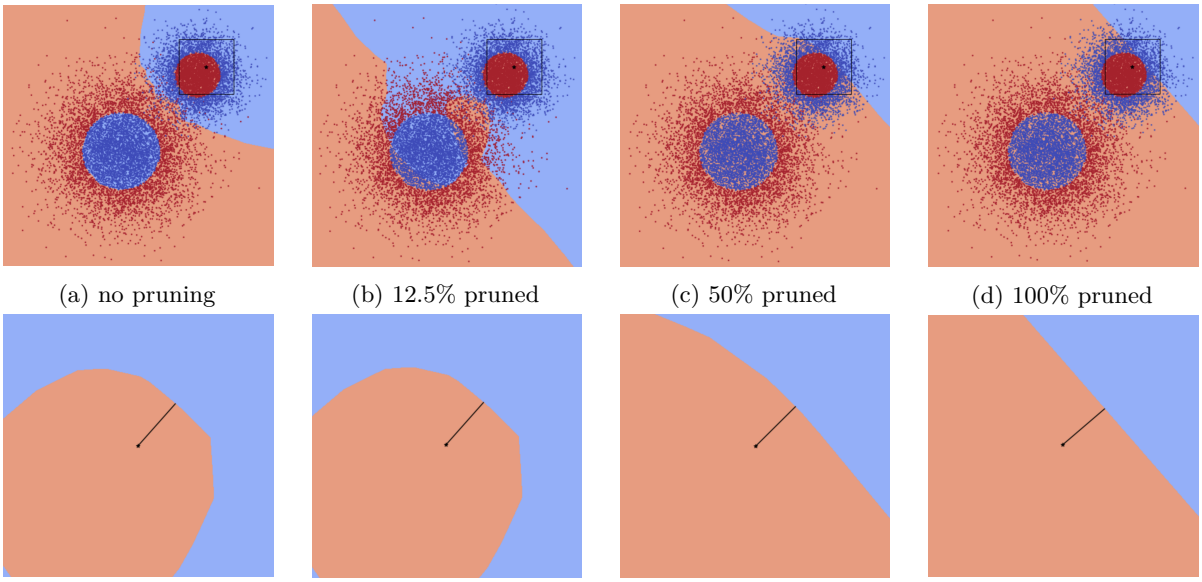
Robustness Evaluation Using Local Substitute Networks

Anonymous authors
Paper under double-blind review

Abstract

The robustness of a neural network against adversarial examples is essential when a deep classifier is applied in safety-critical use cases like health care or autonomous driving. To assess the robustness, practitioners use various tools ranging from adversarial attacks to the exact computation of the distance to the decision boundary. We use the fact that the robustness of a neural network is a local property and empirically show that computing the same metrics for smaller local substitute networks yields reasonable estimates of the robustness for a lower cost. To construct the substitute network, we develop several pruning techniques that preserve the local properties of the initial network around a given anchor point. Our experiments on multiple datasets prove that this approach saves a significant amount of computation.



(a) no pruning (b) 12.5% pruned (c) 50% pruned (d) 100% pruned

(e) Loc., no pruning (f) Loc., 12.5% pruned (g) Loc., 50% pruned (h) Loc., 100% pruned

Figure 1: A toy example in the two-dimensional setting. We show the boundaries of a classifier, before (1a and 1e) and after the pruning (1b - 1d and 1f - 1h), when up to the 100% of the hidden neurons are removed. The sample we are interested in is marked by the black point. The square around it on the plots with the global view (1a - 1d) shows the local region that is depicted on the other four plots (1e - 1h). While the global behavior changes a lot, the boundary around the chosen anchor point remains similar. Most importantly, the distance to the closest adversarial from the anchor point (shown by the black line on the plots 1e - 1h) does not change significantly.

1 Introduction

The impressive success of neural networks in various complicated tasks makes them a popular choice for practitioners despite the known flaws. One problem that continuously gains more attention is the robustness of the deep neural classifiers. While multiple notions of robustness exist, depending on the use case, we consider the most basic concept of robustness against adversarial examples—small perturbations of the correctly classified samples that lead to a false prediction. The presence of adversarial examples severely limits the application of neural networks in safety-critical tasks like health care and autonomous driving, where the data is collected from sensors, and it is not acceptable that the same image, for example, a road sign, is classified differently depending on the signal noise.

While the problem of lacking robustness for the neural network classifiers is widely known, formal robustness verification methods do not allow for assessing the classifier's robustness when the network is large or require specific modifications in the network's architecture or training procedure. In fact, Katz et al. (2017) show that the exact verification task for the ReLU classifiers is NP-complete. Therefore, constructing adversarial attacks and measuring the perturbation's magnitude, which is required to change the prediction, is still one of the most popular ways to estimate a network's robustness. The farther away the adversarial point is from the initial sample, the more robust behavior we expect from the network around this point. Unfortunately, the distance to an adversarial point provides only an upper bound on the distance to the decision boundary. Conversely, formal verification methods output a lower bound on that value by certifying a region around the sample as adversarial-free.

In this work, we develop a novel inexact robustness assessment method that utilizes the fact that the robustness of a network against adversarial perturbations around a given sample is a local property. That is, it does not depend on the behavior of the network outside of the sample's neighborhood. In other words, two networks with similar decision boundaries around the same anchor point must have similar robustness properties, despite showing completely different behavior away from its local neighborhood. Based on these observations, we

1. **develop a novel method to assess the robustness of the deep neural classifiers** based on the local nature of the robustness properties,
2. **develop three pruning techniques** that remove the non-linear activation functions to reduce the complexity of the verification task, but preserve the local behavior of the network as much as possible: one replacing activations by a constant value, one preserving the output of the initial network for one adversarial point and replacing activation functions by the linear ones, and one preserving the network behavior in a whole region around the anchor point,
3. **develop a method to identify the neurons to prune** that is based on a bound propagation technique and
4. **empirically verify that the robustness metrics computed on the pruned substitute networks are good estimates of the robustness of the initial network** by conducting experiments with convolutional networks on MNIST and CIFAR10 datasets.

In Figure 1, we show an example of the difference in the decision boundary before and after we apply one of the proposed pruning techniques in the two-dimensional setting. We remove up to all (1d and 1h) of the hidden neurons while retaining the essential properties of the decision boundary locally around the anchor point. That means to solve the complex task of finding the distance to the decision boundary for the initial model, we save cost by working with the simple local substitute and still get a good approximation of the exact solution.

This work is organized as follows. In Section 2, we introduce the necessary notation and formalize the context of our analysis. In Section 3, we develop both pruning techniques and explain how we focus on the local neighborhood around the base sample. Further, in Section 4, we set up the experimentation workflow and show the results. In Section 5, we mention the relevant related work, and, finally, we present options for future research in Section 6.

