Provable defenses against adversarial examples via the convex outer adversarial polytope

J. Zico Kolter
Computer Science Department
Carnegie-Mellon University
Pittsburgh, PA 15213
`zkolter@cs.cmu.edu`

Eric Wong
Machine Learning Department
Carnegie-Mellon University
Pittsburgh, PA 15213
`ericwong@cs.cmu.edu`

Abstract

We propose a method to learn deep ReLU-based classifiers that are provably robust against norm-bounded adversarial perturbations (on the training data; for previously unseen examples, the approach will be guaranteed to detect all adversarial examples, though it may flag some non-adversarial examples as well). The basic idea of the approach is to consider *a convex outer approximation* of the set of activations reachable through a norm-bounded perturbation, and we develop a robust optimization procedure that minimizes the worst case loss over this outer region (via a linear program). Crucially, we show that the dual problem to this linear program can be represented itself as a deep network similar to the backpropagation network, leading to very efficient optimization approaches that produce guaranteed bounds on the robust loss. The end result is that by executing a few more forward and backward passes through a slightly modified version of the original network (though possibly with much larger batch sizes), we can learn a classifier that is provably robust to *any* norm-bounded adversarial attack. We illustrate the approach on a toy 2D robust classification task, and on a simple convolutional architecture applied to MNIST, where we produce a classifier that provably has less than 8.4% test error for any adversarial attack with bounded ℓ_∞ norm less than $\epsilon = 0.1$. This represents the largest verified network that we are aware of, and we discuss future challenges in scaling the approach to much larger domains. All code for experiments is available at `http://github.com/locuslab/convex_adversarial`.

1 Introduction

Recent work in deep learning has demonstrated the prevalence of *adversarial examples* [Goodfellow et al., 2015], data points fed to a machine learning algorithm which are visually indistinguishable from "normal" examples, but which are specifically tuned so as to fool or somehow mislead the machine learning system. Recent history in adversarial classification has followed something of a virtual "arms race": practioners alternatively design new ways of "hardening" classifiers against existing attacks, and then a new class of attacks is developed that can penetrate this defense. Distillation [Papernot et al., 2016] is effective at preventing adversarial examples until it is not [Carlini and Wagner, 2017]. There is no need to worry about adversarial examples under "realistic" settings of rotation and scaling [Lu et al., 2017] until there is [Athalye and Sutskever, 2017]. Neither does the fact that the adversary lacks full knowledge of the model appear to be a problem: "black-box" attacks are also extremely effective [Papernot et al., 2017]. Given the potentially high-stakes nature of many machine learning systems, we feel this situation is untennable: the "cost" of having a classifier be fooled just once is potentially extremely high, and so the attackers are the defacto "winners" of this current game.

We believe that one of the only ways to make progress in usable defenses is to develop networks that are *provably* robust to adversarial attacks. Verification of neural networks is an active research topic

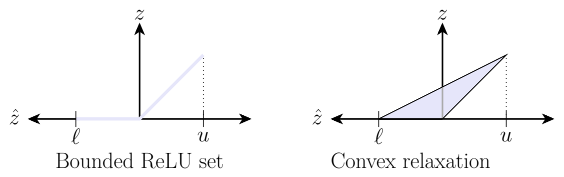

Figure 1: Illustration of the convex ReLU relaxation over the bounded set $[\ell, u]$.

[Huang et al., 2017, Katz et al., 2017, Ehlers, 2017], which typically require a combinatorial search over activation nonlinearities (often via a satisfiability solver) combined with continuous search over the linear regions. But these typically involve hard combinatorial optimization problems that likely will never scale to the size of realistic networks.

In this work we take a different approach: rather than try to verify a "hard" network, we want to learn a network that is inherently *easy* to verify. Specifically, we consider the *adversarial polytope* of a ReLU-based deep network, the set of all final-layer activations achievable through a perturbation of the input. Although this is a highly non-convex set, we can consider a *convex outer approximation* of the polytope, and efficiently optimize over this outer bound; if no adversarial example exists within the outer bound, none can exist within the actual adversarial polytope either. We specifically consider a convex outer bound where optimization over this bound takes the form of a linear program. Crucially however (since repeatedly solving an LP with the number of variables equal to the number of activations is not practical for large networks), we show that we can write the dual problem of this LP in the form of another neural network, similar to the backpropagation network, and which provides guaranteed bounds on the optimal solution of the linear program. The end result is that via a few more forward and backward passes through the network (though potentially with larger batch sizes, to compute the necessary activation bounds), we can make guaranteed statements about the error that *any* adversarial attack can achieve.

Finally, we integrate our approach within a robust learning framework. Robust optimization [Ben-Tal et al., 2009] has been recently brought to the forefront of research in adversarial examples [Madry et al., 2017], but this work focuses on solving the robust optimization problems heuristically, and does not provide guarantees. By using our convex outer bound, we can optimize a classifier's worst-case loss over the entire outer convex approximation; if we are able to achieve suitably low training error, this implies that the classifier is robust to *any* adversarial attack. A similar approach was proposed concurrently to this work Anonymous [2018], though with a very different method, and we discuss connections in Appendix A. We illustrate the results on a small toy problem and on a convolutional network applied to MNIST, where we produce a classifier with a guarantee of less than 8.39% error for any norm-bounded perturbations of size $\epsilon = 0.1$; although still small scale, this latter case represents the largest verified network that we are aware of.

2 Methodology

To fix notation, we consider a k layer feedforward ReLU-based neural network, $f_\theta : \mathbb{R}^{|x|} \to \mathbb{R}^{|y|}$ given by the equations

$$\begin{aligned} \hat{z}_{i+1} &= W_i z_i + b_i,\ i = 1, \ldots, k-1 \\ z_i &= \max\{\hat{z}_i, 0\}, i = 2, \ldots, k-2 \\ z_1 &= x \\ f_\theta(x) &\equiv \hat{z}_k \end{aligned} \tag{1}$$

where use $\theta = \{W_1, \ldots, W_{k-1}, b_1, \ldots, b_{k-1}\}$ to denote the set of all parameters of the network, and where W_i represent a linear operator such as matrix multiply or convolution. We use the set $\mathcal{Z}_\epsilon(x)$ to denote the set of all final-layer activations attainable by perturbing x by some Δ with ℓ_∞ norm bounded by ϵ. $\mathcal{Z}_\epsilon(x)$ is a non-convex set (it can be represented via an integer program as in [Lomuscio and Maganti, 2017]), so cannot easily be optimized over.

A convex outer bound The starting point of our approach is to consider a convex outer bound on the adversarial polytope. Specifically, given known lower and upper bounds ℓ, u for the pre-ReLU activation, we can replace the ReLU constraints with their upper convex envelope, as shown in Figure

5; the same relaxation at the activation level was used in [Ehlers, 2017] though there it was used as a sub-step for exact (combinatorial) verification of networks, and the procedure for actually computing the crucial bounds ℓ and u is different. We denote this outer bound $\tilde{\mathcal{Z}}_\epsilon(x)$.

Efficient optimization The problem of finding the "most adversarial" example within the outer convex adversarial polytope can be cast as the following optimization problem

$$\underset{\hat{z}_k}{\text{minimize}}\, c^T \hat{z}_k, \;\; \text{subject to}\, \hat{z}_k \in \tilde{\mathcal{Z}}_\epsilon(x) \tag{2}$$

where c is a vector implying which activations we want to minimize and which we want to maximize; with the ReLU approximation above this problem becomes a linear program, which theoretically could be solved with off-the-shelf solvers. However, although LP solvers are "efficient", the resulting problems have a optimization variable associated with each hidden unit in the network, which would not be practically feasible to solve. Instead, we focus on the dual problem of this LP. As shown in the Appendix, we can show that the feasible set of the dual problem can *also* be expressed as a deep network, namely the network

$$\begin{aligned}
\nu_k &= -c \\
\hat{\nu}_i &= W_i^T \nu_{i+1}, i = k-1, \ldots, 1 \\
\nu_{i,j} &= \begin{cases} 0 & j \in \mathcal{I}_i^- \\ \hat{\nu}_{i,j} & j \in \mathcal{I}_i^+ \\ \frac{u_{i,j}}{u_{i,j}-\ell_{i,j}}[\hat{\nu}_{i,j}]_+ - \alpha_{i,j}[\hat{\nu}_{i,j}]_- & j \in \mathcal{I}_i \end{cases}
\end{aligned} \tag{3}$$

where α is a set of free variables, and where $\mathcal{I}_i^-$, $\mathcal{I}_i^+$, and $\mathcal{I}_i$ denote the set of activations where the lower and upper bounds are both negative, both positive, or span zero respectively; this in fact is extremely similar to the backpropagation network for ReLU-based activations, except for the terms in $\mathcal{I}_i$. The objective term of the dual problem is then given by

$$J(\nu) = -\sum_{i=1}^{k-1} \nu_{i+1}^T b_i - x^T \hat{\nu}_1 - \epsilon \|\hat{\nu}_1\|_1 + \sum_{i=2}^{k-1} \sum_{j \in \mathcal{I}_i} \ell_{i,j}[\nu_{i,j}]_+ \tag{4}$$

which therefore, for *any* assignment to the α's, gives a guaranteed lower bound on the original LP (despite the fact that this formulation of the dual is not convex).

Computing activation bounds Finally, we note that we can use this dual problem formulation to iteratively find the activation lower and upper bounds $\ell_{i,j}$ and $u_{i,j}$ by choosing $c = I$ or $c = -I$. This can be made practical by 1) choosing a fixed solution $\alpha_{i,j} = u_{i,j}/(u_{i,j} - \ell_{i,j})$ to use for computing all bounds; and 2) by caching terms in the dual network computation. The end result is that we can compute bounds for *all* layers via a single forward pass through the network, albeit at the cost of running a single example through the network *for each input dimension*. This is admittedly the poorest-scaling aspect of our approach, and comes from the non-linear terms in (4). However, a number of approaches to scaling this to larger-sized inputs is possible, including bottleneck layers earlier in the network, e.g. PCA processing of the images, probabilistic norm bounds, or other similar constructs.

