ADVERSARIAL TRAINING AND PROVABLE DEFENSES: BRIDGING THE GAP

Anonymous authors
Paper under double-blind review

ABSTRACT

We propose a new method to train neural networks based on a novel combination of adversarial training and provable defenses. The key idea is to model training as a procedure which includes *both*, the verifier and the adversary. In every iteration, the verifier aims to certify the network using convex relaxation while the adversary tries to find inputs inside that convex relaxation which cause verification to fail. We experimentally show that this training method is promising and achieves the best of both worlds – it produces a model with state-of-the-art accuracy (74.8%) and certified robustness (55.9%) on the challenging CIFAR-10 dataset with a 2/255 L_∞ perturbation. This is a significant improvement over the currently known best results of 68.3% accuracy and 53.9% certified robustness, achieved using a 5 times larger network than our work.

1 INTRODUCTION

The discovery of adversarial examples in deep learning (Szegedy et al., 2013) has increased the importance of creating new training methods which produce accurate and robust neural networks with provable guarantees.

Existing work: adversarial and provable defenses Adversarial training (Goodfellow et al., 2015; Kurakin et al., 2017) provides a basis framework which augments the training procedure with adversarial inputs produced by an adversarial attack. Madry et al. (2018) instantiated adversarial training using a strong iterative adversary and showed that their approach can train models which are highly robust against the strongest known adversarial attacks (Carlini & Wagner, 2017). This method has also been used to train robust ImageNet models (Xie et al., 2019). While promising, the main drawback of the method is that when instantiated in practice, via an approximation of an otherwise intractable optimization problem, it provides no guarantees – it does not produce a certificate that there are no possible adversarial attacks which could potentially break the model. To address this lack of guarantees, recent line of work on provable defenses (Wong & Kolter, 2018; Raghunathan et al., 2018; Mirman et al., 2018) has proposed to train neural networks which are certifiably robust under a specific attacker threat model. However, these guarantees come at the cost of a significantly lower standard accuracy than models trained using adversarial training. This setting raises a natural question – can we leverage ideas from both, adversarial training techniques and provable defense methods, so to obtain models with high accuracy and certified robustness?

This work: combining adversarial and provable defenses In this work, we take a step towards addressing this challenge. We show that it is possible to train more accurate and provably robust neural networks using the *same* convex relaxations as those used in existing, state-of-the-art provable defense methods, but with a new, different optimization procedure inspired by adversarial training. Our optimization works as follows: (i) to certify a property (e.g., robustness) of the network, the verifier produces a convex relaxation of all possible intermediate vector outputs in the neural network, then (ii) an adversary now searches over this (intermediate) convex regions in order to find, what we refer to as a *latent adversarial example* – a concrete intermediate input contained in the relaxation that when propagated through the network causes a misclassification that prevents verification, and finally (iii) the resulting latent adversarial examples are now incorporated into our training scheme using adversarial training. Overall, we can see this method as bridging the gap between adversarial training and provable defenses (it can conceptually be instantiated with any convex relaxation).

We experimentally show that the method is promising and results in a neural network with state-of-the-art 74.8% accuracy and 55.9% certified robustness on the challenging CIFAR-10 dataset with 2/255 L_∞ perturbation (the best known existing results are 68.3% accuracy and 53.9% certified robustness using 5 times larger network Wong et al. (2018)).

Main Contributions Our key contributions are:

- A new method we refer to as *layerwise adversarial training* which can train provably robust neural networks and conceptually bridges the gap between adversarial training and existing provable defense methods.
- Instantiation of layerwise adversarial training using linear convex relaxations used in prior work (accomplished by introducing a projection operator).
- Experimental results showing layerwise adversarial training can train neural network models which achieve both, state-of-the-art accuracy and certified robustness on CIFAR-10 with 2/255 L_∞ perturbation.

Overall, we believe the method presented in this work is a promising step towards training models that enjoy both, higher accuracy and higher certification guarantees. An interesting item for future work would be to explore instantiations of the method with other convex relaxations than the one considered here.

2 Related work

We now discuss some of the closely related work on robustness of neural networks.

Heuristic adversarial defenses After the first introduction of adversarial examples (Szegedy et al., 2013; Biggio et al., 2013), defense mechanisms to train robust neural networks were built based on the inclusion of adversarial examples to the training set (Kurakin et al., 2017; Goodfellow et al., 2015). Models trained using adversarial training with projected gradient descent (PGD) (Madry et al., 2018) were shown to be robust against the strongest known attacks (Carlini & Wagner, 2017). This is in contrast to other defense mechanisms which have been broken by new attack techniques (Athalye et al., 2018). While models trained using adversarial training achieve robustness against strong adversaries, there are no guarantees that model is robust against any kind of adversarial attack under the threat model considered.

Provable adversarial defenses Another line of work proposes to learn classifiers which come with robustness guarantees. These approaches are based on linear (Wong & Kolter, 2018) or semidefinite (Raghunathan et al., 2018; Dvijotham et al., 2018a) relaxations, hybrid zonotope (Mirman et al., 2018) or interval bound propagation (Gowal et al., 2018). While these approaches currently obtain robustness guarantees, accuracy of these networks is relatively small and limits practical use of these methods. There has also been recent work on certification of general neural networks, not necessarily trained in a special way. These methods are based on SMT solvers (Katz et al., 2017), mixed-integer linear programs (Tjeng et al., 2019), abstract interpretation (Gehr et al., 2018), restricted polyhedra (Singh et al., 2019b) or combinations of those (Singh et al., 2019a).

Another line of work proposes to replace neural networks with a randomized classifier (Lecuyer et al., 2018; Cohen et al., 2019; Salman et al., 2019a) which comes with probabilistic guarantees on its robustness. While these approaches scale to larger datasets such as ImageNet (although with probabilistc instead of exact guarantees), their bounds come from the relationship between L_2 robustness and Gaussian distribution. In this paper, we consider general verification problem (Qin et al., 2019) where input is not necessarily limited to an L_p ball, but arbitrary convex set.