2 Notation

The general idea and our pruning methods apply to any deep classifier, and the constraints apply depending on the deployed attack, verification, and bound propagation approaches. We consider a neural network f consisting of L linear layers, dense or convolutional, and $L-1$ activation layers. We denote the number of neurons in each layer l as n_l for $l \in \{0, \dots, L\}$, where the zero's layer contains the input neurons. Furthermore, we consider the L^∞-norm as the perturbation norm since most of the verification methods support it. For a correctly classified anchor point $\boldsymbol{x}^0 \in \mathbb{R}^{n_0}$, weight matrices $\boldsymbol{W}^l \in \mathbb{R}^{n_l \times n_{l-1}}$, and bias vectors $\boldsymbol{b}^l \in \mathbb{R}^{n_l}$, the output of the i-th neuron in the l-th layer before applying the activation function σ is $f_i^l(\boldsymbol{x}^0) = \boldsymbol{W}_{i,:}^l \boldsymbol{x}^{l-1} + \boldsymbol{b}_i^l$ and for the last layer $f^L(\boldsymbol{x}^0) = \boldsymbol{W}^L \boldsymbol{x}^{L-1} + \boldsymbol{b}^L = f(\boldsymbol{x}^0)$. Next, we introduce the notation for four metrics commonly used to assess the robustness of the given classifier. All these algorithms take the network's parameters $\{\boldsymbol{W}^l, \boldsymbol{b}^l\}_{l \in \{1,\dots,L\}}$ and $\boldsymbol{x}^0$ as input.

Pre-activation bounds propagation A pre-activation bounds propagation (or just bounds propagation) algorithm $\mathcal{B}$ is an algorithm that additionally takes a perturbation budget ϵ as an input and returns a set of the lower and upper bounds $\{\underline{\boldsymbol{a}}^l, \bar{\boldsymbol{a}}^l\}_{l \in \{1,\dots,L\}}$ on the possible pre-activation values $\boldsymbol{W}^l \boldsymbol{x}^{l-1} + \boldsymbol{b}^l$. In other words, for all possible points $\tilde{\boldsymbol{x}}^0$ with $\|\boldsymbol{x}^0 - \tilde{\boldsymbol{x}}^0\|_\infty \leq \epsilon$ it must hold that $\underline{\boldsymbol{a}}^l \leq \boldsymbol{W}^l \tilde{\boldsymbol{x}}^{l-1} + \boldsymbol{b}^l \leq \bar{\boldsymbol{a}}^l$ in every layer l.

Adversarial attack An (untargeted) adversarial attack algorithm $\mathcal{A}$ is an algorithm that returns a point $\boldsymbol{x}^0_{\text{adv}}$ that is assigned by f to a different class than $\boldsymbol{x}^0$. We denote the magnitude of the adversarial perturbation $\|\boldsymbol{x}^0 - \boldsymbol{x}^0_{\text{adv}}\|_\infty$ by $\mathcal{A}(f) = d_{\text{adv}}$. The smaller the distance d_{adv}, the better.

Distance to the decision boundary Finally, an algorithm $\mathcal{D}$ computing the distance to the decision boundary from $\boldsymbol{x}^0$ is an algorithm that outputs the distance $\|\boldsymbol{x}^0 - \tilde{\boldsymbol{x}}^0_{\text{adv}}\|_\infty$ to the closest adversarial point $\tilde{\boldsymbol{x}}^0_{\text{adv}}$. Formally, $\mathcal{D}$ must solve the following optimization problem.

$$\min_{\tilde{y} \neq y} \min_{\tilde{\boldsymbol{x}}^0 \in \mathbb{R}^{n_0}} \|\tilde{\boldsymbol{x}}^0 - \tilde{\boldsymbol{x}}^0\|_\infty \text{ s.t. } f_y(\boldsymbol{x}^0) \leq f_{\tilde{y}}(\tilde{\boldsymbol{x}}^0), \tag{1}$$

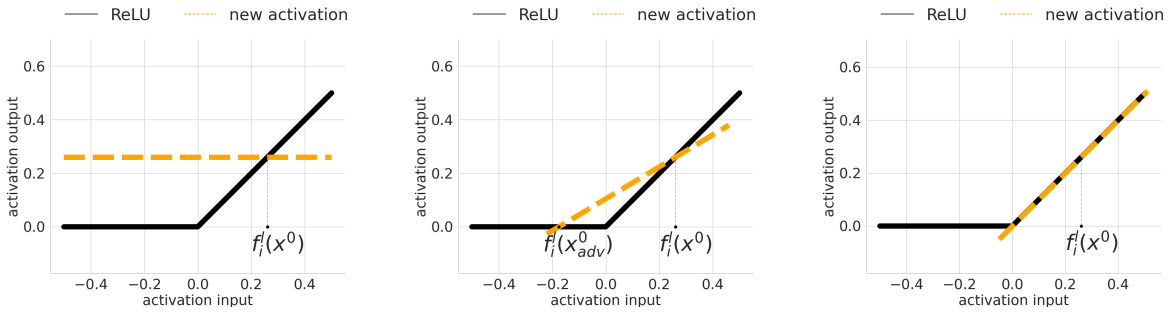
where y is the correctly predicted label of $\boldsymbol{x}^0$. We denote the optimal objective function value of the minimization problem equation 1 by $\mathcal{D}(f) = d_{\text{bnd}}$.

Note, that for any $\mathcal{A}$ and $\mathcal{D}$ it holds that $d_{\text{bnd}} \leq d_{\text{adv}}$. When we consider a pair of networks f and the pruned network g we define $\hat{d}_{\text{adv}}$ and $\hat{d}_{\text{bnd}}$ as $\mathcal{A}(g)$ and $\mathcal{D}(g)$ correspondingly.

3 Local substitutes

Given the initial deep classifier, we reduce its size to construct a neural network that allows for efficient robustness verification. Practitioners pursue the same goal when the network has to be deployed in a setting with strictly constrained available memory or when the forward pass must be particularly fast. In these scenarios, it is common to prune the network by removing its neurons and weights while keeping the drop in performance on the main task as low as possible and even improving generalization to the new data. See Section 5 for an overview of the existing pruning methods.

However, these available pruning techniques are not suitable for our application. First, all pruning techniques act globally and consider the network's behavior on the whole available dataset. Instead, we show how to prune the network such that only the vicinity of a particular point is considered, which determines its robustness properties. Second, when we assess the robustness of an already trained network, we do not care about the classification error of the resulting pruned network, as we do not use it for classification. After reducing the network complexity, our most crucial goal is to preserve a similar decision boundary in a small region around the anchor point. To achieve this property, we prune the network differently for each anchor point. However, as we show in Section 4, the time overhead of the pruning step is insignificant compared to the robustness verification.