Robust optimization After all this has been done, we can replace the standard empirical loss minimization with a guaranteed upper bound

$$\underset{\theta,\alpha}{\text{minimize}} \; \sum_{i=1}^{N} L(-J(g_{\theta,x_i,\epsilon}(I - e_{y_i}1^T, \alpha_i)), y_i) \tag{5}$$

where $g_{\theta,x_i,\epsilon}$ denotes the dual network and L denotes a loss function. Essentially, we are replacing the loss in the minimization problem with a guaranteed bound derived from the dual problem. All the network terms, including the upper and lower bound computation, are differentiable, so the whole optimization can be solved with any standard stochastic gradient variant, and the result is a network that (if we achieve low loss) is guaranteed robust to adversarial examples.

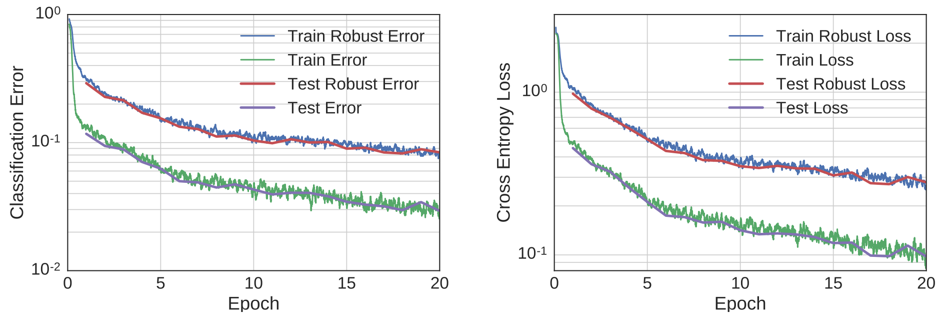

Figure 2: Training progress for our method applied to a simple ConvNet architecture on MNIST. Robust error and loss indicate provable upper bounds on the loss/error achievable by any adversarial perturbation with ℓ_∞ norm bounded by $\epsilon = 0.1$.

3 Experiments

We present experimental results of robustly training a convolutional classifier on the MNIST data set. Code for the experiments here and additional experiments on a simple 2D robust classification task is available at `http://github.com/locuslab/convex_adversarial`. We use a simple LeNet-style network, with maxpooling layers replaced by stride-2 convolutions and ReLUs after all layers, and we use Adam Kingma and Ba [2015] to optimize network parameters. Specifically, our network is of the form:

$$x \Rightarrow \text{Conv\_16x4x4,s=2} \Rightarrow \text{ReLU} \Rightarrow \text{Conv\_16x4x4,s=2} \Rightarrow \text{ReLU} \Rightarrow \text{FC100} \Rightarrow \text{ReLU} \Rightarrow y. \quad (6)$$

Figure 9 shows the training progress using our procedure with a robust softmax loss function and $\epsilon = 0.1$. The "robust error" and "robust loss" here are our bounds on the robust error and loss; that is, we know that any norm-bounded adversarial technique will not be able to achieve loss or error that is any higher. The final classifier after 20 epochs reaches a test error of 2.93% with a robust test error of 8.39%. We also tested our network against two common classes of attacks: the fast gradient sign method Goodfellow et al. [2015], and the projected gradient descent approach Madry et al. [2017]. For a traditionally-trained classifier (with 1.2% test error) the FGSM approach results in 39.7% error, while PGD results in 94.0% error. On the classifier trained with our method, however, we only achieve 2.93% test error, but FGSM and PGD only achieve errors of 5.8% and 6.2% respectively (both, naturally, below our bound of 8.4%). These results are summarized in Table 2. Although these results are relatively small-scale, the somewhat surprising ability here is that by just considering a few more forward/backward passes in a modified network to compute an alternative loss, we can derive *guaranteed* error bounds for any adversarial attack. This represents the largest verified classifier that we are aware of, and by avoiding any combinatorial optimization, has the potential to scale to much larger problems still.

4 Conclusion

This work proposed a simple method for training a verifiable network with guaranteed bounds on adversarial error. Though we believe it represents a significant step in dealing with adversarial examples, future work remains in 1) getting the method to scale to state-of-the-art-sized networks, and 2) in characterizing adversarial inputs beyond simple norm-bounds. Further, the generic techniques of bounding optimization over a neural network, using a dual network, is likely to find additional applications in deep learning, as many problems involve optimizing over inputs to a network.

Table 1: Errors for a standard and our robustly-trained classifier on the MNIST test set.

Network	**Non-adversarial**	**FGSM**	**PGD**	**Robust Bound**
Standard training	*1.2%*	39.7%	94.0%	100%
Robust training	2.9%	*5.8%*	*6.2%*	*8.4%*

References

Anonymous. Certified defenses against adversarial examples. In *Under review at ICLR 2018*, 2018.

Anish Athalye and Ilya Sutskever. Synthesizing robust adversarial examples. *arXiv preprint arXiv:1707.07397*, 2017.

Aharon Ben-Tal, Laurent El Ghaoui, and Arkadi Nemirovski. *Robust optimization*. Princeton University Press, 2009.

Emmanuel J Candes and Terence Tao. Near-optimal signal recovery from random projections: Universal encoding strategies? *IEEE transactions on information theory*, 52(12):5406–5425, 2006.

Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *Security and Privacy (SP), 2017 IEEE Symposium on*, pages 39–57. IEEE, 2017.

Moustapha Cisse, Piotr Bojanowski, Edouard Grave, Yann Dauphin, and Nicolas Usunier. Parseval networks: Improving robustness to adversarial examples. In *International Conference on Machine Learning*, pages 854–863, 2017.

Ruediger Ehlers. Formal verification of piece-wise linear feed-forward neural networks. In *International Symposium on Automated Technology for Verification and Analysis*, 2017.

Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations*, 2015. URL `http://arxiv.org/abs/1412.6572`.

Xiaowei Huang, Marta Kwiatkowska, Sen Wang, and Min Wu. Safety verification of deep neural networks. In *International Conference on Computer Aided Verification*, pages 3–29. Springer, 2017.

Guy Katz, Clark Barrett, David Dill, Kyle Julian, and Mykel Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. *arXiv preprint arXiv:1702.01135*, 2017.

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

Alessio Lomuscio and Lalit Maganti. An approach to reachability analysis for feed-forward relu neural networks. *arXiv preprint arXiv:1706.07351*, 2017.

Jiajun Lu, Hussein Sibai, Evan Fabry, and David Forsyth. No need to worry about adversarial examples in object detection in autonomous vehicles. *arXiv preprint arXiv:1707.03501*, 2017.

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. Distillation as a defense to adversarial perturbations against deep neural networks. In *Security and Privacy (SP), 2016 IEEE Symposium on*, pages 582–597. IEEE, 2016.

Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. Practical black-box attacks against deep learning systems using adversarial examples. In *Proceedings of the 2017 ACM Asia Conference on Computer and Communications Security*, 2017.

Huan Xu, Constantine Caramanis, and Shie Mannor. Robustness and regularization of support vector machines. *Journal of Machine Learning Research*, 10(Jul):1485–1510, 2009.

Appendix

In this appendix we provide a much more detailed description of the general adversarial polytope, our convex outer approximation and algorithms, and further experimental results.

A Relation to concurrent work

Concurrently with this publication, Anonymous [2018] released a paper on similar methods for learning classifiers provably robust to adversarial examples. Although very similar at a high level (both techniques employ an outer bound on the adversarial polytope, and use the dual problem to provide a guaranteed bound on optimization), the actual bound and methodology of the two approaches is quite different. In particular, while this paper proposes a linear programming based outer bound on the adversarial polytope, the other work proposes a semidefinite programming based bound, and currently focuses only on the setting of two-layer networks. While SDP relaxations in general have the ability to be tighter than LP relaxations, the notable advantage of the LP relaxation here is that it allows for a simple bound of a dual problem via a dual network; in contrast the SDP bound requires a more complicated optimization scheme that involves optimizing the eigenvalue of a particular matrix via a Lanczos iteration, and it somewhat unclear to us how the approach extends to multilayer networks and/or operators like convolutions that are too expensive to express in their matrix form (though extensions to the SDP bound may be possible to handle these cases). Finally in terms of the reported results themselves, this other work is able to achieve a provable bound of 35% error on MNIST with $\epsilon = 0.1$, while the approach we present achieves a provable bound of 8.4% error.

B The adversarial polytope

We begin by considering a characterization of the exact adversarial polytope of a deep network; a very similar approach is used in Lomuscio and Maganti [2017], though we include it here for completeness, as the adversarial polytope plays a foundational role in our subsequent main contribution in the following section.

To fix notation, we consider a k layer feedforward ReLU-based neural network, $f_\theta : \mathbb{R}^{|x|} \to \mathbb{R}^{|y|}$, where the output of the network is given by the pre-softmax final-layer activations (so that they would correspond to the logits of a multi-class classification architecture). The network structure is given by the equations

$$\begin{aligned} \hat{z}_{i+1} &= W_i z_i + b_i,\ i = 1, \ldots, k-1 \\ z_i &= \max\{\hat{z}_i, 0\}, i = 2, \ldots, k-2 \\ z_1 &= x \\ f_\theta(x) &\equiv \hat{z}_k \end{aligned} \tag{7}$$

where we will use $\theta = \{W_1, \ldots, W_{k-1}, b_1, \ldots, b_{k-1}\}$ to denote the set of all parameters of the network. Although we write the linear terms in the network as matrix operations $W_i z_i + b_i$, we note that W_i could be any linear operator such as a convolution (and we will indeed consider convolutions in our experimental results). We also are assuming that the network here only has ReLU non-linearities. The method we propose can potentially be extended to other polytopic activations such as max-pooling or leaky ReLUs, but we leave this for future work.[1] We also note that we can integrate more complex networks that have passthrough layers or more complex structure, but again we use the simple feedforward network for simplicity of notation throughout the paper.