3 Background

In this work we consider a threat model where an adversary is allowed to transform an input $\boldsymbol{x} \in \mathbb{R}^{d_0}$ into any point from a convex set $\mathbb{S}_0(\boldsymbol{x}) \subseteq \mathbb{R}^{d_0}$. For example, for a threat model based on L_∞ perturbations, the convex set will be defined as $\mathbb{S}_0(\boldsymbol{x}) = \{\boldsymbol{x}' \in \mathbb{R}^{d_0}, ||\boldsymbol{x} - \boldsymbol{x}'||_\infty < \epsilon\}$.

A neural network consisting of k layers and parameters θ is represented as a function h_θ where $h_\theta = h_\theta^k \circ h_\theta^{k-1} \cdots \circ h_\theta^1$ and $h_\theta^i : \mathbb{R}^{d_{i-1}} \to \mathbb{R}^{d_i}$ denotes a transformation applied at hidden layer i. We also denote the function representing part of the neural network from layer i to the final layer k as $h_\theta^{i:k} = h_\theta^k \circ h_\theta^{k-1} \cdots \circ h_\theta^i$.

Our goal will be to prove a property on the output of the neural network, encoded via a linear constraint:

$$\boldsymbol{c}^T h_\theta(\boldsymbol{x}') + d < 0, \forall \boldsymbol{x}' \in \mathbb{S}_0(\boldsymbol{x}) \tag{1}$$

where $\boldsymbol{c}$ and d are property specific vector and scalar values. This formulation is general enough to capture many interesting safety properties (Dvijotham et al., 2018b; Qin et al., 2019), including robustness to L_p perturbations. The standard approach to train neural networks which satisfy this constraint is to define a surrogate loss $\mathcal{L}$ and solve the following min-max optimization problem:

$$\min_\theta \mathbb{E}_{(\mathrm{x},\mathrm{y})\sim D} \max_{\boldsymbol{x}' \in \mathbb{S}_0(\boldsymbol{x})} \mathcal{L}(h_\theta(\boldsymbol{x}'), y) \tag{2}$$

Because inner maximization is intractable, most existing approaches replace it with an approximation. Depending on the used approximation, we distinguish two families of techniques.

Adversarial training One family of methods, referred to as adversarial training (Goodfellow et al., 2015; Kurakin et al., 2017), replaces the maximum loss with a lower bound which is obtained using an adversarial attack. Madry et al. (2018) maximized the inner loss using a projected gradient descent (PGD) attack and found that, perhaps surprisingly, this optimization procedure succeeds in training deep architectures which are robust against the strongest known adversaries. While this provides strong empirical evidence that the resulting models are indeed robust, the approach offers no guarantees. Thus, it remains unclear whether there exist even stronger attacks that could break a model trained in this manner.

Provable defenses A second family of methods to train certified neural networks is based on the computation of an upper bound to the inner loss, as opposed to a lower bound computed for adversarial training. These methods are typically referred to as provable defenses as they provide guarantees on the robustness of the resulting network, under any kind of attack inside the threat model. An upper bound is typically computed using linear relaxations (Wong & Kolter, 2018), intervals (Gowal et al., 2018) or methods combining interval bounds and linear relaxations (Mirman et al., 2018; Zhang et al., 2019). However, these methods suffer from two disadvantages. First, due to the convex relaxations, an upper bound on the loss is typically not tight and can be quite loose. However, we believe this is less of an issue due to the fact that interval relaxations were shown to experimentally be able to train more provably robust models than methods based on linear relaxations (which usually produce tighter bounds than intervals). For example, Mirman et al. (2018); Zhang et al. (2019) report $\sim 28\%$ robust error using pure interval training on CIFAR-10 with perturbation 8/255 while Wong et al. (2018) achieve 21% using linear relaxations. Second, the way these methods construct the loss makes the relationship between the loss and the network parameters significantly more complex than in standard training. This causes the resulting optimization problem to be more difficult, meaning these training methods often converge to a suboptimal solution. Our experimental results confirm this – we substantially outperform existing methods both in terms of accuracy and certified robustness using the *same* linear relaxation, but a different optimization procedure.

Certification via convex relaxations We now formally describe how provable defenses perform certification. We denote the set of possible intermediate concrete vectors at layer i that can be obtained by propagating vector $\boldsymbol{x}' \in \mathbb{S}_0(\boldsymbol{x})$ through the network as $\mathbb{S}_i(\boldsymbol{x}) = h_\theta^i(\mathbb{S}_{i-1}(\boldsymbol{x})) \subseteq \mathbb{R}^{d_i}$. As it is difficult to explicitly compute the set $\mathbb{S}_i(\boldsymbol{x})$, a standard approach is to approximate it via a convex relaxation $\mathbb{C}_i(\boldsymbol{x})$. As the input set is already convex, there is no need to introduce a relaxation, and thus we set $\mathbb{C}_0(\boldsymbol{x}) = \mathbb{S}_0(\boldsymbol{x})$. Given a neural network layer h_θ^i which transforms one set of vectors into another, we represent its corresponding convex relaxation transformer as g_θ^i. That is, g_θ^i will transform one convex set into another convex set. More formally, for any set $\mathbb{D} \subseteq \mathbb{R}^{d_{i-1}}$, $g_\theta^i(\mathbb{D})$ is convex and $h_\theta^i(\mathbb{D}) \subseteq g_\theta^i(\mathbb{D})$. Then, we recursively define the effect of g_θ^i on a convex relaxation as $\mathbb{C}_i(\boldsymbol{x}) = g_\theta^i(\mathbb{C}_{i-1}(\boldsymbol{x})) \subseteq \mathbb{R}^{d_i}$. Finally, to certify robustness using the obtained convex relaxation, it is enough to check whether all output vectors in $\mathbb{C}_k(\boldsymbol{x})$ satisfy the linear constraint in Equation 1. If this is true, then all output vectors in $\mathbb{S}_k(\boldsymbol{x})$ satisfy the constraint as well due to the fact that $\mathbb{S}_k(\boldsymbol{x}) \subseteq \mathbb{C}_k(\boldsymbol{x})$.