(a) Constant activation (b) Linear activation (c) Identity activation

Figure 2: Three options we use to eliminate the non-linear activation function and the corresponding neuron. Figure 2a shows the constant function that always outputs the value $f_i^l(\boldsymbol{x}^0)$. Figure 2b shows the linear function defined by preserving the neuron's output for two points $\boldsymbol{x}^0$ and $\boldsymbol{x}^0_{\text{adv}}$. Finally, Figure 2c shows the function that is either the identity or zero depending on the state of the neuron at $\boldsymbol{x}^0$ (since $f_i^l(\boldsymbol{x}^0)$ is positive in this example, we use the identity function).

To achieve the primary goal—faster verification—we must reduce the number of non-linear transformations within the neural network. The non-linear nature of the neural networks and the large number of non-linear activation functions in their hidden layers are the main reasons deep networks can perform well on complex, high-dimensional data by learning meaningful representations. On the other hand, these non-linear transformations make it hard to explain and verify other properties of the network, like robustness. While the exact reasons for that vary depending on the verification approach, a common problem is to formally describe the output sets of the points from the neighborhood $\mathbb{B}_\epsilon(\boldsymbol{x}^0)$. The more non-linear transformation this set undergoes, the more complex the output becomes and the worse the possible approximations of the above-mentioned output sets. See Appendix A.2, where we show in detail how the number of activation functions affects the complexity of the verification task.

3.1 How to prune the neurons

For our task, we remove the non-linear transformations from the network f by replacing them with linear or constant functions and getting a new network g. We use one of the following three options to retain the network's output at $\boldsymbol{x}^0_{\text{adv}}$ and its neighborhood. For each of the pruning methods, we provide proven evidence that they preserve the network's output values locally around $\boldsymbol{x}^0$ laying theoretical foundation for the empirical evaluations.

Constant substitute for the non-linear activation is the simplest option to reduce the complexity of the verification problem $\mathcal{D}$. For each neuron i in layer l, we set its output value to $f_i^l(\boldsymbol{x}^0)$, the same as when the anchor point $\boldsymbol{x}^0$ is propagated through the network. This way we ensure that $g(\boldsymbol{x}^0) = f(\boldsymbol{x}^0)$ remains the same. Figure 2a shows how the activation function changes in the example of a ReLU neuron.

One problem that seems to be unavoidable for this type of pruning is what we call boundary collapse. When we prune too much, the resulting network g becomes not expressive enough to maintain a non-trivial boundary even locally around $\boldsymbol{x}^0$. In this case, the prediction region grows, and the classifier outputs the same label for every point, becoming useless for the robustness assessment of the initial network. However, we develop two different pruning techniques to overcome this problem and ensure that even under severe simplification of the decision boundary, it does not collapse but remains near $\boldsymbol{x}^0$ as follows.

Adversarial preserving linear substitute uses an adversarial attack $\mathcal{A}$ to determine the radius d_{adv} of the relevant neighborhood around $\boldsymbol{x}^0$. Now, we do not only ensure that the network's output after the pruning at the base point $\boldsymbol{x}^0$ remains the same but also at the point $\boldsymbol{x}^0_{\text{adv}}$ that is assigned to a different class. That means we force g to output the same value as f in the center of $\mathbb{B}_{d_{\text{adv}}}(\boldsymbol{x}^0)$ and at one point on its boundary. Therefore, by maintaining g's predicted label at these points, we implicitly enforce that the decision boundary does not move farther away from $\boldsymbol{x}^0$ than $\boldsymbol{x}^0_{\text{adv}}$.

To achieve this adversarial preserving property, we replace the non-linear activation function with a linear function with the same output for the points $\boldsymbol{x}^0$ and $\boldsymbol{x}^0_{\text{adv}}$. Figure 2b shows an example of the new activation function for a ReLU neuron. Note that there is always precisely one linear function that satisfies this property and goes through the points $(f_i^l(\boldsymbol{x}^0), \sigma(f_i^l(\boldsymbol{x}^0)))$ and $(f_i^l(\boldsymbol{x}^0_{\text{adv}}), \sigma(f_i^l(\boldsymbol{x}^0_{\text{adv}})))$ unless $f_i^l(\boldsymbol{x}^0) = f_i^l(\boldsymbol{x}^0_{\text{adv}})$, in which case we set the new activation function to be constant as we do it above for the bounds propagation based pruner.

Activation pattern preserving linear substitute is the third option for pruning the ReLU networks with an even stronger similarity property between f and g. If a neuron is activated at $\boldsymbol{x}^0$, we replace the ReLU function with the identity and otherwise by the constant zero function. Due to the continuity of the ReLU, the new network g has the same output as f not only in a single point but in a neighborhood around $\boldsymbol{x}^0$. That means, there exists $\delta > 0$ such that $f(\tilde{\boldsymbol{x}}^0) = g(\tilde{\boldsymbol{x}}^0)$ for all $\tilde{\boldsymbol{x}}^0 \in B_\delta(\boldsymbol{x}^0)$. This neighborhood is defined as the set of points with the same activation pattern for the pruned neurons as $\boldsymbol{x}^0$. However, the more neurons we prune, the smaller this set becomes. See Section 3.2 for further details, including a proven result that directly links the robustness verification problem of f and g. Figure 2b shows the new activation function for the case $f_i^l(\boldsymbol{x}^0) > 0$.

3.2 Theoretical Guaranties

We prove that the local similarity of two networks allows us to derive robustness guarantees for one network if we prove similar properties for the second one without running the verification algorithm a second time. Here, we address the following so-called ϵ-verification problem. Like in Section 2, given the anchor point $\boldsymbol{x}^0$, network f, predicted label y, and a target label $\hat{y}$, we consider the following problem, which, in case of the ReLU activation, is equivalent to a mixed-integer optimization task.

$$\min_{\tilde{\boldsymbol{x}}^0 \in \mathbb{R}^{n_0}} f_y(\tilde{\boldsymbol{x}}^0) - f_{\hat{y}}(\tilde{\boldsymbol{x}}^0) \text{ s.t. } \|\boldsymbol{x}^0 - \tilde{\boldsymbol{x}}^0\| \leq \epsilon \tag{2}$$

We denote its optimal value by $\mathcal{V}_f(\epsilon)$. Note, that $\mathcal{V}_f(\epsilon) \geq 0$ if and only if $f_y(\tilde{\boldsymbol{x}}^0) \geq f_{\hat{y}}(\tilde{\boldsymbol{x}}^0)$ for all $\tilde{\boldsymbol{x}}^0 \in \mathcal{B}_\epsilon(\boldsymbol{x}^0)$, meaning that the initial label y has a higher score than $\hat{y}$ everywhere on $\mathcal{B}_\epsilon(\boldsymbol{x}^0)$. Algorithm $\mathcal{D}$ solves equation 2 for different ϵ to find the distance to the decision boundary d_{bnd}, which is the largest ϵ, such that $\mathcal{V}_f(\epsilon) \geq 0$.