Now we consider a set of allowable perturbations $x + \Delta$ where $\|\Delta\|_\infty \leq \epsilon$ (we use the ℓ_∞ norm to bound adversarial examples, as this is one of the most common settings in practice, though other norms are possible as well). The *adversarial polytope* is defined as *the set of all possible z_k terms achievable under this perturbation*

$$\mathcal{Z}_\epsilon(x) = \{\hat{z}_k | \hat{z}_k = f(x + \Delta), \|\Delta\|_\infty \leq \epsilon\} \tag{8}$$

The conceptual idea of the adversarial polytope is illustrated in Figure 3. This set provides us with all the information we need about adversarial examples (at least those defined by the simple perturbations defined above, which already encompass a powerful set). We will discuss this in more detail shortly, but optimizing over the adversarial polytope can tell us, for instance, whether there is any example within ϵ of x that can change the class of the example. It also forms the basis for developing a robust

[1] We can also potentially integrate other non-piecewise-linear activations such as sigmoids, though this requires an approximation if we do want to use polyhedral methods, so we focus here on piecewise-linear activations.

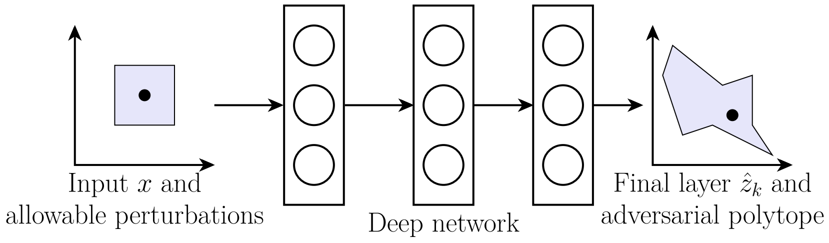
Input x and allowable perturbations
Deep network
Final layer $\hat{z}_k$ and adversarial polytope

Figure 3: Illustration of the adversarial polytope (i.e., the "reachable" region for a given allowable perturbation).

optimization approach that guarantees that no adversarial examples (at least on the training data) are possible.

Although it may not be apparent, $\mathcal{Z}_\epsilon(x)$ is a connected polytope, a property which follows from the fact that the entire function is composed of linear operations and the non-linear ReLU operator, which is still defined by a (non-convex) polytope: for $y = \mathrm{ReLU}(x)$, the ReLU is defined by the set $(y = 0 \wedge x \leq 0) \vee (y = x \wedge x > 0))$. Indeed, we can express the true adversarial polytope in terms of the *bilinear integer constraints*

$$\begin{aligned} \hat{z}_{i+1} &= W_i z_i + b_i,\ i = 1, \ldots, k-1 \\ z_i &= \hat{z}_i \cdot y_i,\ i = 2, \ldots, k-1 \\ z_1 &\leq x + \epsilon \\ z_1 &\geq x - \epsilon \\ y_i &\in \{0, 1\}^{|z_i|} \end{aligned} \tag{9}$$

where y_i is a set of binary integer variables that encode "which side" of the ReLU the pre-ReLU activations $\hat{z}_i$ lies on. Although this is not in a form that can be easily handled by existing IP solvers because of the bilinear term $\hat{z}_i \cdot y_i$, it can be put into an allowable form using a standard trick from integer programming know as *linearization*. Specifically, supposing that we have some lower and upper bounds ℓ_i, u_i on the pre-ReLU activations $\hat{z}_i$ (for the integer programming formulations, these bounds can be very loose, so can be derived for instance from Lipschitz bounds of the network), then the above set of constraints is equivalent to the mixed integer constraints (we don't cover this transformation in detail here, but it is a very common pattern in the IP literature)

$$\begin{aligned} \hat{z}_{i+1} &= W_i z_i + b_i,\ i = 1, \ldots, k-1 \\ \hat{z}_i &\leq u_i \cdot y_i,\ i = 2, \ldots, k-1 \\ z_i &\geq \hat{z}_i,\ i = 2, \ldots, k-1 \\ z_i &\geq 0,\ i = 2, \ldots, k = 1 \\ z_i &\leq \hat{z}_i - (1 - y_i) \cdot \ell_i \\ z_1 &\leq x + \epsilon \\ z_1 &\geq x - \epsilon \\ y_i &\in \{0, 1\}^{|z_i|}. \end{aligned} \tag{10}$$

It is simple to check that if $y_{i,j} = 0$, this implies the corresponding $z_{i,j} = 0$ (the constraints simplify to $z_{i,j} \geq 0$ and $z_{i,j} \leq 0$); similarly, for $y_{i,j} = 1$, the constraints imply that $z_{i,j} = \hat{z}_{i,j}$. This form, which now can be fed to off-the-shelf solvers, lets us reasonable about the exact nature of adversarial examples for a classifier.

B.1 Queries within the adversarial polytope

The adversarial polytope allows us to formalize the notion of finding or preventing adversarial examples in a classifier. Considering some datapoint/label pair (x, y), as well as a *target class* y_{targ} (the class that we are going to attempt to fool the classifier into producing), we can determine whether or not there exists an adversarial example (within ϵ ℓ_∞ norm of x) that makes the label y_{targ} *more* *likely* that y by solving the optimization problem

$$\text{minimize } (z_k)_y - (z_k)_{y_{\text{targ}}}, \ \text{ subject to } \ z_k \in \mathcal{Z}_\epsilon(x). \tag{11}$$

If the optimal objective of this optimization problem is less than 0, then there *does* exist an adversarial example that makes the label y_{targ} more likely than y; this follows immediately from the nature of the optimization problem: if the objective is negative, then the value of the y_{targ} activation is greater than the y activation, which guarantees the desired property. This optimization problem is a mixed integer linear program, and (using the polytope formulation provided in (10)) can be solved by a number of off-the-shelf integer programming solvers. Other questions can be answered as well: for instance if we want to find the minimum-perturbation adversarial example possible, then we can solve the MILP

$$\text{minimize } \epsilon, \ \text{ subject to } (z_k)_y \le (z_k)_{y_{\text{targ}}}, \ z_k \in \mathcal{Z}_\epsilon(x). \tag{12}$$

Besides creating adversarial examples, the adversarial polytope can also be used to *test* the potential adversarial nature of unknown examples. If we are provided an example $x \in \mathbb{R}^n$ as input to a classifier, a natural question to ask is: is this potentially an adversarial example, e.g., an example that has been crafted to fool our classifier. While it is difficult to answer this question in general, in some cases it *is* possible to guarantee that a particular example is *not* adversarial, at least under the set of allowable perturbations. Specifically, if we let $y_{\text{pred}} = \text{argmax}_i\, f_i(x)$ be the predicted class of the example, then we can check (using the approach above) whether there are adversarial examples for all $y_{\text{targ}} \neq y_{\text{pred}}$. If no such example exists, then *we know that the network classifies the sample x as being the same class for* all *ϵ perturbations*, i.e., it could not be an instance of a "good" example (which would be classified correctly) perturbed by ϵ to be classified incorrectly. This allows us a method for guaranteeing that an example is not adversarial, even if we cannot do the converse, guarantee an example that does cross classes within the adversarial polytope is actually adversarial. However, because we know from experience that most deep classifiers *are* easy to fool, it is likely that a typically-trained network will flag most inputs as being potentially adversarial; thus, as we will address next, we need additional methods for training classifiers such that they are likely to produce classifiers not sensitive to adversarial perturbations.

B.2 Robust optimization and learning robust classifier

Finally, we discuss how we can use the adversarial polytope to develop "hardened" classifiers (more) robust to adversarial examples, based upon approaches in robust optimization. Robust optimization in the context of linear classifiers has been used for some time within machine learning (e.g Xu et al. [2009]). The basic idea is that, given a training set of x_i, y_i pairs, instead of simply minimizing the loss at these data points, we minimize the loss at the *worst* location (i.e. that with the highest loss) in an ϵ ball around each x_i. Assuming a linear classifier $h_\theta(x) = \theta^T x$ (and binary classification, for simplicity), this results in the optimization problem

$$\underset{\theta}{\text{minimize}} \ \sum_{i=1}^{N} \max_{\|\Delta\| \le \epsilon} \ell(\theta^T(x+\Delta)\cdot y_i) \tag{13}$$

Using the the fact that

$$\max_{\|\Delta\| \le \epsilon} \theta^T \Delta = \epsilon \|\theta\|_* \tag{14}$$

where $\|\cdot\|_*$ denotes the dual norm, the robust optimization problem can be simplified as

$$\underset{\theta}{\text{minimize}} \ \sum_{i=1}^{N} \max_{\|\Delta\| \le \epsilon} \ell(\theta^T x \cdot y_i + \|\theta\|_*). \tag{15}$$

As mentioned above, recent work has highlighted the connection between adversarial classification and robust optimization Madry et al. [2017], in particular attempting to solve the non-convex analogue of the native min-max problem (13) by (projected) gradient descent over θ and Δ; this connection to robust optimization was also discussed in the original adversarial example paper Goodfellow et al. [2015], with the fast gradient sign method reducing to the same solution of this min-max problem for a linear classifier.

