the high optimization accuracy of convex standard training exacerbates this issue. This issue highlights the importance of the development of convex adversarial training (Algorithm 4.1), which achieves a higher accuracy on clean data and adversarial data compared with GD-FGSM and GD-PGD. While Algorithm 4.1 solves the upper-bound problem (4.9), it returns a lower training objective compared with GD-FGSM and GD-PGD, showing that the back-propagation method fails to find an optimal network. Moreover, we have observed that Algorithms 2.1 and 4.1 are much more stable. Note that our algorithms are theoretically guaranteed to converge to their global optima.

We also compare the aforementioned SDP relaxation adversarial training method [44] and the LP relaxation method [50] against our work on the CIFAR-10 subset. While an iteration of the LP or the SDP method is faster than a GD-PGD iteration, the ANNs trained with the LP or the SDP method achieve worse accuracy and robustness than those trained with

[6]The parameters are $\epsilon = 10/255$, $\beta = 10^{-4}$, and $P_s = 36$, corresponding to an ANN width of at most 72.

Table 5

Average optimal objective and accuracy on clean and adversarial data over seven runs on the CIFAR-10 dataset. The standard deviations across the runs are shown in parentheses.

Method	Clean accuracy	FGSM adv.	PGD adv.	Objective	CPU time (s)
GD-std	79.56% (.414%)	47.09% (.4290%)	45.60% (.4796%)	.3146	108.4
GD-FGSM	75.30% (3.10%)	61.03% (4.763%)	60.99% (4.769%)	.8370	154.9
GD-PGD	76.56% (.604%)	62.48% (.2215%)	62.44% (.1988%)	.8220	1764
Algorithm 2.1	81.01% (.809%)	.4857% (.1842%)	.3571% (.1239%)	6.910×10^{-3}	37.77
Algorithm 4.1	78.36% (.325%)	66.95% (.4564%)	66.81% (.4862%)	.6511	1544

Algorithm 4.1: 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.[7] These results support that Algorithm 4.1 trains more robust ANNs and that the LP and SDP relaxations can be extremely loose and unstable. While [44, 50] apply the convex relaxation method to the adversarial training problem, their training formulations are nonconvex.

The presence of an ℓ_1 norm term in the upper-bound formulations (4.9) and (4.12) indicates that adversarial training with a small ϵ has a regularizing effect, which can improve generalization, supporting the finding of [34]. In the above experiments, Algorithm 4.1 outperforms Algorithm 2.1 on adversarial data, highlighting the contribution of Algorithm 4.1: a novel convex adversarial training procedure that reliably trains robust ANNs.

6. Concluding remarks.

We use the SCP theory to characterize the quality of the solution obtained from an approximation method, providing theoretical insights into practical convex training. We then develop a separating scheme and apply the ADMM algorithm to a family of convex training formulations. When combined with the approximation method, the algorithm achieves a quadratic per-iteration computational complexity and a linear convergence toward an approximate global optimum. We also introduced a simpler unconstrained convex training formulation based on an SCP relaxation. The characterization of its solution quality shows that ELMs are convex relaxations to ANNs. Compared to the traditional back-propagation algorithms, our proposed training algorithms possess theoretical convergence rate guarantees and enjoy the absence of spurious local minima. Compared with naively solving the convex training formulation using general-purpose solvers, our algorithms have much improved complexities, making a significant step toward practical convex training.

We also use 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:** 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 4.3, the inner maximization over the robust loss function is solved exactly.

[7] For SDP, the robustness parameter is chosen as $\lambda = .04$, since a larger λ causes the algorithm to fail.

- **Hyperparameter-free:** Algorithm 4.1 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, facilitating the potential application of ANNs in safety-critical applications.

REFERENCES

[1] A. Agrawal, R. Verschueren, S. Diamond, and S. Boyd, *A rewriting system for convex optimization problems*, J. Control Decis., 5 (2018), pp. 42–60.

[2] B. Anderson, Z. Ma, J. Li, and S. Sojoudi, *Tightened convex relaxations for neural network robustness certification*, in Proceedings of the IEEE Conference on Decision and Control, 2020.

[3] B. Anderson and S. Sojoudi, *Certified robustness via locally biased randomized smoothing*, in Proceedings of the 4th Annual Learning for Dynamics and Control Conference, 2022, pp. 207–220.