In the following lemma, we show that the desired property $\mathcal{V}_f(\epsilon) \geq 0$ follows from the corresponding properties of $\mathcal{V}_g(\epsilon)$ if f and g have the same values in a neighborhood around $\boldsymbol{x}^0$ (see Appendix A.1 for the proof).

Lemma 1. *If for some non-negative δ we have $f(\tilde{\boldsymbol{x}}^0) = g(\tilde{\boldsymbol{x}}^0)$ for all $\tilde{\boldsymbol{x}}^0 \in \mathcal{B}_\delta(\boldsymbol{x}^0)$ and*

1. *$\delta \geq \epsilon$, then $\mathcal{V}_f(\epsilon) \geq 0$ if $\mathcal{V}_g(\epsilon) \geq 0$,*
2. *$\delta \leq \epsilon$, then $\mathcal{V}_f(\epsilon) \geq 0$ if $\mathcal{V}_g(\epsilon) \geq 2(L_f + L_g)(\epsilon - \delta)$,*

where L is the Lipschitz constant of the corresponding function on $\mathcal{B}_\epsilon(\boldsymbol{x}^0)$.

In our setting, for the network g being constructed via pruning as described in Section 3, Lemma 1 provides a way to certify the neighborhood $\mathcal{B}_\epsilon(\boldsymbol{x}^0)$ as adversarial free for the network f by solving the optimization problem 2 for a smaller network g. However, we don't use this theoretical result to certify f but instead rely on the empirical comparison of d_{bnd} and $\hat{d}_{bnd}$. First, only the third from the three developed pruning methods, called *activation preserving pruning* (Figure 2c), leads to a new network g that has the same output as f in a ball around $\boldsymbol{x}^0$. The *constant pruning* and *adversarial preserving linear pruning* (Figure 2a and 2b) preserve the output for single points and do not possess this property (except for $\delta = 0$). Second, for the *activation preserving pruning* we need to know δ, such that $f(\tilde{\boldsymbol{x}}^0) = g(\tilde{\boldsymbol{x}}^0)$ for $\tilde{\boldsymbol{x}}^0 \in \mathcal{B}_\delta(\boldsymbol{x}^0)$. In other words, we must find a radius where the pruned neurons' activation pattern does not change. Unfortunately, the task of computing δ, knowing the neurons that we prune, is of the same complexity as the same verification task

formulated by equation 1 itself. More precisely, δ equals the optimal value of the following problem.

$$\min_{\text{pruned } (l,i)} \quad \min_{\tilde{\boldsymbol{x}}^0 \in \mathbb{R}^{n_0}} \|\boldsymbol{x}^0 - \tilde{\boldsymbol{x}}^0\| \text{ s.t.}$$
$$f_i^l(\tilde{\boldsymbol{x}}^0) \leq 0 \text{ if the pruned neuron } (l,i) \text{ is activated (that is } f_i^l(\boldsymbol{x}^0) \geq 0\text{), or}$$
$$f_i^l(\tilde{\boldsymbol{x}}^0) \geq 0 \text{ otherwise}$$

Compared to the initial verification problem equation 1, it has the same objective function and constraints of the same type. Therefore, we do not use Lemma 1 to derive the robustness certificates for the initial network f, as solving the intermediate task of computing δ is not feasible in this framework. Nevertheless, this result proves an underlying theoretical connection between the robustness properties of the initial and pruned networks and provides a starting point for future research in this intriguing direction.

3.3 How to choose the neurons to prune

Bounds propagation techniques, as defined in Section 2, are widely used in verification literature (see Section 5). The most straightforward way to utilize them for the ReLU networks is to detect the neurons that do not change their state for the inputs from $\mathbb{B}_\epsilon(\boldsymbol{x}^0)$. If $\bar{\boldsymbol{a}}_i^l \leq 0$, the input to the neuron i from layer l is always negative or zero, and the neuron remains deactivated such that the activation function returns zero for all relevant inputs. Also, if we get the case $\underline{\boldsymbol{a}}_i^l \geq 0$, this neuron is never deactivated and effectively acts as the identity function. By pruning these stable neurons, we get the new smaller network g with precisely the same output $g(\tilde{\boldsymbol{x}}^0) = f(\tilde{\boldsymbol{x}}^0)$ for all $\tilde{\boldsymbol{x}}^0 \in \mathbb{B}_\epsilon(\boldsymbol{x}^0)$.

Unfortunately, this does not work for the not piece-wise linear activation functions like the sigmoid. Moreover, in this procedure, the only parameter that controls the number of the removed non-linear transformations is ϵ, which defines the size of $\mathbb{B}_\epsilon(\boldsymbol{x}^0)$. The smaller the input region around $\boldsymbol{x}^0$, the tighter are the bounds $\underline{\boldsymbol{a}}_i^l, \bar{\boldsymbol{a}}_i^l$, such that less of the neurons remain in the undecided state $\underline{\boldsymbol{a}}_i^l \leq 0 \leq \bar{\boldsymbol{a}}_i^l$. However, we use this parameter to specify the region we consider for pruning. When ϵ is chosen too small to remove more neurons, we risk that the decision boundary around $\boldsymbol{x}^0$ will lie entirely outside of $\mathbb{B}_\epsilon(\boldsymbol{x}^0)$, meaning that the whole set will be assigned to the same class as $\boldsymbol{x}^0$. This is not desirable since the computation of the bounds in $\mathcal{B}$ does not account for the behavior of the network at the points outside of the given ϵ-neighborhood around $\boldsymbol{x}^0$, which, in this case, does determine the network's robustness properties. That means we would miss essential information about the network when computing the bounds $\underline{\boldsymbol{a}}_i^l, \bar{\boldsymbol{a}}_i^l$ in the first place.

Instead, we first set $\epsilon = d_{\text{adv}}$ by running an attack $\mathcal{A}$. This way, we ensure that $\mathbb{B}_\epsilon(\boldsymbol{x}^0)$ is not too small and contains the decision boundary between the points with different predicted classes $\boldsymbol{x}^0$ and $\boldsymbol{x}^0_{\text{adv}}$ (both are points from $\mathbb{B}_\epsilon(\boldsymbol{x}^0)$). Given a user-defined ratio $\gamma \in (0,1)$ of the neurons to be pruned, we run the bounds propagation algorithm $\mathcal{B}$ and remove the neurons with the smallest difference $\bar{\boldsymbol{a}}_i^l - \underline{\boldsymbol{a}}_i^l$, that we use as a measure of the neuron's variability given the input perturbation budget d_{adv}. That is, we prune the neurons with the least variable output, considering the inputs from $\mathbb{B}_{d_{\text{adv}}}(\boldsymbol{x}^0)$.