To consider how to use the adversarial polytope to more exactly formulate the robust optimization problem, we consider training a deep classifier with a multiclass hinge loss; other losses such as the

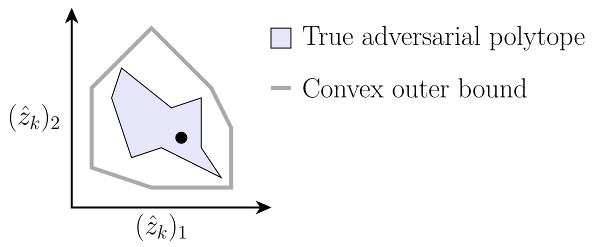

Figure 4: Simple conceptual illustration of the (non-convex) adversarial polytope, and an outer convex bound.

more common cross-entropy loss can be easily approximated, but the connection is more obvious with this loss. Optimizing such a deep network is given by the optimization problem

$$\underset{\theta}{\text{minimize}} \quad \sum_{i=1}^{N} [1 + \max_{y \neq y_i} f_y(x) - f_{y_i}(x)]_+. \tag{16}$$

This loss is zero if the true class output $f_{y_i}(x)$ is at least 1 larger than any alternative class output $f_y(x)$ for $y \neq y_i$, and scales linearly with the largest activation other than the true class otherwise. Because the term $f_y(x) - f_{y_i}(x)$ is exactly the objective that we considered previously, we can encode the min-max formulation of robust optimization by considering $|y| - 1$ different classification problems (one for each alternative label $y \neq y_i$) over the adversarial polytope

$$\begin{aligned} &\underset{\theta}{\text{minimize}} \quad \sum_{i=1}^{N} \max_{\|\Delta\|_\infty \leq \epsilon} [1 + \max_{y \neq y_i} f_y(x+\Delta) - f_{y_i}(x+\Delta)]_+ \\ \equiv\, &\underset{\theta}{\text{minimize}} \quad \sum_{i=1}^{N} [1 + \max_{y \neq y_i} \max_{\hat{z} \in \mathcal{Z}_\epsilon}(x_i) \left((\hat{z}_k)_y - (\hat{z}_k)_{y_i}\right)]_+. \end{aligned} \tag{17}$$

The inner maximization is precisely the optimization problem we considered earlier of finding the most extreme point within the adversarial polytope. If this objective can be minimized with zero training loss, then we are guaranteed that for every data point in the training set, no adversarial example is possible (all points within the adversarial polytope are labeled correctly by the classifier). Test points naturally have no such guarantee, but we can still use the method described above to test whether or not they are potentially adversarial. And if empirical generalization patterns for deep networks remain in this setting, it is likely that classifiers which don't admit to adversarial examples on the training set may well have similar properties on the test set (and we will demonstrate this empirically ourselves shortly).

C Convex outer bounds on the adversarial polytope

Despite all the promise of the preceding section, there is a notable disadvantage that renders this approach more a conceptual strategy rather than an actual tool: the fact that actually *solving* a mixed binary integer program is an NP-hard task. Although it is well-appreciated that many instances of reasonably-sized binary integer programs *are* empirically tractable via branch and cut algorithms, in this case the optimization problem of interest has a number of binary variables equal to the number of hidden units in the network, a number that can easily grow into the millions for large deep networks. This is simply not tractable to solve via exact solutions to the integer program.

Instead, the key claim of this section is that we can instead consider a convex relaxation of the adversarial polytope that is a strict *outer bound*. This concept is illustrated in Figure 4. That is, it contains all points (and more) within the adversarial polytope, but is a convex set and so can be optimized over efficiently (at least according to some definition of "efficient", which will be a major discussion point throughout this section). The key point is that because of this outer bound property, *we can perform the same tests for potential adversarial examples or develop the same robust optimization techniques, which will provide identical guarantees*. That is, if we look for the "most adversarial" example over our outer approximation, and find that it does not change the predicted

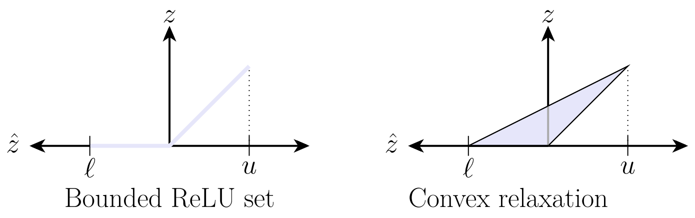

Figure 5: Illustration of the convex ReLU relaxation over the bounded set $[\ell, u]$.

class of an example, then we are guaranteed that no actual adversarial example can change the class either. Or we can develop robust classifiers that optimize over the worst-case loss within the outer approximation, and these (assuming zero training loss) also will guarantee that the classifier does not allow for adversarial examples. [2].

In this section we will first define our convex outer approximation to the adversarial polytope, and discuss the optimization problem that arises when optimizing over it. However, although this is a convex optimization problem and hence "tractable", it is far from practically efficient to solve this optimization problem exactly for large networks. Thus, in the next section we consider the *dual problem* of optimization over the convex outer bound. By standard results in convexity, any dual feasible solution represents a *lower bound* on minimization over the convex adversarial approximation, and as we show, the dual problem itself can be formulated as a deep network that is effectively a type of "adjoint system" of the original network, and which easily allows us to attain dual feasible points; thus, we can employ simple gradient descent methods to optimize this dual problem, giving us guaranteed lower bounds on the original optimization problem, even if we don't solve the dual problem to optimality. Next, we discuss how this dual framework can be fit into the context of learning a robust classifier, showing that we can indeed solve the robust classifier learning problem using the dual problem. Finally, we discuss methods for finding the necessary upper and lower bounds needed for the convex outer approximation.

C.1 A convex outer bound

Note that the definition of our adversarial polytope involved two types of terms: the linear terms $\hat{z}_{i+1} = W_i z_i + b_i$ (which are convex), and the non-convex ReLU terms $z_i = \max\{0, \hat{z}_i\}$. Thus, to form a convex relaxation of this set, we need to replace the ReLU terms by some relaxed version.

The idea that underlies our convex outer bound is quite simple, and indeed has been previously considered in the context of searching over the actual adversarial polytope Ehlers [2017]; however, as mentioned above, this previous work focused on the outer bound as a useful step in searching over the true adversarial polytope for a given classifier, while we consider it here as an outer bound that we will explicitly train over. However, we do emphasize that the formalism in this section at the single activation level is virtually identical to that previously proposed in this prior work, though because our full bound will also involve iterative computation of upper and lower bounds for intermediate activations, our final outer bound ends up being quite different.

Suppose that for some pre-ReLU activation $\hat{z}$ and adversarial polytope $\mathcal{Z}_\epsilon(x)$ we have an upper and lower bound ℓ and u over the allowable values for the activation (in the remainder of this paragraph, for simplicity, we'll let $\hat{z}$ and z denote scalar values for a *single* activation, which we'll later extend elementwise to all activations). Note that these upper and lower bounds are guaranteed to exist, as perturbation of ϵ of the input can only change the activations of the network by some bounded amount (this is effectively the "adversarial" polytope but for inner activations of the network). Then we can replace the non-convex ReLU constraint by it's convex hull over the allowable upper and lower bounds. This reduces to three cases:

1. If $\ell \leq u \leq 0$, then we can replace the constraint $z = \max\{0, \hat{z}\}$ with the constraint $z = 0$, as the ReLU will always lie in the zero region.

[2]Note, however, that the outer approximation is less useful for actually find adversarial examples: as we will see, because elements in this polytope may not correspond to any actual path through the original network, it is unclear what if anything can be said about the actual "examples" found via this method. However, given the relative ease of finding versus preventing adversarial examples in modern deep architectures, this is not overly bothersome

2. Similarly, if $0 \leq \ell \leq u$, then we can replace the constraint with $z = \hat{z}$, as the ReLU will always lie in the linear region.

3. In the remaining case (the "interesting" non-convex case), where $\ell < 0 < u$, we can replace the non-convex ReLU constraint with the convex hull over the ReLU function over the domain $[\ell, u]$. This is shown in Figure 5, and the constraint set in this case can represented by the three linear inequalities

$$z \geq 0,\ z \geq \hat{z}, -u\hat{z} + (u - \ell)z \leq -u\ell. \tag{18}$$

With this relaxation as our starting point, we can formally define our relaxation of the adversarial polytope, which we denote $\tilde{\mathcal{Z}}_\epsilon^{[\ell,u]}(x)$ (here ℓ and u signify tensors of upper and lower bounds for each activation in the network, which are naturally going to be different for different activations in the network, and which we discuss how to compute shortly) as the following set of linear constraints

$$\begin{aligned}
&\hat{z}_{i+1} = W_i z_i + b_i,\ i = 1, \ldots, k-1 \\
&z_1 \leq x + \epsilon \\
&z_1 \geq x - \epsilon \\
&z_{i,j} = 0,\ i = 2, \ldots, k-1, j \in \mathcal{I}_i^- \\
&z_{i,j} = \hat{z}_{i,j},\ i = 2, \ldots, k-1, j \in \mathcal{I}_i^+ \\
&\left.\begin{aligned}
&z_{i,j} \geq 0, \\
&z_{i,j} \geq \hat{z}_{i,j}, \\
&-u_{i,j}\hat{z}_{i,j} + (u_{i,j} - \ell_{i,j})z_{i,j} \leq -u_{i,j}\ell_{i,j}
\end{aligned}\right\}\ i = 2, \ldots, k-1, j \in \mathcal{I}_i
\end{aligned} \tag{19}$$

where $\mathcal{I}_i^-$, $\mathcal{I}_i^+$, and $\mathcal{I}_i$ denote the set of activations in the ith layer that fit the categories 1, 2, and 3 above respectively. Intuitively, our convex outer approximation allows some flexibility in choosing activations as subsequent layers of the network. For activations that can span over zero, the "post-ReLU" activation need not exactly equal the ReLU of the input, but can lie anywhere between the ReLU and the line connecting the maximum to minimum values; this extra freedom allows us to pick examples that are "more adversarial" at later layers of the network, because they may not correspond to any actual input fed through the network, but instead will correspond to allowing the intermediate activations in a network to be adjusted to make the example as adversarial as possible.

At this point, we could theoretically use our convex outer approximation in lieu of the true adversarial polytope, and obtain a tractable formulation to identify potential adversarial examples or learn a robust classifier. However, two key challenges arise in this approach:

1. We need a way of optimizing over the adversarial polytope efficiently, that is, solving the optimization problem

$$\text{minimize } c^T \hat{z}_k,\ \text{ subject to } \hat{z}_k \in \tilde{\mathcal{Z}}_\epsilon^{[\ell,u]}(x). \tag{20}$$

   Although this is now a convex linear program, it still has the number of variables equal to the number of hidden units in the network, and standard solution methods for LPs would be intractable for all but very small networks.

2. We need some way of finding the lower and upper bounds ℓ and u. Simple Lipschitz bounds are extremely loose here (a fact we will demonstrate shortly), and provide no real use. We need some alternative way of finding these bounds that will eventually provide a meaningful convex outer bound.