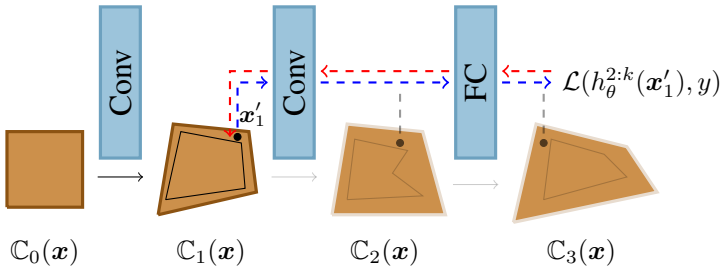

Figure 1: An iteration of layerwise adversarial training. Latent adversarial example x'_1 is found in the convex region $\mathbb{C}_1(\boldsymbol{x})$ and propagated through the rest of the layers in a forward pass which is shown with the blue line. During backward pass, gradients are propagated through the same layers, shown with the red line. Note that the first convolutional layer does not receive any gradients.

4 Provable Defense via Layerwise Adversarial Training

We now describe our layerwise adversarial training approach which yields a provable defense that bridges the gap between standard adversarial training and existing provable defenses.

Motivation: latent adversarial examples Consider an already trained neural network model h_θ which we would like to certify using convex relaxations. A fundamental issue here is that certification methods based on convex relaxations can struggle to prove the target property (e.g., robustness) due to the iterative accumulation of errors introduced by the relaxation. More precisely, assume the neural network actually satisfies the property from Equation 1 for an input $\boldsymbol{x}$, meaning that $\boldsymbol{c}^T h_\theta(\boldsymbol{x}') + d < 0, \forall \boldsymbol{x}' \in \mathbb{S}_0(\boldsymbol{x})$. Naturally, this also implies that the neural network behaves correctly in the latent space of its first hidden layer in the region $\mathbb{S}_1(\boldsymbol{x})$. Formally, this means that $\boldsymbol{c}^T h_\theta^{2:k}(\boldsymbol{x}'_1) + d < 0, \forall \boldsymbol{x}'_1 \in \mathbb{S}_1(\boldsymbol{x})$. However, if one would use a certification method which replaces the region $\mathbb{S}_1(\boldsymbol{x})$ by its convex relaxation $\mathbb{C}_1(\boldsymbol{x})$, then it is possible that we would fail to certify our desired property. This is due to the fact that there may exist an input $\boldsymbol{x}'_1 \in \mathbb{C}_1(\boldsymbol{x}) \setminus \mathbb{S}_1(\boldsymbol{x})$ such that $\boldsymbol{c}^T h_\theta^{2:k}(\boldsymbol{x}'_1) + d \geq 0$. Of course, we could repeat the above thought experiment and possibly find more violating latent inputs in the set $\mathbb{C}_i(\boldsymbol{x}) \setminus \mathbb{S}_i(\boldsymbol{x})$ of any hidden layer i. The existence of points found in the difference between a convex relaxation and the true region is a fundamental reason for the failure of certification methods based on convex approximations. For convenience, we refer to such points as *latent adversarial examples*. Next, we describe a method which trains the neural network in a way that aims to minimize the number of latent adversarial examples.

Layerwise provable optimization via convex relaxations Our key observation is that the two families of defense methods described earlier are in fact different ends of the same spectrum: methods based on *adversarial training* maximize the cross-entropy loss in the first convex region $\mathbb{C}_0(\boldsymbol{x})$ while *provable defenses* maximize the same loss, but in the last convex region $\mathbb{C}_k(\boldsymbol{x})$. Both methods then backpropagate the loss through the network and update the parameters using SGD. However, as explained previously, certification methods may fail even before the last layer due to the presence of latent adversarial examples in the difference of the regions $\mathbb{C}_i(\boldsymbol{x})$ and $\mathbb{S}_i(\boldsymbol{x})$. A natural question then is – can we leverage adversarial training so to eliminate latent adversarial examples from hidden layers and obtain a provable network?

To this end, we propose adversarial training in layerwise fashion. The initial phase of training is equivalent to adversarial training as used by Madry et al. (2018). In this phase in the inner loop we repeatedly find an input in $\mathbb{C}_0(\boldsymbol{x})$ which maximizes the cross-entropy loss and update the parameters of the neural network so to minimize this loss using SGD. Note that the outcome of this phase is a model which is highly robust against strong multi-step adversaries. However, certification of this fact often fails due to the previously mentioned accumulation of errors in the particular convex relaxation being used.

The next step of our training method is visually illustrated in Figure 1. Here, we propagate the initial convex region through the first layer of the network and obtain the convex relaxation $\mathbb{C}_1(\boldsymbol{x})$. We then solve the optimization problem to find a *concrete point* $\boldsymbol{x}'_1$ inside of $\mathbb{C}_1(\boldsymbol{x})$ which produces