Finally, we observe that the neurons of the first layer have $\bar{\boldsymbol{a}}_i^1$ and $\underline{\boldsymbol{a}}_i^1$ with the smallest gap due to the magnification during bounds estimation when $\bar{\boldsymbol{a}}_i^l$ and $\underline{\boldsymbol{a}}_i^l$ are propagated in $\mathcal{B}$ and become looser. Therefore, to prevent the elimination of the neurons exclusively from the first layers, we apply the pruning iteratively by removing a small portion at a time and then recomputing the pre-activation bounds. This way, each time, we prune neurons with the most tight bounds and distribute the removed neurons more evenly throughout the layers. We present the described method in Algorithm 1.

By applying our pruning methods, we achieve the following. 1. **We preserve local behavior** of the network around $\boldsymbol{x}^0$ by pruning the least uncertain neurons, while we don't take into account the points outside of $\mathbb{B}_{d_{\text{adv}}}(\boldsymbol{x}^0)$. 2. **We preserve the value of the network either at $\boldsymbol{x}^0$, at $\boldsymbol{x}^0$ and $\boldsymbol{x}^0_{\text{adv}}$, or even in a neighborhood of $\boldsymbol{x}^0$**, by construction of g as described in Section 3.1. 3. We have a **dedicated parameter γ to set the desired amount of the pruned neurons**.

Algorithm 1: Bounds propagation based iterative pruning

Input: Algorithms $\mathcal{A}$, $\mathcal{B}$, network f, point $\boldsymbol{x} \in \mathbb{R}^{n_0}$, $\gamma \in (0, 1)$ and $\gamma_0 \ll \gamma$.
Output: Network with at most $(1-\gamma)(n_1 + \cdots + n_L)$ ReLU activations.
Compute $d_{\text{adv}} = \mathcal{A}(f)$.
Compute $\bar{\boldsymbol{a}}^l, \underline{\boldsymbol{a}}^l$ using $\mathcal{B}$ with $\epsilon = d_{\text{adv}}$, initialize $k = 1$.
repeat
  Compute the threshold δ as the γ_0-quantile of the values $\bar{\boldsymbol{a}}_i^l - \underline{\boldsymbol{a}}_i^l$ for $l \in \{2, \ldots, L\}$ and $i \in \{1, \ldots, n_l\}$.
  for $l = 2, \ldots, L$ **do**
    Define $I^l \in \{\text{True}, \text{False}\}^{n_l}$ as the indicator vector of the remaining neurons: $I_i^l = 1_{\bar{\boldsymbol{a}}_i^l - \underline{\boldsymbol{a}}_i^l > \delta}$
    For $i \in \{1, \ldots, n_l\}$ where I_i^l is False replace ReLU by a linear function according to Section 3.1.
  end for
  Update $k = k + 1$.
until $(1-\gamma_0)^k \geq 1 - \gamma$
Return: Network with the new activation layers.

4 Experiments

Setup We conduct the experiments using convolutional networks with ReLU activations publicly provided by Li et al. (2023) that are pretrained on the MNIST (LeCun, 1998) and CIFAR10 (Krizhevsky, 2009) datasets. The networks have 166406 (for MNIST) and 214918 (for CIFAR10) parameters and consist of two convolutional layers with 16 and 32 channels respectively, followed by two dense layers with 2028 and 100 neurons. The models are trained either normally or using adversarial training. The following perturbation magnitudes are used by Li et al. (2023) to generate adversarial examples during adversarial training: 0.1 or 0.3 for MNIST, 0.008 or 0.031 for CIFAR10. We evaluate the verification time of all approaches on a machine with a single NVIDIA GTX 1080 Ti GPU with 11 GB memory. For the algorithms $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{D}$ (see Section 2), we use the following popular approaches correspondingly (all available within our implementation based on the unified toolbox by Li et al. (2023)): projected gradient descent (PGD) attack, FastLinIBP interval bound propagation (combination of the methods by Weng et al. (2018) and Gowal et al. (2018)), exact verification by Tjeng et al. (2019) based on mixed integer linear programming (MILP). Unless we mention otherwise, all third-party methods are used with the default parameters by Li et al. (2023).

Procedure The goal is to get a good estimate of d_{bnd} without running $\mathcal{D}$ on the full network. We apply the three pruning methods (Section 3.1) with varying magnitudes for each network. To choose the neurons to prune, we use the bounds propagation-based approach described in Section 3.3 (Algorithm 1) with $\gamma \in \{0.9, 0.99\}$. Note, that the pruning step is independent of the downstream task solved for the pruned model. The same holds for the key insight we verify in our work: the proposed pruning procedure **preserves the network's local boundary** around the anchor point.

Before and after the pruning, we run $\mathcal{D}$ on 10% of the test samples. We use a particular complete verification approach in our experiments, but the core property of the pruning method (i.e., preserving the boundary) does not depend on it. Furthermore, for other complete verification approaches, the resulting d_{bnd} and $\hat{d}_{\text{bnd}}$ must be the same since the verification approaches are complete and solve the same task. Thus, the difference in the computed radii does not depend on choosing a particular complete verification approach. For all the pruned networks, we compute the mean absolute difference $|\hat{d}_{\text{bnd}} - d_{\text{bnd}}|/d_{\text{bnd}}$. The goal is to construct an estimate close to d_{bnd} and do it faster than $\mathcal{D}$ when applied to the initial model f. However, due to the complexity of $\mathcal{D}$, the exact verification often terminates without reaching the optimal point of the MILP optimization problem due to the maximum runtime constraint for a single optimization run. Therefore, to compute the difference in the distance to the decision boundaries fairly, we consider only the points where the algorithm $\mathcal{D}$ does compute the true $\hat{d}_{\text{bnd}}$ and d_{bnd} (see Appendix A.2 for more details).

The primary instance of the MILP-based verification approach by Tjeng et al. (2019) is the mixed-integer optimization problem that must be solved for each sample, target label, and the currently verified radius. It is applied within a binary search to find d_{bnd} as the largest radius where a successful robustness verification is possible. We compare the average runtime of the solver needed to solve this task before and after pruning. However, we have to take into account the fact that the larger the verified radius ϵ is, the more complex the corresponding task becomes as the set of admissible points $\mathbb{B}_\epsilon(\boldsymbol{x}^0)$ becomes larger. As the pruned models have fewer non-linear functions and fewer binary variables in the MILP formulation, we verify larger radii for these models. Therefore, we restrict the set of the solver runs used to compute the average runtime to those that evaluate the robustness of the pruned models for the radii from the same range as for the unpruned models. Thus, we ensure that the computed average runtime represents the solver runs solving the verification task of the same complexity with respect to ϵ. We provide more details about this issue in Appendix A.2.