C.2 Efficient optimization via the dual problem

In this section we deal with the first of the two problems above, and provide an efficient method for solving (or more precisely, for bounding the optimal value of) the optimization problem above, written explicitly as

$$\begin{aligned}
\text{minimize} \quad & c^T \hat{z}_k \\
\text{subject to} \quad & \hat{z}_{i+1} = W_i z_i + b_i,\ i = 1, \ldots, k-1 \\
& z_1 \leq x + \epsilon \\
& -z_1 \leq -x + \epsilon \\
& z_{i,j} = 0,\ i = 2, \ldots, k-1, j \in \mathcal{I}_i^- \\
& z_{i,j} = \hat{z}_{i,j},\ i = 2, \ldots, k-1, j \in \mathcal{I}_i^+ \\
& \left.\begin{aligned} & -z_{i,j} \leq 0, \\ & \hat{z}_{i,j} - z_{i,j} \leq 0, \\ & -u_{i,j}\hat{z}_{i,j} + (u_{i,j} - \ell_{i,j}) z_{i,j} \leq -u_{i,j}\ell_{i,j} \end{aligned}\right\} i = 2, \ldots, k-1, j \in \mathcal{I}_i
\end{aligned} \tag{21}$$

The basic idea we propose is to consider the dual of this linear program. By standard results in convex analysis, any feasible solution of the dual problem provides a guaranteed lower bound on the optimal objective of the primal problem. Crucially, we will show that the dual problem can be characterized via a "dual network", similar to the backpropagation network through the original graph, but with additional free parameters that can be optimized over with e.g. gradient descent. And although we don't expect to find an optimal solution to the dual problem in this manner, it will still provide the guarantee we need.

In detail, we associate the following dual variables with each of the constraints

$$\begin{aligned}
\hat{z}_{i+1} = W_i z_i + b_i &\Rightarrow \nu_{i+1} \in \mathbb{R}^{|\hat{z}_{i+1}|} \\
z_1 \leq x + \epsilon &\Rightarrow \xi^+ \in \mathbb{R}^{|x|} \\
-z_1 \leq -x + \epsilon &\Rightarrow \xi^- \in \mathbb{R}^{|x|} \\
-z_{i,j} \leq 0 &\Rightarrow \mu_{i,j} \in \mathbb{R} \\
\hat{z}_{i,j} - z_{i,j} \leq 0 &\Rightarrow \tau_{i,j} \in \mathbb{R} \\
-u_{i,j}\hat{z}_{i,j} + (u_{i,j} - \ell_{i,j}) z_{i,j} \leq -u_{i,j}\ell_{i,j} &\Rightarrow \lambda_{i,j} \in \mathbb{R}
\end{aligned} \tag{22}$$

where we note that can easily eliminate the dual variables corresponding to the $z_{i,j} = 0$ and $z_{i,j} = \hat{z}_{i,j}$ from the optimization problem, so we don't define explicit dual variables for these; we also note that $\mu_{i,j}$, $\tau_{i,j}$, and $\lambda_{i,j}$ are only defined for i, j such that $j \in \mathcal{I}_i$, but we keep the notation as above for simplicity. With these definitions, the dual problem becomes

$$\begin{aligned}
\text{maximize} \quad & -\sum_{i=1}^{k-1} \nu_{i+1}^T b_i - (x+\epsilon)^T \xi^+ + (x-\epsilon)^T \xi^- + \sum_{i=2}^{k-1} \lambda_i^T (u_i \ell_i) \\
\text{subject to} \quad & \nu_k = -c \\
& \nu_{i,j} = 0,\ j \in \mathcal{I}_i^- \\
& \nu_{i,j} = (W_i^T \nu_{i+1})_j,\ \ j \in \mathcal{I}_i^+ \\
& \left.\begin{aligned} & (u_{i,j} - \ell_{i,j})\lambda_{i,j} - \mu_{i,j} - \tau_{i,j} = (W_i^T \nu_{i+1})_j \\ & \nu_{i,j} = u_{i,j}\lambda_{i,j} - \mu_i \end{aligned}\right\} i = 2, \ldots, k-1, j \in \mathcal{I}_i, \\
& W_1^T \nu_2 = \xi^+ - \xi^- \\
& \lambda, \tau, \mu, \xi^+, \xi^- \geq 0
\end{aligned} \tag{23}$$

The key insight we highlight here is that *the dual problem can also be written in the form of a deep network*, which provides a trivial way to find feasible solutions to the dual problem, which can then be optimized over. Specifically, consider the constraints

$$\begin{aligned}
(u_{i,j} - \ell_{i,j})\lambda_{i,j} - \mu_{i,j} - \tau_{i,j} &= (W_i^T \nu_{i+1})_j \\
\nu_{i,j} &= u_{i,j}\lambda_{i,j} - \mu_i.
\end{aligned} \tag{24}$$

Note that the dual variable λ corresponds to the upper bounds in the convex ReLU relaxation, while μ and τ correspond to the lower bounds $z \geq 0$ and $z \geq \hat{z}$ respectively; by the complementarity property, we know that at the optimal solution, these variables will be zero if the ReLU constraint is non-tight, or non-zero if the ReLU constraint is tight. Because we cannot have the upper and lower bounds be simultaneously tight (this would imply that the ReLU input $\hat{z}$ would exceed its upper or lower bound otherwise), we know that either λ or $\mu + \tau$ must be zero. This means that at the optimal solution to the dual problem

$$\begin{aligned}(u_{i,j} - \ell_{i,j})\lambda_{i,j} &= [(W_i^T \nu_{i+1})_j]_+ \\ \tau_{i,j} + \mu_{i,j} &= [(W_i^T \nu_{i+1})_j]_-\end{aligned} \tag{25}$$

i.e., the dual variables capture the positive and negative portions of $(W_i^T \nu_{i+1})_j$ respectively. Combining this with the constraint that

$$\nu_{i,j} = u_{i,j}\lambda_{i,j} - \mu_i \tag{26}$$

means that

$$\nu_{i,j} = \frac{u_{i,j}}{u_{i,j} - \ell_{i,j}}[(W_i^T \nu_{i+1})_j]_+ - \alpha[(W_i^T \nu_{i+1})_j]_-, \quad \text{for } j \in \mathcal{I}_i \tag{27}$$

for some $0 \leq \alpha \leq 1$ (this accounts for the fact that we can either put the "weight" of $[(W_i^T \nu_{i+1})_j]_-$ into μ or τ, which will or will not be passed to the next ν_i). This is exactly a type of leaky ReLU operation, with a slope in the positive portion of $u_{i,j}/(u_{i,j} - \ell_{i,j})$ (a term between 0 and 1), and a negative slope anywhere between 0 and 1. Similarly, and more simply, note that ξ^+ and ξ^- simply denote the positive and negative portions of $W_1^T \nu_2$, so we can replace these terms with an absolute value in the objective. Finally, we note that although it is possible to have $\mu_{i,j} > 0$ and $\tau_{i,j} > 0$ simultaneously, this corresponds to an activation that is identically zero pre-ReLU (both constraints being tight), and so is expected to be relatively rare. Putting this all together, and using $\hat{\nu}$ to denote "pre-activation" variables in the dual network, we can write the dual problem in terms of the network

$$\begin{aligned}\nu_k &= -c \\ \hat{\nu}_i &= W_i^T \nu_{i+1}, i = k-1, \ldots, 1 \\ \nu_{i,j} &= \begin{cases} 0 & j \in \mathcal{I}_i^- \\ \hat{\nu}_{i,j} & j \in \mathcal{I}_i^+ \\ \frac{u_{i,j}}{u_{i,j} - \ell_{i,j}}[\hat{\nu}_{i,j}]_+ - \alpha_{i,j}[\hat{\nu}_{i,j}]_- & j \in \mathcal{I}_i \end{cases}\end{aligned} \tag{28}$$

which we will abbreviate as $\nu = g_{\theta,x}(c, \alpha)$ to emphasize the fact that $-c$ acts as the "input" to the network and α are per-layer inputs we can also specify (for only those activations in $\mathcal{I}_i$), where ν in this case is shorthand for all the ν_i and $\hat{\nu}_i$ activations. It is worth noting that this is exactly the backpropagation for our original classifier (the ν variables are the backprop gradients, and the $\nu_{i,j}$ terms are set to be either zero or $\hat{\nu}_{i,j}$ if we are on the zero or linear portions of the ReLU respectively) except that the pass for the $\mathcal{I}_i$ variables has some additional free parameters: positive terms $\hat{\nu}_{i,j}$ are weighted by $\frac{u_{i,j}}{u_{i,j} - \ell_{i,j}}$ and for negative terms we have the freedom to choose a weighting between zero and one via the free parameter α.

The final objective we are seeking to optimize can also be written

$$\begin{aligned}J(\nu) &= -\sum_{i=1}^{k-1} \nu_{i+1}^T b_i - (x + \epsilon)^T[\hat{\nu}_1]_+ + (x - \epsilon)^T[\hat{\nu}_1]_- + \sum_{i=2}^{k-1}\sum_{j \in \mathcal{I}_i} \frac{u_{i,j}\ell_{i,j}}{u_{i,j} - \ell_{i,j}}[\hat{\nu}_{i,j}]_+ \\ &= -\sum_{i=1}^{k-1} \nu_{i+1}^T b_i - x^T \hat{\nu}_1 - \epsilon\|\hat{\nu}_1\|_1 + \sum_{i=2}^{k-1}\sum_{j \in \mathcal{I}_i} \ell_{i,j}[\nu_{i,j}]_+\end{aligned} \tag{29}$$

Thus our final rewritten form of the dual problem is

$$\text{maximize} \quad J(g(c, \alpha)) \tag{30}$$

where we optimize over the variables α, and where we can solve the problem using any standard deep learning optimization method such as gradient descent (projected gradient descent, as there are the constraints that $0 \leq \alpha_{i,j} \leq 1$). Furthermore, the above consideration that it is unlikely for $\tau_{i,j}$ and $\mu_{i,j}$ to simultaneously be 0, means that we expect α to attain values of either 0 or 1. This means we can even apply methods like the fast gradient sign to simply observe the gradient w.r.t. $\alpha_{i,j}$ at some intermediate value (e.g. 0.5), and then move it to either 0 or 1 depending on the sign of its gradient, in order to provide a quick bound with just a single optimization iteration.