[4] B. Anderson and S. Sojoudi, *Data-driven certification of neural networks with random input noise*, IEEE Transactions on Control Network Systems, 10 (2023), pp. 249–260.

[5] *The MOSEK Optimization Toolbox for MATLAB Manual. Version* 9.0, Mosek ApS, 2019.

[6] R. Arora, A. Basu, P. Mianjy, and A. Mukherjee, *Understanding deep neural networks with rectified linear units*, in Proceedings of the International Conference on Learning Representations, 2018.

[7] A. Athalye, N. Carlini, and D. Wagner, *Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples*, in Proceedings of the International Conference on Machine Learning, 2018.

[8] F. Bach, *Breaking the curse of dimensionality with convex neural networks*, J. Mach. Learn. Res., 18 (2017), pp. 1–53.

[9] Y. Bai, B. G. Anderson, and S. Sojoudi, *Improving the Accuracy-Robustness Trade-off of Classifiers via Adaptive Smoothing*, preprint, https://arxiv.org/pdf/2301.12554.pdf, 2023.

[10] Y. Bai, T. Gautam, Y. Gai, and S. Sojoudi, *Practical convex formulation of robust one-hidden-layer neural network training*, in Proceedings of the American Control Conference, 2022.

[11] E. Belilovsky, M. Eickenberg, and E. Oyallon, *Greedy layerwise learning can scale to ImageNet*, in Proceedings of the International Conference on Machine Learning, 2019.

[12] Y. Bengio, N. Roux, P. Vincent, O. Delalleau, and P. Marcotte, *Convex neural networks*, in Proceedings of the Annual Conference on Neural Information Processing Systems, 2006.

[13] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, *Distributed optimization and statistical learning via the alternating direction method of multipliers*, Found. Trends Mach. Learn., 3 (2011), pp. 1–122.

[14] A. Brutzkus and A. Globerson, *Globally optimal gradient descent for a convnet with gaussian inputs*, in Proceedings of the International Conference on Machine Learning, 2017.

[15] J. Cohen, E. Rosenfeld, and Z. Kolter, *Certified adversarial robustness via randomized smoothing*, in Proceedings of the International Conference on Machine Learning, 2019.

[16] S. Diamond and S. Boyd, *CVXPY: A Python-embedded modeling language for convex optimization*, J. Mach. Learn. Res., 17 (2016), pp. 1–5.

[17] S. S. Du, X. Zhai, B. Poczos, and A. Singh, *Gradient descent provably optimizes over-parameterized neural networks*, in Proceedings of the International Conference on Learning Representations, 2019.

[18] D. Dua and C. Graff, *UCI Machine Learning Repository*, University of California, Irvine, 2017.

[19] J. Eckstein and W. Yao, *Approximate ADMM algorithms derived from lagrangian splitting*, Comput. Optim. Appl., 68 (2017), pp. 363–405.

[20] T. Ergen and M. Pilanci, *Global optimality beyond two layers: Training deep ReLU networks via convex programs*, in Proceedings of the International Conference on Machine Learning, 2021.

[21] T. Ergen and M. Pilanci, *Implicit convex regularizers of CNN architectures: Convex optimization of two- and three-layer networks in polynomial time*, in Proceedings of the International Conference on Learning Representations, 2021.