Table 1: Comparison of d_{bnd} and optimization runtime before and after the pruning.

Network Dataset | Trn	Pruning Type | γ	Avg rel d_{bnd} diff		pruning div. by #runs		Avg time (s) opt. run with pruning		opt. run w/o pruning	
mnist | clean	con | 0.9	0.39	± 0.063	0.02	± 0.01	0.73	± 0.56	33.28	± 26.14
mnist | clean	con | 0.99	0.39	± 0.063	0.04	± 0.02	0.39	± 0.22	33.28	± 26.14
mnist | clean	lin | 0.9	0.0	± 0.0	0.04	± 0.03	31.57	± 24.16	33.28	± 26.14
mnist | clean	lin | 0.99	0.0054	± 0.0062	0.06	± 0.03	16.91	± 17.95	33.28	± 26.14
mnist | clean	act | 0.9	0.016	± 0.011	0.04	± 0.03	31.99	± 24.49	33.28	± 26.14
mnist | clean	act | 0.99	0.019	± 0.01	0.06	± 0.04	17.19	± 18.64	33.28	± 26.14
mnist | adv1	con | 0.9	0.19	± 0.061	0.03	± 0.02	10.91	± 19.90	34.01	± 25.84
mnist | adv1	con | 0.99	0.72	± 0.76	0.08	± 0.05	0.79	± 0.55	34.01	± 25.84
mnist | adv1	lin | 0.9	0.0	± 0.0	0.05	± 0.04	33.78	± 25.62	34.01	± 25.84
mnist | adv1	lin | 0.99	0.0068	± 0.012	0.07	± 0.04	9.73	± 14.12	34.01	± 25.84
mnist | adv1	act | 0.9	0.0067	± 0.0058	0.03	± 0.02	30.93	± 26.11	34.01	± 25.84
mnist | adv1	act | 0.99	0.01	± 0.01	0.05	± 0.02	3.75	± 5.91	34.01	± 25.84
mnist | adv2	con | 0.9	0.061		0.04	± 0.03	20.51	± 26.21	33.10	± 25.96
mnist | adv2	con | 0.99	0.3		0.08	± 0.06	0.68	± 0.56	33.10	± 25.96
mnist | adv2	lin | 0.9	0.0		0.06	± 0.05	37.64	± 24.63	33.10	± 25.96
mnist | adv2	lin | 0.99	0.031		0.06	± 0.06	7.38	± 11.24	33.10	± 25.96
mnist | adv2	act | 0.9	0.01		0.03	± 0.02	31.57	± 26.17	33.10	± 25.96
mnist | adv2	act | 0.99	0.031		0.04	± 0.02	2.17	± 1.94	33.10	± 25.96
cifar10 | clean	con | 0.9	2.1	± 0.83	0.02	± 0.01	1.35	± 1.52	60.12	± 52.64
cifar10 | clean	con | 0.99	5.5	± 5.7	0.03	± 0.02	0.24	± 0.26	60.12	± 52.64
cifar10 | clean	lin | 0.9	0.0098	± 0.014	0.02	± 0.01	37.22	± 48.50	60.12	± 52.64
cifar10 | clean	lin | 0.99	0.0098	± 0.014	0.04	± 0.03	9.20	± 20.13	60.12	± 52.64
cifar10 | clean	act | 0.9	0.0098	± 0.014	0.02	± 0.02	39.01	± 48.82	60.12	± 52.64
cifar10 | clean	act | 0.99	0.0098	± 0.014	0.04	± 0.03	11.01	± 21.63	60.12	± 52.64
cifar10 | adv1	con | 0.9	2.8	± 0.21	0.01	± 0.01	1.44	± 5.23	63.00	± 54.41
cifar10 | adv1	con | 0.99	47	± 69	0.04	± 0.02	0.62	± 0.33	63.00	± 54.41
cifar10 | adv1	lin | 0.9	0.019	± 0.032	0.03	± 0.02	47.08	± 50.26	63.00	± 54.41
cifar10 | adv1	lin | 0.99	0.019	± 0.032	0.06	± 0.04	19.27	± 31.93	63.00	± 54.41
cifar10 | adv1	act | 0.9	0.0	± 0.0	0.03	± 0.02	44.74	± 49.35	63.00	± 54.41
cifar10 | adv1	act | 0.99	0.0	± 0.0	0.04	± 0.02	14.89	± 29.26	63.00	± 54.41
cifar10 | adv2	con | 0.9	1.5	± 0.43	0.02	± 0.01	10.77	± 29.97	59.76	± 55.19
cifar10 | adv2	con | 0.99	3.3	± 2.6	0.04	± 0.02	0.69	± 0.41	59.76	± 55.19
cifar10 | adv2	lin | 0.9	0.013	± 0.025	0.04	± 0.04	49.03	± 51.70	59.76	± 55.19
cifar10 | adv2	lin | 0.99	0.017	± 0.026	0.06	± 0.05	22.78	± 38.50	59.76	± 55.19
cifar10 | adv2	act | 0.9	0.0018	± 0.0055	0.03	± 0.02	46.50	± 50.80	59.76	± 55.19
cifar10 | adv2	act | 0.99	0.0033	± 0.0071	0.04	± 0.01	13.76	± 30.00	59.76	± 55.19

Results Table 1 shows the results of our experiments. Each row represents an evaluation of an initial and pruned network pair. Training methods *clean*, *adv1*, and *adv2* refer to the standard and adversarial training procedures with different attack budgets, as mentioned above. We encode the pruning approach by one of the three options *constant*, *linear* or *activation* described in Section 3.1 and Figures 2a-2c as well as the sparsity level γ. Column *Avg rel* d_{bnd} *diff* contains the mean difference of the computed distance to the decision boundary without reaching the time limit. Column *Avg. time (s) pruning div. by #runs* shows the time needed to obtain the pruned model divided by the number of optimization runs (for different target labels and verified radii) for this model, averaged over all anchor points. Columns *Avg time with pruning* and *Avg time w/o pruning* show the average time needed to solve a single verification MILP after and before pruning for similar ϵ as we described above. In addition to these metrics, we provide the corresponding standard deviations after the $\pm$ sign (the standard deviation of *Avg bnd diff* for the model *mnist / adv2* are missing since $\mathcal{D}$ terminates without exceeding the runtime limit only for a single datapoint).