A crucial point here is that this formulation of the problem is *not* convex in α, and so in general we have no guarantee of attaining optimal solutions. However, they key point is that *because the dual network always corresponds to feasible solutions of the dual problem, any solution will be a guaranteed lower bound on the optimal objective of the primal*. Thus, as long as the optimization procedure works well in practice (which we know to be typically the case for deep network optimization), we generally expect good performance from the method, i.e., that it will produce good bounds on the optimal solution of the primal.

C.3 Determining activation bounds

We have thus far ignored the question of how to actually find bounds ℓ_i and u_i on the pre-ReLU activations in the network. The discussion above, however, motivates a simple approach to doing so. We can simply build these bounds iteratively, layer by layer, using precisely the optimization problem formulated above. For instance, suppose that we have determined bounds for layers $2, \ldots, n$; then we can solve the above optimization problem with $c = e_i$ or $c = -e_i$ to find lower and upper bounds respectively for the ith activation in the $n + 1$ layer.[3] Note again that because we solve this optimization problem via the dual, these bounds are guaranteed to be strict, even if we don't solve the dual problem to optimality.

Naturally, solving two optimization problems per activation in the network (per example) would be a daunting task, and in this section we present alternative approaches that still give guaranteed bounds but which are much more efficient than this "brute force" approach. However, we fully admit that currently this step is by far the least scalable element of our overall approach, and finding the proper approximation schemes to scale to e.g. ImageNet-sized systems remains an open challenge.

The basis of our approximation approach here is precisely the fact we mentioned before: that even suboptimal solutions of the dual optimization problem are guaranteed to give valid bounds. Thus, instead of optimizing over α for each activation (and each example), we use a specific fixed feasible solution for α that substantially simplifies the problem. In particular, choosing

$$\alpha_{i,j} = \frac{u_{i,j}}{u_{i,j} - \ell_{i,j}} \tag{31}$$

guarantees that the "slope" of the leaky ReLU for those elements $j \in \mathcal{I}_i$ is the same in the negative portion as the positive portion, i.e., the operation is simply linear (note that $0 < \alpha_{i,j} < 1$ since $\ell_{i,j}$ is negative). Then the value of ν_i for all activations simultaneously (though still for a single example, since ℓ and u are sample-dependent), is given by

$$\begin{aligned} \hat{\nu}_i &= W_i^T D_{i+1} W_{i+1}^T \ldots D_n W_n^T \\ \nu_i &= D_i \hat{\nu}_i \end{aligned} \tag{32}$$

where again, we suppose that we have thus far already computed bounds up until layer n, and where D_i is a diagonal matrix with

$$(D_i)_{jj} = \begin{cases} 0 & j \in \mathcal{I}_i^- \\ 1 & j \in \mathcal{I}_i^+ \\ \frac{u_{i,j}}{u_{i,j} - \ell_{i,j}} & j \in \mathcal{I}_i \end{cases} . \tag{33}$$

Now we consider computing the objective value $J(\nu)$ for this particular setting of ν. The two linear terms in the objective

$$x^T \hat{\nu}_1 \text{ and } \sum_{i=1}^{k-1} b_i^T \nu_{i+1} \tag{34}$$

[3]Although we have discussed using the procedure to generate lower bounds, it can also trivially produce upper bound on $\max c^T \hat{z}_k$ by minimizing $\min -c^T \hat{z}_k$; if J is a lower bound on the minimization problem, then $-J$ will be an upper bound on the maximization problem.

are both simple to handle in the above case: we simply perform the multiplication with ν_i left-to-right, which corresponds simply to feeding the initial example and the bias terms through the network (without bias terms).

The $\epsilon\|\nu\|_1$ term in the objective is harder to deal with; because this is a non-linear term, in general there is nothing to do except form the entire ν_1 matrix explicitly, then compute the ℓ_1 norm of each column. This still may not be entirely intractable: we can perform the matrix multiplication (32) in any order we want, so if the input to the network is substantially smaller than the number of activations at a particular layer, or if there is some "bottleneck" layer of limited dimension, such as a PCA pre-processing of the data, then computing the full ν_1 matrix may be reasonable. Alternatively, when the linear operators are highly structured (such as convolutions), there will be a large amount of sparsity that can be exploited, at least for networks that are not too deep; we could also use random projection methods [Candes and Tao, 2006] to derive high-probability bounds on the norms in question; or we can even use a optimization formulation to in turn bound *this* quantity via convex duality. However, we leave a thorough investigation of these approaches for future work, and focus here are cases where we simply compute the matrix exactly.

The same considerations hold true for the $\ell_{i,j}[\nu_{i,j}]_+$ terms in the objective: they are non-linear so cannot easily be simplified. However, in this case we only need to compute these terms for i, j such that $j \in \mathcal{I}_j$, which is hopefully a manageably small subset of the total number of activations (if it is not, the convex outer adversarial polytope is likely to be extremely large anyway).

With these considerations in mind, we propose a layer-by-layer method for generating all the ℓ_i and u_i bounds: we start by generating lower and upper bounds for z_2 by the above procedure (which actually reduces to just a simple norm bound in this case), use these bounds to generate upper and lower bounds for z_3, etc. Several terms can be reused through this computation, and we highlight a reasonably efficient method for doing so (though still explicitly maintaining $\hat{\nu}_1$ and the various $\nu_{i,j}$ terms for $j \in \mathcal{I}_i$) in Algorithm 1.

It is worth highlighting the connection between this incremental approach and standard norm bounds. Note that for the first layer in the network, all terms effectively lie in the "linear" set $\mathcal{I}_1^+$. Thus we have $\hat{\nu}_1 = -W_1^T$, and are upper and lower bounds $J(\nu)$ reduce to

$$\begin{aligned} \ell_2 &= x^T W_1^T + b_1^T - \epsilon \|W_1^T\|_1 \\ u_2 &= x^T W_1^T + b_1^T + \epsilon \|W_1^T\|_1 \end{aligned} \tag{35}$$

where the first term is just the example x fed through the first layer of the network and the $\epsilon\|W_1^T\|_1$ (where for a matrix this denotes the column-wise ℓ_1 norm) is a standard norm bound on how far a bounded ℓ_∞ perturbation can reach. For later layers, the first two terms are replaced by

$$x^T \hat{\nu}_1 + \sum_{i=1}^{k-1} b_i^T \nu_{i+1} \tag{36}$$

which again just denotes the example and bias terms fed through the network, whereas the later term is replaced by

$$\epsilon\|\hat{\nu}_1\|_1 = \epsilon\|W_1^T D_2 W_2^T \ldots D_n W_n^T\|_1 \tag{37}$$

which is just the ℓ_1 norms of the actual product of weights and activation terms. This bound would be sufficient if all activations lay in $\mathcal{I}^-$ and $\mathcal{I}^+$, i.e. if the network could only be perturbed within a linear region. But obviously in reality the perturbation can cause some activations to switch between positive and negative (exactly the $\mathcal{I}_i$ set) and in these cases we pay an additional "cost" in our bounds of $\ell_{i,j}[\nu_{i,j}]_+$. Remember that $\nu_{i,j}$ positive at the optimal solution corresponds to activations that "cheat" by lying on the upper portion of the convex ReLU relaxation, and the bound nicely capture the fact that it will be looser in this case.

As a final note, we highlight the fact that the incremental computation of all these bounds, followed by some number of optimization steps for the final dual problem, can all be implemented within an automatic differentiation library (this entire procedure really is what defines our actual outer bound on the adversarial polytype). This means that the final lower bound on $c^T \hat{z}_k$ can be differentiated with respect to the parameters of the network itself. Thus, in learning our classifier we will explicitly minimize over the entire computation of this bound.

Algorithm 1 Compute network bounds

```
input: Network parameters $\{W_i, b_i\}_{i=1}^{k-1}$, data point $x$, ball size $\epsilon$
// initialization
    $\hat{\nu}_1 := -W_1^T$
    $\gamma_1 := -b_1^T$
    $\ell_2 := x^T W_1^T + b_1^T - \epsilon \|W_1^T\|_1$ // $\|\cdot\|_1$ for a matrix here denotes $\ell_1$ norm of all columns
    $u_2 := x^T W_1^T + b_1^T + \epsilon \|W_1^T\|_1$
for $i = 2, \ldots, k-1$ do
    form $\mathcal{I}_i^-, \mathcal{I}_i^+, \mathcal{I}_i$; form $D_i$ as in (33)
    // initialize new terms
        $\nu_{i,\mathcal{I}_i} := (D_i)_{\mathcal{I}_i} W_i^T$
        $\gamma_i := -b_i^T$
    // propagate existing terms
        $\nu_{j,\mathcal{I}_j} := \nu_{j,\mathcal{I}_j} D_i W_i^T, \;\; j = 2, \ldots, i-1$
        $\gamma_j := \gamma_j D_i W_i^T, \;\; j = 1, \ldots, i-1$
        $\hat{\nu}_1 := \hat{\nu}_1 D_i W_i^T$
    // compute bounds
        $\ell_i := x^T \hat{\nu}_1 + \sum_{j=1}^{i} \gamma_i - \epsilon \|\hat{\nu}_1\|_1 + \sum_{j=2}^{i} \ell_i [\nu_{i,\mathcal{I}_i}]_+$
        $u_i := x^T \hat{\nu}_1 + \sum_{j=1}^{i} \gamma_i + \epsilon \|\hat{\nu}_1\|_1 - \sum_{j=2}^{i} \ell_i [-\nu_{i,\mathcal{I}_i}]_+$
end for
output: bounds $\{\ell_i, u_i\}_{i=2}^{k-1}$
```

C.4 Integration with robust optimization

Finally, we highlight how the above approach can be easily integrated within the robust classifier learning objective we highlighted before. Recall that our robust optimization problem was given by

$$\underset{\theta}{\text{minimize}} \quad \sum_{i=1}^{N} [1 + \max_{y \neq y_i} \max_{\hat{z} \in \mathcal{Z}_\epsilon}(x_i) \left((\hat{z}_k)_y - (\hat{z}_k)_{y_i}\right)]_+. \tag{38}$$

Using the techniques from this section, we can bound this term as

$$\begin{aligned} \min_{\theta} \sum_{i=1}^{N} [1 + \max_{y \neq y_i} \max_{\hat{z} \in \mathcal{Z}_\epsilon}(x_i) \left((\hat{z}_k)_y - (\hat{z}_k)_{y_i}\right)]_+ &\leq \min_{\theta} \sum_{i=1}^{N} [1 + \max_{y \neq y_i} \max_{\hat{z} \in \tilde{\mathcal{Z}}_\epsilon(x_i)} \left((\hat{z}_k)_y - (\hat{z}_k)_{y_i}\right)]_+ \\ &\leq \min_{\theta, \alpha_{1:N}} \sum_{i=1}^{N} [1 - \max_{y \neq y_i} J(g_{\theta, x_i}(e_y - e_{y_i}, \alpha_i))]_+ \end{aligned} \tag{39}$$

i.e., we can substitute the cost function from our dual network into the loss of our original optimization problem, then solve simultaneously over θ and α_i. Note that in this formulation, computing our dual network $\nu = g_{\theta, x_i, \epsilon}(c, \alpha_i)$ implicitly also means computing the sequential bounds as described in the previous section, and we use the subscripts x_i, ϵ on g to emphasize the fact that these bounds (and hence the dual network) depends on the actual example and ϵ. In practice, during training we often forgo optimization over α and simply use the solution mentioned in the previous section to the compute the upper and lower bounds when computing the final bound as well.