Algorithm 1: Layerwise adversarial training via convex relaxations

Data: k-layer neural network h_θ, training set $(\mathcal{X}, \mathcal{Y})$, learning rate η, step size α, inner steps n
Result: Certifiably robust neural network h_θ

for $l \leq k$ **do**
  for $i \leq n_{epochs}$ **do**
    Sample mini-batch $\{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), ..., (\boldsymbol{x}_b, y_b)\} \sim (\mathcal{X}, \mathcal{Y})$;
    Compute convex relaxations $\mathbb{C}_l(\boldsymbol{x}_1), \mathbb{C}_l(\boldsymbol{x}_2), ..., \mathbb{C}_l(\boldsymbol{x}_k)$;
    Initialize $\boldsymbol{x}'_1 \sim \mathbb{C}_l(\boldsymbol{x}_1), \boldsymbol{x}'_2 \sim \mathbb{C}_l(\boldsymbol{x}_2), ..., \boldsymbol{x}'_b \sim \mathbb{C}_l(\boldsymbol{x}_b)$;
    for $j \leq b$ **do**
      Update in parallel $\boldsymbol{x}'_j \leftarrow \Pi_{\mathbb{C}_l(\boldsymbol{x}_j)}(\boldsymbol{x}'_j + \alpha \nabla_{\boldsymbol{x}'_j} \mathcal{L}(h_\theta^{j+1:k}(\boldsymbol{x}'_j), y_j))$;
    end
    Update parameters $\theta \leftarrow \theta - \eta \frac{1}{M} \sum_{j=1}^{b} \nabla_\theta \mathcal{L}(h_\theta^{j+1:k}(\boldsymbol{x}'_j), y_j)$;
  end
  Freeze parameters θ_l of layer l;
end

the maximum loss when this point is propagated further through the network (this forward pass is shown with the blue line). Finally, we backpropagate the final loss (red line) and update the parameters of the network so to minimize the loss. Critically, we do not backpropagate through the convex relaxation in the first layer as standard provable defenses do (Wong & Kolter, 2018; Mirman et al., 2018; Gowal et al., 2018). We instead freeze the first layer and stop backpropagation after the update of the second layer. Because of this, our optimization problem is significantly easier – the neural network only has to learn to behave well on the *concrete points* that were found in the convex region $\mathbb{C}_l(\boldsymbol{x})$. This can be viewed as an extension of the robust optimization method that Madry et al. (2018) found to work well in practice.

We then proceed with the above process for later layers. Formally, this training process amounts to (approximately) solving the following min-max optimization problem at the l-th step:

$$\min_{\theta^{l+1:k}} \mathbb{E}_{(\mathrm{x},\mathrm{y})\sim D} \max_{\boldsymbol{x}'_l \in \mathbb{C}_l(\boldsymbol{x})} \mathcal{L}(h_\theta^{l+1:k}(\boldsymbol{x}'_l), y, \theta) \tag{3}$$

Note that for $l = 0$ this formulation is equivalent to the standard min-max formulation in Equation 2 because $\mathbb{C}_0(\boldsymbol{x}) = \mathbb{S}_0(\boldsymbol{x})$. Our approach to solve this min-max optimization problem for every layer l is shown in Algorithm 1. We initialize every batch by random sampling from the corresponding convex region. Then, in every iteration we use projected gradient descent (PGD) to maximize the inner loss in 3. We first update $\boldsymbol{x}'_j$ in the direction of the gradient of the loss and then project it back to $\mathbb{C}_l(\boldsymbol{x}_j)$ using the projection operator Π. Note that this approach assumes the existence of an efficient projection method to the particular convex relaxation the method is instantiated with. In the next section, we show how to instantiate the training algorithm described above to a particular convex relaxation which is generally tighter than a hyperrectangle and where we derive an efficient projection operation.

5 Layerwise adversarial training using linear relaxations

So far we described the general approach of layerwise adversarial training. Now we show how to instantiate it for a particular convex relaxation based on linear approximations. If instead one would use interval approximation (Mirman et al., 2018; Gowal et al., 2018) as the convex relaxation, then all regions $\mathbb{C}_l(\boldsymbol{x})$ will be hyperrectangles and projection to these sets is fast and simple. However, the interval relaxation provides a coarse approximation which motivates the need to train with relaxations that provide tighter bounds. Thus, we consider linear relaxations which are generally tighter than those based on intervals.

In particular we leverage the same relaxation which was previously proposed in Wong & Kolter (2018); Weng et al. (2018); Singh et al. (2018) as an effective way to certify neural networks. Here, each convex region is represented as a set $\mathbb{C}_l(\boldsymbol{x}) = \{\boldsymbol{a}_l + \boldsymbol{A}_l \boldsymbol{e} \mid \boldsymbol{e} \in [-1, 1]^{m_l}\}$. Vector $\boldsymbol{a}_l$ represents the center of the set and the matrix $\boldsymbol{A}_l$ represents the affine transformation of the hypercube

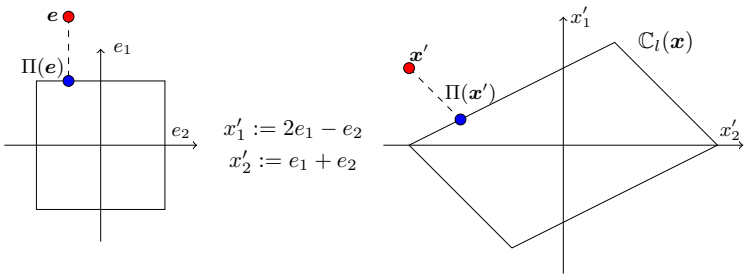

Figure 2: Projection to a region based on linear relaxation using change of variables.

$[-1, 1]^{m_l}$. The initial convex region $\mathbb{C}_0(\boldsymbol{x})$ is represented using $\boldsymbol{a}_0 = \boldsymbol{x}$ and $\boldsymbol{A}_0 = \epsilon \boldsymbol{I}_{d_0}$ is a diagonal matrix. Propagation of these convex regions through the network is out of the scope of this paper – a full description can be found in Wong & Kolter (2018) or Singh et al. (2018). At a high level, the convolutional and fully connected layers are handled by multiplying $\boldsymbol{A}_l$ and $\boldsymbol{a}_l$ by appropriate matrices. To handle the ReLU activation, for ReLU units which cross 0, we apply a convex relaxation which amounts to multiplying $\boldsymbol{A}_l$ and $\boldsymbol{a}_l$ by appropriately chosen scalar values, depending whether the ReLU is activated or not. Using this relaxation of ReLU, we can recursively obtain all convex regions $\mathbb{C}_l(\boldsymbol{x})$. We provide more detailed description of this propagation in Appendix A.1.