Results clearly show that the proposed method achieves a consistent speedup of the verification procedure. When 99% of the ReLU functions are removed, the average time to solve a single atomic mixed-integer task is 2 to 5 times shorter compared to the initial model f. The constant pruning achieves the most speedup as it allows not only to reduce the number of the binary variables that encode the state of the ReLU but the continuous variables corresponding to the pruned neurons as well (see Appendix A.2).

Note, that the average runtime overhead per optimization run from pruning (column *Avg. time (s) pruning div. by #runs*) is smaller than 0.1s compared to the average MILP solution time of several seconds for all considered datasets and networks. It shows that the pruning step applies to the models without significant runtime overhead compared to the complete verification procedure.

Looking at the distance difference to the decision boundary before and after the pruning, we report that the adversarial and activation pattern preserving linear ReLU-substitutes result in a local behavior of g similar to f. Due to their theoretical properties, these pruning methods keep the decision boundary around $\boldsymbol{x}^0$, such that the average relative absolute difference between $\hat{d}_{\text{bnd}}$ and d_{bnd} stays at around 1%. Thus, we empirically justify the usage of the smaller substitute models constructed by considering the initial network's local properties for the robustness verification task. Contrary to these two pruning methods, the *constant* pruning does not lead to any useful approximations of d_{bnd} as the relative difference between $\hat{d}_{\text{bnd}}$ and d_{bnd} is at least 1.5 for CIFAR10 models and at least 0.19 for MNIST except the configuration *mnist|adv2|con|0.9*, where the relative difference is 0.061. Overall, the adversarial preserving pruner provides an excellent tool for practitioners not interested in the exact verification of their network but need a faster way to estimate the distance to the nearest adversarial point.

5 Related Work

Regarding robustness verification, two prominent families of methods are used in practice: complete (or exact) and incomplete. **Complete verification** techniques verify exactly whether an adversarial example exists inside the given perturbation budget. The theoretical formulation of this direction uses the MILP framework, for which multiple solvers are available based on either satisfiability modulo theory (SMT) (Katz et al., 2017) or integer programming methods (Tjeng et al., 2019; Dutta et al., 2017).

Incomplete verification techniques relax the constraints on the activations of each layer in the neural network. An example is the work by Wong & Kolter (2018), where the authors replace ReLU activations with a convex substitute and, to increase the method's efficiency further, use the dual formulation of the obtained relaxed linear programming (LP) problem. Ehlers (2017) uses a similar approximation of the ReLU. However, they do not optimize over the dual LP and solve the problem using SMT. Following this direction, Weng et al. (2018); Zhang et al. (2018a) approximate the ReLU activation functions and propagate the possible bounds of the input to the final layer. Other approaches use semi-definite programming (Raghunathan et al., 2018a;b), Lipschitz constant (Weng et al., 2018; Hein & Andriushchenko, 2017; Zhang et al., 2019; Tsuzuku et al., 2018) and abstraction of the network (Gehr et al., 2018). Similar to our work, Croce et al. (2019) look at the local area around the base point and, in particular, utilize the fact that (in the case of ReLU activations) it consists of regions where the network act as a linear function. Besides their incomplete verification method, the authors propose a robust training loss.

All incomplete verification approaches provide a lower bound on the distance to the decision boundary, which is strongly dependent on the tightness of the used relaxation and how each algorithm computes the bounds on the pre-activation values in each layer. Li et al. (2023) provide a detailed overview of the verification methods, including a proposed taxonomy and a publicly available verification toolbox.

Pruning methods allow for minimizing the models' size at only a small cost of their accuracy. They eliminate weights, neurons, or layers, such that the output changes as little as possible. Hoefler et al. (2021) divide the pruning methods into two categories: **data independent pruning** and **data dependent pruning**. The first group consists of pruning methods that do not explicitly use the training data. These methods consider the weights and neurons of the network after training and heuristically find a maximal subset of them that has the minimal impact on the final accuracy (Li et al., 2016; Han et al., 2015; Frankle & Carbin, 2019). Shumailov et al. (2019) consider the transferability of adversarial examples between the pruned (Han et al., 2015) and initial model. Their findings are connected to our work since we also deal with the transferability of the robustness properties to the network after pruning. However, the authors suggest that adversarial examples are interchangeable only at low sparsity levels, which, unfortunately, limits the potential gain in computational cost.

Data dependent pruning methods consider the effect of the training data on the network's output when deciding which neurons or weights to prune (Lee et al., 2019; Wang et al., 2020; Evci et al., 2020). Specifically, these methods consider the influence of the training data on either the output, activations, or a network's derivatives and eliminate a network's extraneous parts without using its weights and biases directly. This idea does not apply well to our scenario, as the training data cannot be used for pruning while preserving the local decision boundary. However, these methods are successful in preserving the classification accuracy.

Finally, another method to create a small network with similar robustness properties as the initial large network is to **train it from scratch**. For example, using the initial network's output to guide the training process. Papernot et al. (2016) use this approach to create a substitute network for attacking the initial model in a black-box environment (that means without access to its gradients). They specifically train the substitute to mimic the initial network on carefully selected data points, iteratively picked by going towards the decision boundary using the gradient information of the substitute network. This method seems very promising, as they also look at the problem by creating a network with a similar decision boundary to the initial one. However, we abstain from performing training in our use case since we must do it for each of the considered base points (unlike Papernot et al. (2016), who train a single substitute network).

Pruning and robust training are combined by several papers to achieve multiple goals at once: train small networks with high clean accuracy that are resilient against adversarial attacks. Gui et al. (2019); Sehwag et al. (2019); Ye et al. (2019) demonstrate the attainment of empirical adversarial robustness through a pruning heuristic based on the lowest weight magnitude and alternating direction method of multipliers pruning framework (Zhang et al., 2018b). In contrast, HYDRA (Sehwag et al., 2020) takes a distinct direction by allowing the robust training objective itself to determine the connections for pruning. This direction of research is different compared to our setting due to the following two major reasons. First, again, the mentioned methods achieve certain global properties of the network while we work with a local neighborhood around a given point. Second, other methods thrive to train a network from scratch or to modify an existing one to get a new network that solves a certain task and has desirable properties. For us, the pruned network is just a tool to evaluate the robustness of the initial network and we do not use it for any other purpose.

Thus our work is the first to use local pruning techniques to preserve *local* similarity and use it for neural network robustness evaluation.

6 Conclusion

Successful sustainable integration of machine learning methods into areas relevant to the industry and society is a challenging task. Especially after deep learning based technologies have proven to be able to tackle complex classification problems, we have to be cautious. We should not rely on the demonstrated excellent performance only.