[22] T. Ergen and M. Pilanci, *Path Regularization: A Convexity and Sparsity Inducing Regularization for Parallel Relu Networks*, preprint, https://arxiv.org/abs/2110.09548, 2021.
[23] T. Ergen, A. Sahiner, B. Ozturkler, J. Pauly, M. Mardani, and M. Pilanci, *Demystifying batch normalization in ReLU networks: Equivalent convex optimization models and implicit regularization*, in Proceedings of ICLR 2022.
[24] C. Gallicchio and S. Scardapane, *Deep Randomized Neural Networks*, preprint, https://arxiv.org/abs/2002.12287, 2020.
[25] I. J. Goodfellow, J. Shlens, and C. Szegedy, *Explaining and harnessing adversarial examples*, in Proceedings of the International Conference on Learning Representations, 2015.
[26] M. Grant and S. Boyd, *CVX: Matlab Software for Disciplined Convex Programming, version* 2.1, CVX Research, 2014.
[27] K. He, X. Zhang, S. Ren, and J. Sun, *Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification*, in Proceedings of the IEEE International Conference on Computer Vision, 2015.
[28] M. R. Hestenes, *Multiplier and gradient methods*, J. Optim. Theory Appl., 4 (1969), pp. 303–320.
[29] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, *Extreme learning machine: A new learning scheme of feedforward neural networks*, in Proceedings of the IEEE International Joint Conference on Neural Networks, Vol. 2, 2004, pp. 985–990.
[30] R. Huang, B. Xu, D. Schuurmans, and C. Szepesvári, *Learning with a Strong Adversary*, preprint, https://arxiv.org/abs/1511.03034, 2015.
[31] S. H. Huang, N. Papernot, I. J. Goodfellow, Y. Duan, and P. Abbeel, *Adversarial attacks on neural network policies*, in Proceedings of the International Conference on Learning Representations, 2017.
[32] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, in Proceedings of the International Conference on Learning Representations, 2015.
[33] A. Krizhevsky, *Learning Multiple Layers of Features from Tiny Images*, 2009.
[34] A. Kurakin, I. J. Goodfellow, and S. Bengio, *Adversarial machine learning at scale*, in Proceedings of the International Conference on Learning Representations, 2017.
[35] Y. Wang, J. Lacote, and M. Pilanci, *The Hidden Convex Optimization Landscape of Regularized Two-Layer Re{LU} Networks: An Exact Characterization of Optimal Solutions,* in International Conference on Learning Representations, 2022.
[36] Z. Lu and L. Xiao, *On the complexity analysis of randomized block-coordinate descent methods*, Math. Program., 152 (2015), pp. 615–642.
[37] Z. Ma and S. Sojoudi, *A sequential framework towards an exact SDP verification of neural networks*, in International Conference on Data Science and Advanced Analytics, 2021.
[38] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, *Towards deep learning models resistant to adversarial attacks*, in Proceedings of the International Conference on Learning Representations, 2018.
[39] A. Mishkin, A. Sahiner, and M. Pilanci, *Fast Convex Optimization for Two-Layer ReLU Networks: Equivalent Model Classes and Cone Decompositions*, preprint, https://arxiv.org/abs/2202.01331, 2022.
[40] S. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, *DeepFool: A simple and accurate method to fool deep neural networks*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016.
[41] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, *PyTorch: An imperative style, high-performance deep learning library*, in Advances in Neural Information Processing Systems, 2019.
[42] M. Pilanci and T. Ergen, *Neural networks are convex regularizers: Exact polynomial-time convex optimization formulations for two-layer networks*, in Proceedings of the International Conference on Machine Learning, 2020.
[43] L. Prechelt, *Early stopping—but when?*, in Neural Networks: Tricks of the Trade, Lecture Notes in Comput. Sci. 7700, 2nd ed., Springer, New York, 2012, pp. 53–67.

[44] A. Raghunathan, J. Steinhardt, and P. Liang, *Certified defenses against adversarial examples*, in Proceedings of the International Conference on Learning Representations, 2018.
[45] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, *Learning representations by back-propagating errors*, Nature, 323 (1986), pp. 533–536.
[46] A. Sahiner, T. Ergen, J. M. Pauly, and M. Pilanci, *Vector-output ReLU neural network problems are copositive programs: Convex analysis of two layer networks and polynomial-time algorithms*, in Proceedings of the International Conference on Learning Representations, 2021.
[47] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. J. Goodfellow, and R. Fergus, *Intriguing properties of neural networks*, in Proceedings of the International Conference on Learning Representations, 2014.
[48] G. Taylor, R. Burmeister, Z. Xu, B. Singh, A. Patel, and T. Goldstein, *Training neural networks without gradients: A scalable ADMM approach*, in Proceedings of the 33rd International Conference on Machine Learning, 2016.
[49] J. Wang, F. Yu, X. Chen, and L. Zhao, *ADMM for efficient deep learning with global convergence*, in Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2019.
[50] E. Wong and Z. Kolter, *Provable defenses against adversarial examples via the convex outer adversarial polytope*, in Proceedings of the International Conference on Machine Learning, 2018.
[51] H. Xiao, K. Rasul, and R. Vollgraf, *Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms*, preprint, https://arxiv.org/pdf/1708.07747.pdf, 2017.