Projection to linear convex regions To use our training method we now need to instantiate Algorithm 1 with a suitable projection operator $\Pi_{\mathbb{C}_l(\boldsymbol{x})}$. The key insight here is that the vector $\boldsymbol{x}' \in C_l(\boldsymbol{x})$ is uniquely determined by auxiliary vector $\boldsymbol{e} \in [-1, 1]^{m_l}$ where $\boldsymbol{x}' = \boldsymbol{a}_l + \boldsymbol{A}_l \boldsymbol{e}$. Then instead of directly solving for $\boldsymbol{x}'$ which requires projecting to $\mathbb{C}_l(\boldsymbol{x})$, we can solve for $\boldsymbol{e}$ instead which would uniquely determine $\boldsymbol{x}$. Crucially, the domain of $\boldsymbol{e}$ is a hyperrectangle $[-1, 1]^{m_l}$ which is easy to project to. To visualize this further we provide an example in Figure 2. The goal is to project the red point $\boldsymbol{x}'$ in the right picture to the convex region $\mathbb{C}_l(\boldsymbol{x})$. To project, we first perform change of variables to substitute $\boldsymbol{x}'$ with $\boldsymbol{e}$ and then project $\boldsymbol{e}$ to the square $[-1, 1] \times [-1, 1]$ to obtain the blue point $\Pi(\boldsymbol{e})$ on the left. Then, we again perform change of variables to obtain the blue point $\Pi(\boldsymbol{x}')$ on the right, the projection of $\boldsymbol{x}'$ we were looking for.

Based on these observations, we modify Line 7 of Algorithm 1 to first update the coefficients $\boldsymbol{e}_j$ using the following update rule: $\boldsymbol{e}_j \leftarrow clip(\boldsymbol{e}_j + \alpha \boldsymbol{A}_l^T \nabla_{\boldsymbol{x}'_j} \mathcal{L}(\boldsymbol{x}'_j, y_j), -1, 1)$. Here $clip$ is function which thresholds its argument between -1 and 1, formally $clip(x, -1, 1) = \min(\max(x, -1), 1)$. This is followed by an update to $\boldsymbol{x}'_j$ via $\boldsymbol{x}'_j \leftarrow \boldsymbol{a}_l + \boldsymbol{A}_l \boldsymbol{e}_j$, completing the update step.

Sparse representation While our representation of convex regions with matrix $\boldsymbol{A}_l$ and vector $\boldsymbol{a}_l$ has clean mathematical properties, in practice, a possible issue is that the matrix $\boldsymbol{A}_l$ can grow to be quite large. Because of this, propagating it through the network can be memory intensive and prohibit the use of larger batches. To overcome this difficulty, we first observe that $\boldsymbol{A}_l$ is quite sparse. We start with a very sparse, diagonal matrix $\boldsymbol{A}_0$ at the input. After each convolution, an element of matrix $\boldsymbol{A}_{l+1}$ is non-zero only if there is a non-zero element inside of its convolutional kernel in matrix $\boldsymbol{A}_l$. We can leverage this observation to precompute positions of all non-zero elements in matrix $\boldsymbol{A}_{l+1}$ and compute their values using matrix multiplication. This optimization is critical to enabling training to take place altogether. An interesting item for future work is further optimizing the current relaxation (via a specialized GPU implementation) or developing more memory friendly relaxations, so to scale the training to larger networks.

6 Certification of Neural Networks

After training a neural network via layerwise adversarial training, our goal is to certify the target property (e.g., robustness). Here we leverage certification techniques which are not fast enough to be incorporated into the training procedure, but which can significantly boost the certification performance.

Refinement of the linear approximation The linear relaxation of ReLU that we are using is parameterized by slopes $\boldsymbol{\lambda}$ of the linear relaxation. Prior work which employed this relaxation (Wong & Kolter, 2018; Weng et al., 2018; Singh et al., 2018) chose these slopes in a greedy manner by minimizing the area of the relaxation. During training we also choose $\boldsymbol{\lambda}$ in the same way. However, during certification, we can also optimize for the values of $\boldsymbol{\lambda}$ that give rise to the convex region inside of which the maximum loss is minimized. This optimization problem can be written as:

$$\min_{\boldsymbol{\lambda}\in[0,1]^{d_l}} \max_{\boldsymbol{x}'\in\mathbb{C}_l(\boldsymbol{x};\boldsymbol{\lambda})} \mathcal{L}(h_\theta^{l+1:k}(\boldsymbol{x}'), y)$$

Solving this is computationally too expensive inside the training loop, but during certification it is feasible to approximate the solution. We solve for $\boldsymbol{\lambda}$ using the Adam optimizer and clipping the elements between 0 and 1 after each update. We remark that the idea of learning the slope is similar to Dvijotham et al. (2018b) who propose to optimize dual variables in a dual formulation, however here we stay in the primal formulation.

Combining convex relaxations with exact bound propagation During layerwise adversarial training we essentially train the network to be certified on all regions $\mathbb{C}_0(\boldsymbol{x}), ..., \mathbb{C}_k(\boldsymbol{x})$. While computing exact regions $\mathbb{S}_l(\boldsymbol{x}) \subseteq \mathbb{C}_l(\boldsymbol{x})$ is not feasible during training, we can afford it to some extent during certification. The idea is to first propagate the bounds using convex relaxations until one of the hidden layers l and obtain a region $\mathbb{C}_l(\boldsymbol{x})$. If training was successful, there should not exist a concrete point $\boldsymbol{x}'_l \in \mathbb{C}_l(\boldsymbol{x})$ which, if propagated through the network, violates the correctness property in Equation 1. We can encode both, the property and the propagation of the *exact bounds* $\mathbb{S}_l(\boldsymbol{x})$ using a Mixed-Integer Linear Programming (MILP) solver. Note that we can achieve this because we represent the region $\mathbb{C}_l(\boldsymbol{x})$ using a set of linear constraints, however, for general convex shapes this may not be possible. We perform the MILP encoding using the formulation from Tjeng et al. (2019). It is usually possible to encode only the last two layers using MILP due to the poor scalability of these solvers for realistic network sizes. One further improvement we also include is to tighten the convex regions $\mathbb{C}_l(\boldsymbol{x})$ using refinement via linear programming as described in Singh et al. (2019a). We remark that this combination of convex relaxation and exact bound propagation does not fall under the recently introduced convex barrier to certification Salman et al. (2019b).