Lastly, although the hinge loss is useful for illustrating the connection between the loss function and the optimization problem, in practice it is less effective than more common losses such as cross entropy. Fortunately, for any convex monotonic loss function (which includes cross entropy), we can use the bound

$$\max_{\|\Delta\|_1 \leq \epsilon} \ell(f_\theta(x_i + \Delta), y_i) \leq \ell(-J(g_{\theta, x_i}(I - e_{y_i} 1^T, \alpha_i)), y_i) \tag{40}$$

to use more general loss functions, where $-J(g_{\theta,x}(I - e_{y_i} 1^T, \alpha_i))$ (i.e., the dual bound evaluated at $c = e_y - e_{y_i}$ for all y) takes the place of the activations.

C.5 Summary and discussion

The presentation in this section was quite lengthy, so it is useful to provide a summary of the end result. The final outcome of our approach is that if we learn our network by solving the optimization problem

$$\underset{\theta,\alpha}{\text{minimize}} \quad \sum_{i=1}^{N} \ell(-J(g_{\theta,x_i,\epsilon}(I - e_{y_i}1^T, \alpha_i)), y_i) \tag{41}$$

where g_{θ,x_i} is a network we compute using a few passes through the original network and its backpropagation network (albeit with much larger "batch" sizes through the network, since in the worst case we need to feed in a sample for each dimension of the input to compute the ℓ and u bounds), then this provides a *guaranteed* bound on the adversarial loss we can suffer for *any* adversarial perturbation with ℓ_∞ norm bounded by ϵ. The method also produces a guaranteed bound on the adversarial error, by simply checking whether the y_i coordinate of $-J(g_{\theta,x_i,\epsilon}(I - e_{y_i}1^T, \alpha_i))$ is the largest (by the scaling with the input $c = I - e_{y_i}1^T$, this coordinate is always zero, so it amounts to checking whether all other coordinates are negative).

Although the method is based upon convex duality and linear programming, no recourse to an actual linear programming solver is needed, nor is any iterative method for solving any inner optimization problems; we simply optimize the entire (nonconvex) objective with stochastic gradient descent or any other of its variants used in deep learning, and we get the guaranteed bound.

Finally, we note that all the bounds and guarantees we have discussed so far apply to the training data (they can be evaluated on a test set, but like all test error, only if the test labels are known). What are we to do with such networks in deployment, where we see new examples without knowledge of the actual label? Fortunately, by the transitivity of adversarial examples, we can use the same technique to determine whether the example *might* be adversarial. Specifically, given a new example x we compute the network prediction $\hat{y} = f(x)$, then determine whether there is any potential adversarial example for this predicted label within ϵ. If not, then the example must *not* be adversarial, because there is no point within ϵ that changes the class prediction (i.e., there couldn't be a "normal" input ϵ away from this possibly adversarial example). Obviously, this approach may sometimes classify non-adversarial inputs as potentially adversarial, but it has zero false negatives, in that it will never fail to flag an adversarial example. Given the challenge in even defining adversarial examples in general, this seems to be as strong a guarantee as is currently possible.

D Experimental results

Here we present experiments on small-scale problems designed to demonstrate the approach. Although the method does not yet scale to ImageNet-sized classifiers, we do demonstrate the approach on a simple convolutional network applied to MNIST, illustrating that the method can apply to approaches beyond fully-connected networks with very small sizes (which represent the state of the art for most existing work on neural network verification). Scaling challenges were discussed briefly above, and we highlight them more below. As mentioned above, code for the examples included in the paper is available at `http://github.com/locuslab/convex_adversarial`.

D.1 2D toy domain

Our first set of experiments involves a simple domain where both the input and output spaces of the network are two dimensional, so can be easily visualized. Specifically, we consider a 2-100-100-100-100-2 fully connected network.

Visualizations of the convex outer adversarial polytope To begin, we consider some simple cases of visualizing the outer approximation to the adversarial polytope for random networks. Because the output space is two-dimensional we can easily visualize the polytopes in the output layer, and because the input space is two dimensional, we can easily cover the entire input space densely to enumerate the true adversarial polytope. In this experiment, we initialized the weights of the all layers to be normal $\mathcal{N}(0, 1/\sqrt{n_{\text{in}}})$ and biases normal $\mathcal{N}(0, 1)$ (due to scaling, the actual absolute value of weights is not particularly important except as it relates to ϵ). Although obviously not too much should be read into these experiments with random networks, the main takeaways are that 1) for "small" ϵ, the outer bound is an extremely good approximation to the adversarial polytope; 2) as ϵ increases, the

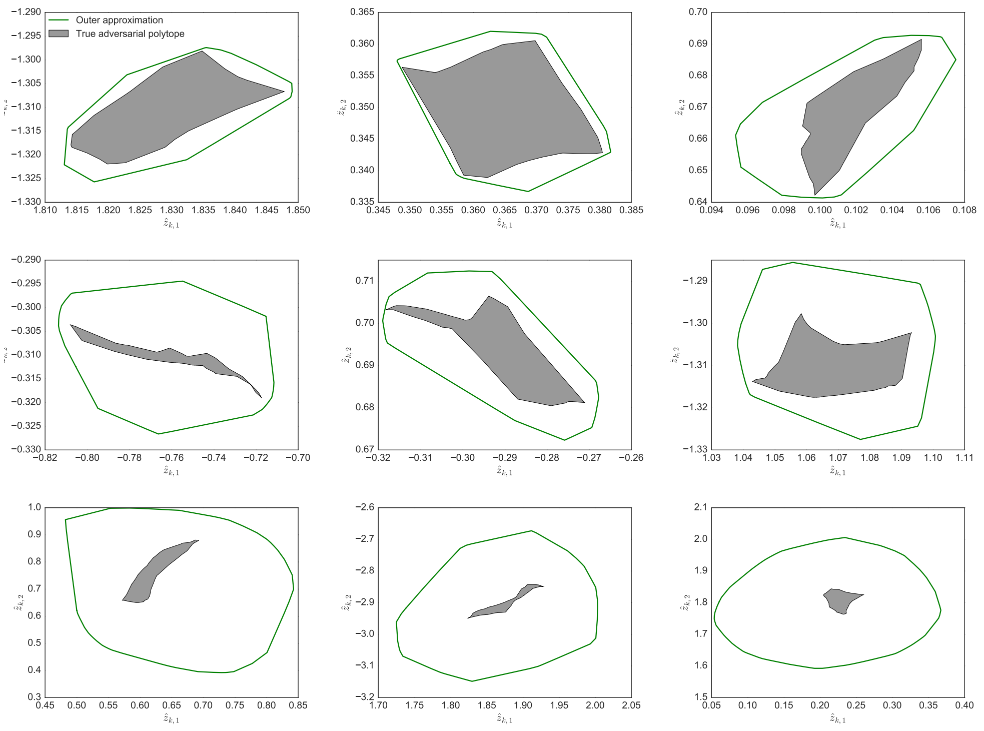

Figure 6: Illustrations of the true adversarial polytope (gray) and our convex outer approximation (green) for a random 2-100-100-100-100-2 network with $\mathcal{N}(0, 1/\sqrt{n})$ weight initialization. Polytopes are shown for $\epsilon = 0.05$ (top row), $\epsilon = 0.1$ (middle row), and $\epsilon = 0.25$ (bottom row).

bound gets substantially weaker. This is to be expected: for small ϵ, the number of elements in $\mathcal{I}$ will also be relatively small, and thus additional terms that make the bound lose are expected to be relatively small (in the extreme, when no activation can change, the bound will be exact, and the adversarial polytope will be a convex set). However, as ϵ gets larger, more activations enter the set $\mathcal{I}$, and the available freedom in the convex relaxation of each ReLU increases substantially, making the bound looser. Naturally, the question of interest is how tight this bound is for networks that are actually trained to minimize the robust loss, which we will look at shortly.