7 Experimental Evaluation

We now present an evaluation of our training method on the challenging CIFAR-10 dataset.

Experimental setup We evaluate on a desktop PC with 2 GeForce RTX 2080 Ti GPU-s and 16-core Intel(R) Core(TM) i9-9900K CPU @ 3.60GHz. We use Gurobi as a MILP solver. Our method is implemented in PyTorch and we plan to release both, the code and the trained models.

Neural network architecture All presented results are on a 3-layer convolutional network with 16 544 neurons: first 2 layers are convolutional layers with 32 and 128 filters, and both have kernel size 4 and stride 2. These are followed by a fully connected layer with 150 hidden units. After each of these layers, there is a ReLU activation.

Training We use batch size 50 and L1 regularization 0.00001 for training. We perform optimization using Adam (Kingma & Ba, 2014) with initial learning rate 0.001 which is decreased by $10\times$ every 100 epochs. During layerwise training we start with ϵ perturbation which is 10% higher than the one we certify and we decrease it by 5% when the training progresses to the next layer.

Certification After training completes, we perform certification as follows: for every image, we first try to certify it using only linear relaxations (with the improvement of learned slopes, Section 6). If this fails, we encode the last layer as MILP and try again. Finally, if this fails we encode the ReLU activation after the last convolution using additional 300 binary variables and the rest using the triangle formulation Ehlers (2017). We consider an image to be not certifiable if we fail to verify it using these methods. We always evaluate on the first 1 000 images from the test set.

Comparison to prior work We first train and certify using our method for the L_∞ perturbation 2/255. Results are shown in Table 1. We always compare to the best reported and reproducible

Table 1: Evaluation on CIFAR-10 dataset with L_∞ perturbation 2/255

Method	Accuracy(%)	Certified Robustness(%)
Our work	**74.8**	**55.9**
Wong et al. (2018)	68.3	53.9
Gowal et al. (2018)	70.1	50.0
Zhang et al. (2019)	59.9	46.1
Mirman et al. (2018)	62.3	45.5
Xiao et al. (2019)	61.1	45.9

Table 2: Evaluation on CIFAR-10 dataset with L_∞ perturbation 8/255

Method	Accuracy(%)	Certified Robustness(%)
Our work	**46.2**	24.4
Wong et al. (2018)	28.7	21.8
Zhang et al. (2019)	41.3	**28.8**
Mirman et al. (2018)	**46.2**	27.2
Xiao et al. (2019)	40.5	20.3

results in the literature on *any* architecture. We do not compare to smoothing-based approaches Cohen et al. (2019), as these provide probabilistic instead of exact guarantees. Extensions to Cohen et al. (2019) such as Salman et al. (2019a) also use additional existing techniques such as pre-training on ImageNet and unlabeled data which are orthogonal. We also do not compare to using cascades from Wong et al. (2018), as this improvement is also orthogonal to the method here. Thus, we only consider their best *single* network architecture (inline with prior work Zhang et al. (2019) which compares to a single architecture). We believe all methods listed in Table 1, including ours, would benefit from additional techniques such as cascades, pre-training and leveraging unlabeled data.

Experimentally, we find that the neural network trained using our method substantially outperforms all existing approaches, both in terms of standard accuracy and certified robustness for 2/255. Note that here we are using the *same linear relaxation* as Wong et al. (2018), but our optimization procedure is different and shows significant improvements over the one used in their work. We also run the same experiment for L_∞ perturbation 8/255. Here we do not include comparison with Gowal et al. (2018) as their results were found to be not reproducible (Zhang et al., 2019; Mirman et al., 2019; Xu, 2019). These results are presented in Table 2. Here we match state-of-the-art accuracy 46.2% reported by Mirman et al. (2018) and substantially outperform all other approaches. However, in terms of certified robustness we are not able to achieve similar results to Zhang et al. (2019) whose method is based on a combination of interval approximation and linear relaxation. The main issue is that our 3-layer network lacks capacity to solve this task – even if training only using standard adversarial training our empirical robustness does not go above $\sim 32\%$. We remark that capacity was found to be one of the key components necessary to obtain a robust classifier (Madry et al., 2018). Indeed, prior work achieves best results using significantly larger networks, e.g. Mirman et al. (2018) achieve these results on a network with ~ 170000 neurons, more than 10 times larger than ours. Due to promising results for 2/255, we believe achieving state-of-the-art results for 8/255 is very likely an issue of instantiating our method with a convex relaxation that is more memory efficient, which we believe is an interesting item for future work.

8 Conclusion

We presented a new method to train certified neural networks. The key concept was to combine techniques from provable defenses using convex relaxations with those of adversarial training. Our method achieves state-of-the-art 74.8% accuracy and 55.9% certified robustness on CIFAR-10 with a 2/255 L_∞ perturbation, significantly outperforming prior work when considering a single network (it also achieves competitive results on 8/255 L_∞). The method is general and can be instantiated with *any* convex relaxation. A promising future work item is scaling to larger networks: this will require tight convex relaxations with a low memory footprint that allow for efficient projection.

REFERENCES

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

Battista Biggio, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim Šrndić, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. Evasion attacks against machine learning at test time. In *Joint European conference on machine learning and knowledge discovery in databases*, 2013.

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

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

Krishnamurthy Dvijotham, Sven Gowal, Robert Stanforth, Relja Arandjelovic, Brendan O'Donoghue, Jonathan Uesato, and Pushmeet Kohli. Training verified learners with learned verifiers. *arXiv preprint arXiv:1805.10265*, 2018a.

Krishnamurthy Dvijotham, Robert Stanforth, Sven Gowal, Timothy A Mann, and Pushmeet Kohli. A dual approach to scalable verification of deep networks. In *UAI*, 2018b.

Ruediger Ehlers. Formal verification of piece-wise linear feed-forward neural networks. In *International Symposium on Automated Technology for Verification and Analysis*, pp. 269–286. Springer, 2017.

Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, Petar Tsankov, Swarat Chaudhuri, and Martin Vechev. Ai2: Safety and robustness certification of neural networks with abstract interpretation. In *2018 IEEE Symposium on Security and Privacy*, 2018.

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

Sven Gowal, Krishnamurthy Dvijotham, Robert Stanforth, Rudy Bunel, Chongli Qin, Jonathan Uesato, Timothy Mann, and Pushmeet Kohli. On the effectiveness of interval bound propagation for training verifiably robust models. *arXiv preprint arXiv:1810.12715*, 2018.

Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. In *International Conference on Computer Aided Verification*, pp. 97–117. Springer, 2017.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

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

Mathias Lecuyer, Vaggelis Atlidakis, Roxana Geambasu, Daniel Hsu, and Suman Jana. Certified robustness to adversarial examples with differential privacy. *2019 IEEE Symposium on Security and Privacy (SP)*, pp. 656–672, 2018.

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

Matthew Mirman, Timon Gehr, and Martin Vechev. Differentiable abstract interpretation for provably robust neural networks. In *International Conference on Machine Learning*, 2018.

Matthew Mirman, Gagandeep Singh, and Martin Vechev. A provable defense for deep residual networks. *arXiv preprint arXiv:1903.12519*, 2019.

Chongli Qin, Krishnamurthy (Dj) Dvijotham, Brendan O'Donoghue, Rudy Bunel, Robert Stanforth, Sven Gowal, Jonathan Uesato, Grzegorz Swirszcz, and Pushmeet Kohli. Verification of non-linear specifications for neural networks. In *International Conference on Learning Representations*, 2019.

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

Hadi Salman, Greg Yang, Jerry Li, Pengchuan Zhang, Huan Zhang, Ilya Razenshteyn, and Sebastien Bubeck. Provably robust deep learning via adversarially trained smoothed classifiers. *arXiv preprint arXiv:1906.04584*, 2019a.

Hadi Salman, Greg Yang, Huan Zhang, Cho-Jui Hsieh, and Pengchuan Zhang. A convex relaxation barrier to tight robust verification of neural networks. *arXiv preprint arXiv:1902.08722*, 2019b.

Gagandeep Singh, Timon Gehr, Matthew Mirman, Markus Püschel, and Martin Vechev. Fast and effective robustness certification. In *Advances in Neural Information Processing Systems*, 2018.

Gagandeep Singh, Timon Gehr, Markus Puschel, and Martin Vechev. Robustness certification with refinement. In *International Conference on Learning Representations*, 2019a.

Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. An abstract domain for certifying neural networks. *Proceedings of the ACM on Programming Languages*, (POPL), 2019b.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

Vincent Tjeng, Kai Y. Xiao, and Russ Tedrake. Evaluating robustness of neural networks with mixed integer programming. In *International Conference on Learning Representations*, 2019.

Lily Weng, Huan Zhang, Hongge Chen, Zhao Song, Cho-Jui Hsieh, Luca Daniel, Duane Boning, and Inderjit Dhillon. Towards fast computation of certified robustness for ReLU networks. In *Proceedings of the 35th International Conference on Machine Learning*, 2018.

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

Eric Wong, Frank Schmidt, Jan Hendrik Metzen, and J. Zico Kolter. Scaling provable adversarial defenses. In *Advances in Neural Information Processing Systems 31*. 2018.

Kai Y. Xiao, Vincent Tjeng, Nur Muhammad (Mahi) Shafiullah, and Aleksander Madry. Training for faster adversarial robustness verification via inducing reLU stability. In *International Conference on Learning Representations*, 2019.

Cihang Xie, Yuxin Wu, Laurens van der Maaten, Alan L Yuille, and Kaiming He. Feature denoising for improving adversarial robustness. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 501–509, 2019.

Weilin Xu. Reproducibility issue. 2019. URL `https://github.com/deepmind/interval-bound-propagation/issues/1#issuecomment-492552237`.

Huan Zhang, Hongge Chen, Chaowei Xiao, Bo Li, Duane Boning, and Cho-Jui Hsieh. Towards stable and efficient training of verifiably robust neural networks. *arXiv preprint arXiv:1906.06316*, 2019.

A APPENDIX

Here we provide additional details that were omitted in the main body of the paper.

A.1 CONVEX RELAXATION USING LINEAR APPROXIMATIONS

Here we describe how we propagate convex relaxations of the form $\mathbb{C}_l(\boldsymbol{x}) = \{\boldsymbol{a}_l + \boldsymbol{A}_l\boldsymbol{e} \mid \boldsymbol{e} \in [-1,1]^{m_l}\}$ through the network. As explained before, these relaxations were previously proposed in Wong & Kolter (2018); Weng et al. (2018); Singh et al. (2018). For the sake of completeness we describe them here using our notation. The initial convex region $\mathbb{C}_0(\boldsymbol{x})$ is represented using $\boldsymbol{a}_0 = \boldsymbol{x}$ and $\boldsymbol{A}_0 = \epsilon \boldsymbol{I}_{d_0}$ is a diagonal matrix. Now we describe how to propagate these regions through different operations in the neural network. Depending on the form of function h^i_θ representing operation applied at layer i we distinguish different cases. Here we assume we have obtained region $\mathbb{C}_i(\boldsymbol{x})$ and our goal is to compute the region $\mathbb{C}_{i+1}(\boldsymbol{x})$ using convex relaxation g^i_θ of the function h^i_θ.