Comparison to naive layerwise bounds One additional point is worth making in regards to the bounds we propose. It would also be possible to achieve a naive "layerwise" bound by iteratively determining absolute allowable ranges for each activation in a network (via the simple norm bound mentioned above), then for future layers, assuming each activation can vary arbitrarily within this range. This provides a simple iterative formula for computing layer-by-layer absolute bounds on the coefficients, and similar techniques have been used e.g. in Parseval Networks [Cisse et al., 2017] to produce more robust classifiers (albeit there considering ℓ_2 perturbations instead of ℓ_∞ perturbations, which likely are better suited for such an approach). Unfortunately, these naive bounds are extremely loose for multi-layer networks (in the first hidden layer, they naturally match our bounds exactly). For instance, for the adversarial polytope shown in Figure 6 (top left), the actual adversarial polytope is contained within the range

$$\hat{z}_{k,1} \in [1.81, 1.85], \quad \hat{z}_{k,2} \in [-1.33, -1.29] \tag{42}$$

with the convex outer approximation mirroring it rather closely. In contrast, the layerwise bounds produce the bound:

$$\hat{z}_{k,1} \in [-11.68, 13.47], \quad \hat{z}_{k,2} \in [-16.36, 11.48]. \tag{43}$$

Such bounds are essentially vacuous in our case, which makes sense intuitively. The naive bound has no way to exploit the "tightness" of activations that lie entirely in the positive space, and effectively

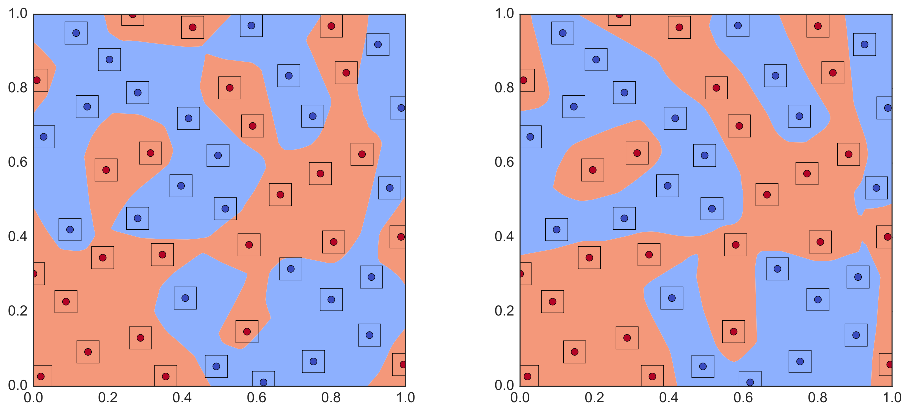
Figure 7: Illustration of classification boundaries resulting from standard training (left) and robust training (right) with ℓ_∞ balls of size $\epsilon = 0.03$ (shown in figure). The standard training procedure allows for some points within ball to have incorrect class labels, while the robust training does not (and the training procedure provides a bound verifying this fact).

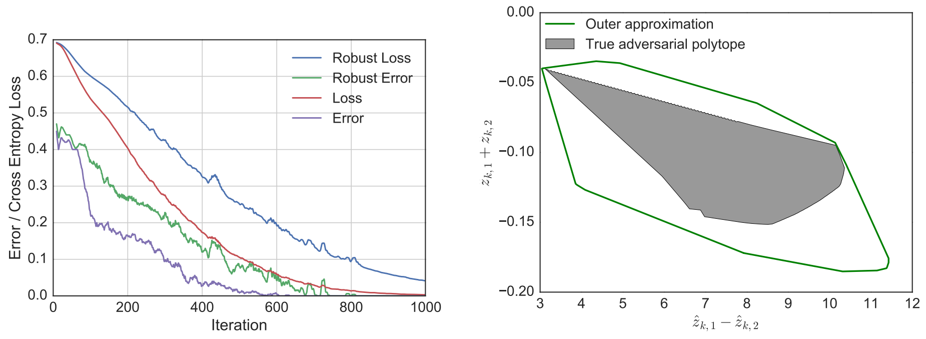

Figure 8: (left) Training curves for the 2D robust adversarial problem, where each iteration denotes a step of the Adam optimizer. (right) Illustration of the actual adversarial polytope and the convex outer approximation for one of the training points after the robust optimization procedure.

replaces the convex ReLU approximation with a (larger) box covering the entire space. Thus, such bounds are not of particular use when considering robust classification.

Robust classifier training Finally, we consider training a simple robust classifier in our 2D example. Specifically, we incrementally randomly sample 50 points within the $[0, 1]$ xy-plane, at each point waiting until we find a sample that is at least 0.08 away from other points via ℓ_∞ distance, and assign each point a random label. We then attempt to learn a robust classifier that will correctly classify all points with an ℓ_∞ ball of $\epsilon = 0.03$. Note that there is no notion of generalization here, we are just evaluating the ability of the learning approach to fit a classification function robustly.

Figure 7 shows the resulting classifiers produced by standard training over just the data points themselves (left) and robust training via our method (right). As expected, the standard training approach results in points that are classified differently somewhere within their ℓ_∞ ball of radius $\epsilon = 0.03$ (this is exactly and adversarial example for the training set). In contrast, also as expected (because our procedure is able to attain zero robust error), the robust training method provides a classifier that is guaranteed to classify all points within the balls correctly. We use the Adam optimizer [Kingma and Ba, 2015] (over the entire batch of samples) with a learning rate of 0.001.

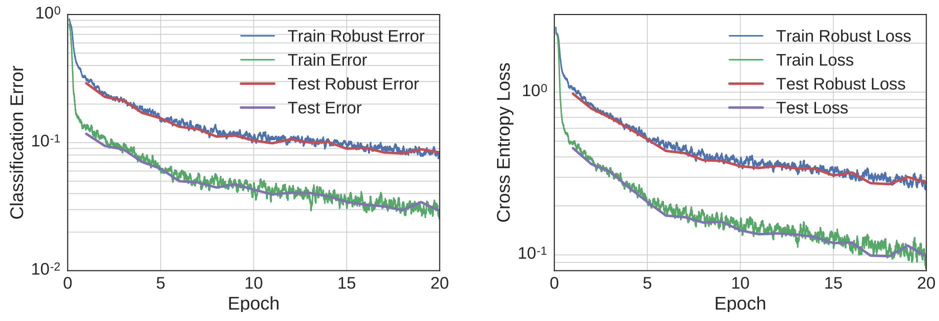

Figure 9: Training progress for our method applied to a simple ConvNet architecture on MNIST. Robust error and loss indicate provable upper bounds on the loss/error achievable by any adversarial perturbation with ℓ_∞ norm bounded by $\epsilon = 0.1$.

Figure 8 (left) shows the training progress of the standard and robust learning processes on this example. Of note is the fact that the robust loss seems empirically "harder" to minimize, which is not particularly surprising since it effectively always looks for (a bound on) the worse case loss within the entire ℓ_∞ ball around the example. However, the network is able to eventually achieve zero robust error, which is expected given the fact that the data points can indeed be separated perfectly with a nonlinear classifier. Finally, it is of some interest to see what the true adversarial polytope for the examples in this data set looks like versus the convex approximation, evaluated at the solution of the robust optimization problem. Figure 8 (right) shows one of these figures, highlighting the fact that for the final network weights and choice of epsilon, the outer bound is empirically quite tight in this case.

D.2 MNIST ConvNet

Finally, we present results on producing a provably robust classifier on the MNIST data set. Specifically, we consider a ConvNet architecture that includes two convolutional layers, with 16 and 32 channels (each with a stride of two, to decrease the resolution by half without requiring max pooling layers), and two fully connected layers stepping down to 100 and then 10 (the output dimension) hidden units, with ReLUs following each layer except the last. That is, the network has the form:

$$x \Rightarrow \text{Conv\_16x4x4,s=2} \Rightarrow \text{ReLU} \Rightarrow \text{Conv\_16x4x4,s=2} \Rightarrow \text{ReLU} \Rightarrow \text{FC100} \Rightarrow \text{ReLU} \Rightarrow y. \quad (44)$$

This is a reasonable if fairly small network for MNIST: training the network using standard training achieves a test error of about 1.2%. Adding layers like batch normalization and dropout easily push the error below 1%, but since we don't integrate these yet into our robust framework, we consider just the naive network. We use the Adam optimizer [Kingma and Ba, 2015] with a learning rate of 0.001 (the default option) with no additional hyperparameter selection.

Figure 9 shows the training progress using our procedure with a robust softmax loss function and $\epsilon = 0.1$. The "robust error" and "robust loss" here are our bounds on the robust error and loss; that is, we know that any norm-bounded adversarial technique will not be able to achieve loss or error that is any higher, though in practice it could be substantially lower as well. The final classifier after 20 epochs reaches a test error of 2.93% with a robust test error of 8.39%.

To see where other techniques would stand, we also tested our network against two common classes of attacks: the fast gradient sign method Goodfellow et al. [2015], and the projected gradient descent approach Madry et al. [2017].[4] For a traditionally-trained classifier (with 1.2% test error) the FGSM approach results in 39.7% error, while PGD results in 94.0% error. On the classifier trained with our method, however, we only achieve 2.93% test error, but FGSM and PGD only achieve errors of 5.8% and 6.2% respectively (both, naturally, below our bound of 8.4%). These results are summarized in Table 2.

While this is by no means state of the art performance on standard MNIST (if such a thing really has any meaning at this point, but it is certainly valid to say that these results are decidedly *subpar* as far

[4]For PGD, as in Madry et al. [2017] we use ℓ_∞ ball gradient descent and 100 iterations of step size 0.01, which was sufficient for convergence.

Network	**Non-adversarial**	**FGSM**	**PGD**	**Robust Bound**
Standard training	*1.2%*	39.7%	94.0%	100%
Robust training	2.9%	*5.8%*	*6.2%*	*8.4%*

Table 2: Adversarial errors for a traditionally-trained and our robustly-trained classifier on the MNIST test set.

as generic MNIST classification goes), a few points stand out. First, this is by far the largest provably verified network we are currently aware of, and 8% error represents a reasonable performance given that it is against *any* adversarial attack strategy; the only other such bound we are aware of, from the very recent work mentioned previously Anonymous [2018], provides a guaranteed bound of 35% error. Second, in the example above training and testing error/loss are still tracking quite closely, suggesting that we should be able to further improve performance by simply using larger models. Therein lies the catch, though: the current model took 10 hours to train for 20 epochs on a Titan X (Maxwell) GPU. This is between two and three orders of magnitude more costly than training the naive network (though the naive network, of course, is trivially susceptible to simple adversarial attacks like the PGD method). Building robust models within the framework to scale to e.g. ImageNet-sized image classification problems remains a challenging task. But because the approach is not combinatorial in its complexity, we believe it also represents a much more feasible approach than those based upon integer programming or satisfiability, which seem highly unlikely to ever scale to such problems. Thus, we believe the current performance represents a substantial step forward in research on adversarial examples in deep networks.