Convolutional layers Consider the case where h^i_θ is a convolution parametrized by kernel matrix $\boldsymbol{W}_\theta$ and bias $\boldsymbol{b}_\theta$ meaning that $h^i_\theta(\boldsymbol{x}') = conv(\boldsymbol{x}, \boldsymbol{W}_\theta) + \boldsymbol{b}_\theta$. Recall that $\boldsymbol{x}' = \boldsymbol{a}_l + \boldsymbol{A}_l\boldsymbol{e}$ where $\boldsymbol{e} \in [-1,1]^{m_l}$. We can then compute:

$$\begin{aligned} h^i_\theta(\boldsymbol{x}') &= conv(\boldsymbol{x}', \boldsymbol{W}_\theta) + \boldsymbol{b}_\theta \\ &= conv(\boldsymbol{a}_l + \boldsymbol{A}_l\boldsymbol{e}, \boldsymbol{W}_\theta) + \boldsymbol{b}_\theta \\ &= conv(\boldsymbol{a}_l, \boldsymbol{W}_\theta) + conv(\boldsymbol{A}_l\boldsymbol{e}, \boldsymbol{W}_\theta) + \boldsymbol{b}_\theta \\ &= conv(\boldsymbol{a}_l, \boldsymbol{W}_\theta) + conv(\boldsymbol{A}_l, \boldsymbol{W}_\theta)\boldsymbol{e} + \boldsymbol{b}_\theta \end{aligned}$$

Using this formula, we can define region $\mathbb{C}_{l+1}(\boldsymbol{x}) = \{\boldsymbol{a}_{l+1} + \boldsymbol{A}_{l+1}\boldsymbol{e} \mid \boldsymbol{e} \in [-1,1]^{m_{l+1}}\}$ where:

$$\begin{aligned} m_{l+1} &= m_l \\ a_{l+1} &= conv(\boldsymbol{a}_l, \boldsymbol{W}_\theta) + \boldsymbol{b}_\theta \\ A_{l+1} &= conv(\boldsymbol{A}_l, \boldsymbol{W}_\theta) \end{aligned}$$

Fully-connected layers Consider the case where h^i_θ is a fully connected layer parametrized by weight matrix $\boldsymbol{W}_\theta$ and bias $\boldsymbol{b}_\theta$ meaning that $h^i_\theta(\boldsymbol{x}') = \boldsymbol{W}_\theta\boldsymbol{x}' + \boldsymbol{b}_\theta$. Recall that $\boldsymbol{x}' = \boldsymbol{a}_l + \boldsymbol{A}_l\boldsymbol{e}$ where $\boldsymbol{e} \in [-1,1]^{m_l}$. We can then compute:

$$\begin{aligned} h^i_\theta(\boldsymbol{x}') &= \boldsymbol{W}_\theta\boldsymbol{x}' + \boldsymbol{b}_\theta \\ &= \boldsymbol{W}_\theta(\boldsymbol{a}_l + \boldsymbol{A}_l\boldsymbol{e}) + \boldsymbol{b}_\theta \\ &= \boldsymbol{W}_\theta\boldsymbol{a}_l + \boldsymbol{W}_\theta\boldsymbol{A}_l\boldsymbol{e} + \boldsymbol{b}_\theta \end{aligned}$$

Using this formula, we can define region $\mathbb{C}_{l+1}(\boldsymbol{x}) = \{\boldsymbol{a}_{l+1} + \boldsymbol{A}_{l+1}\boldsymbol{e} \mid \boldsymbol{e} \in [-1,1]^{m_{l+1}}\}$ where:

$$\begin{aligned} m_{l+1} &= m_l \\ a_{l+1} &= \boldsymbol{W}_\theta\boldsymbol{a}_l + \boldsymbol{b}_\theta \\ A_{l+1} &= \boldsymbol{W}_\theta\boldsymbol{A}_l \end{aligned}$$

ReLU activation In this case we can assume that h^i_θ is ReLU function, $h^i_\theta(\boldsymbol{x}') = \max(\boldsymbol{x}', 0)$ applied componentwise. We will explain the transformation of a single element $\boldsymbol{x}'_i = \boldsymbol{a}_l + \boldsymbol{A}_{l,i}\boldsymbol{e}$. We first compute lower and upper bound l_i, u_i that this element can take in the convex region $\mathbb{C}_l(\boldsymbol{x})$:

$$\begin{aligned} l_i &= \boldsymbol{a}_l + \sum_{j=1}^{m_l} A_{l,i,j} \\ u_i &= \boldsymbol{a}_l - \sum_{j=1}^{m_l} A_{l,i,j} \end{aligned}$$

If $u_i < 0$ then $ReLu(\boldsymbol{x}') = 0$ and if $l_i > 0$ then $ReLU(\boldsymbol{x}') = \boldsymbol{x}'$. In the other case where 0 is between l_i and u_i we define $ReLU(\boldsymbol{x}') = \lambda_i \boldsymbol{x}' + \mu e_{m_l+1}$ where $e_{m_l+1} \in [-1, 1]$ is a new coefficient. Formulas for λ and μ are the following:

$$\lambda = u_i/(u_i - l_i)$$
$$\mu = -\frac{1}{2} u_i l_i/(u_i - l_i)$$

This computation can be written also in the matrix form to obtain matrix $\boldsymbol{\Lambda}$ and vector $\boldsymbol{b}$ which would update:

$$\boldsymbol{A}_{l+1} = \boldsymbol{\Lambda} A_l$$
$$\boldsymbol{a}_{l+1} = \boldsymbol{a}_l + \boldsymbol{b}$$