The presented verification method contributes to the general acceptance of deep learning models. We develop a novel framework to estimate the robustness of a deep classifier around a given base point. We apply the available MILP-based method the exact robustness verification to a specifically constructed local substitute network. We observe that the decision boundary of the initial network far away from the base point does not affect its robustness against adversarial perturbations in the local neighborhood and develop three pruning techniques that preserve the local properties of the network while reducing its overall complexity. This idea allows for a more efficient application of the verification method $\mathcal{D}$ on the pruned models. At the same time, the resulting metrics are reasonable estimates for the robustness of the initial network because of the similarity in the local behavior. We conduct the experiments on the MNIST and CIFAR10 datasets and convolutional networks. The results show that, in particular, the adversarial and activation pattern preserving pruners (see Section 3.3) approximate d_{bnd} very well (around 1% average absolute relative difference, when at least 90% of activations are removed) with a significant speedup compared to $\mathcal{D}$ applied on the initial model. In summary, the proposed technique allows practitioners interested in faster inexact robustness assessment to get a reliable evaluation of the robustness properties.

Future work Katz et al. (2017) show that the ϵ-verification task and thus also computing the distance to the decision boundary is NP-complete. Therefore, we do not set the goal to generally make complete verification computationally tractable, as it is impossible due to the complexity of the underlying optimization task. Instead, we believe that providing reliable estimates of the robustness of the network is a very promising direction.

The success of the presented approach indicates an intriguing direction for future work toward larger networks and datasets. Also, conducting the experiments with different algorithms $\mathcal{D}$ should verify that the speedup we get is agnostic to the applied verification methods. Furthermore, using the result from Lemma 1, we think about deriving additional theoretical guarantees about the difference of d_{bnd} and $\hat{d}_{\text{bnd}}$.

References

Francesco Croce, Maksym Andriushchenko, and Matthias Hein. Provable robustness of relu networks via maximization of linear regions. In Kamalika Chaudhuri and Masashi Sugiyama (eds.), *Proceedings of the Twenty-Second International Conference on Artificial Intelligence and Statistics*, volume 89 of *Proceedings of Machine Learning Research*, pp. 2057–2066. PMLR, 16–18 Apr 2019. URL `https://proceedings.mlr.press/v89/croce19a.html`.

Souradeep Dutta, Susmit Jha, Sriram Sanakaranarayanan, and Ashish Tiwari. Output range analysis for deep neural networks, 2017. Preprint at `https://arxiv.org/abs/1709.09130`.

Ruediger Ehlers. Formal verification of piece-wise linear feed-forward neural networks. In *Automated Technology for Verification and Analysis: 15th International Symposium, ATVA 2017, Pune, India, October 3–6, 2017, Proceedings 15*, pp. 269–286. Springer, 2017.

Utku Evci, Trevor Gale, Jacob Menick, Pablo Samuel Castro, and Erich Elsen. Rigging the lottery: Making all tickets winners. In Hal Daumé III and Aarti Singh (eds.), *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 2943–2952. PMLR, 13–18 Jul 2020. URL `https://proceedings.mlr.press/v119/evci20a.html`.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=rJl-b3RcF7`.

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 *IEEE symposium on security and privacy (SP)*, pp. 3–18. IEEE, 2018.

Sven Gowal, Krishnamurthy Dvijotham, Robert Stanforth, Rudy Bunel, Chongli Qin, Jonathan Uesato, Relja Arandjelovic, Timothy Mann, and Pushmeet Kohli. On the effectiveness of interval bound propagation for training verifiably robust models, 2018. Preprint at `https://arxiv.org/abs/1810.12715`.

Shupeng Gui, Haotao Wang, Haichuan Yang, Chen Yu, Zhangyang Wang, and Ji Liu. Model compression with adversarial robustness: A unified optimization framework. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/2ca65f58e35d9ad45bf7f3ae5cfd08f1-Paper.pdf.

Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding, 2015. Preprint at https://arxiv.org/abs/1510.00149.

Matthias Hein and Maksym Andriushchenko. Formal guarantees on the robustness of a classifier against adversarial manipulation. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/e077e1a544eec4f0307cf5c3c721d944-Paper.pdf.

Torsten Hoefler, Dan Alistarh, Tal Ben-Nun, Nikoli Dryden, and Alexandra Peste. Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks. *The Journal of Machine Learning Research*, 22(1):10882–11005, 2021.

Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. In *Computer Aided Verification: 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I 30*, pp. 97–117. Springer, 2017.

Alex Krizhevsky. Learning multiple layers of features from tiny images, 2009. figshare https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf.

Yann LeCun. The mnist database of handwritten digits, 1998. figshare http://yann.lecun.com/exdb/mnist/.

Namhoon Lee, Thalaiyasingam Ajanthan, and Philip Torr. SNIP: Single-shot network pruning based on connection sensitivity. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=B1VZqjAcYX.

Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets, 2016. Preprint at https://arxiv.org/abs/1608.08710.

Linyi Li, Tao Xie, and Bo Li. Sok: Certified robustness for deep neural networks. In *2023 IEEE symposium on security and privacy (SP)*, pp. 1289–1310. IEEE, 2023.

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, 2016. Preprint at https://arxiv.org/abs/1602.02697.

Aditi Raghunathan, Jacob Steinhardt, and Percy Liang. Certified defenses against adversarial examples, 2018a. Preprint at https://arxiv.org/abs/1801.09344.

Aditi Raghunathan, Jacob Steinhardt, and Percy S Liang. Semidefinite relaxations for certifying robustness to adversarial examples. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018b. URL https://proceedings.neurips.cc/paper_files/paper/2018/file/29c0605a3bab4229e46723f89cf59d83-Paper.pdf.

Vikash Sehwag, Shiqi Wang, Prateek Mittal, and Suman Jana. Towards compact and robust deep neural networks, 2019. Preprint at https://arxiv.org/abs/1906.06110.

Vikash Sehwag, Shiqi Wang, Prateek Mittal, and Suman Jana. Hydra: Pruning adversarially robust neural networks. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 19655–19666. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/e3a72c791a69f87b05ea7742e04430ed-Paper.pdf.

Ilia Shumailov, Yiren Zhao, Robert Mullins, and Ross Anderson. To compress or not to compress: Understanding the interactions between adversarial attacks and neural network compression. *Proceedings of Machine Learning and Systems*, 1:230–240, 2019.

Vincent Tjeng, Kai Xiao, and Russ Tedrake. Evaluating robustness of neural networks with mixed integer programming, 2019. Preprint at `https://arxiv.org/abs/1711.07356`.

Yusuke Tsuzuku, Issei Sato, and Masashi Sugiyama. Lipschitz-margin training: Scalable certification of perturbation invariance for deep neural networks. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL `https://proceedings.neurips.cc/paper_files/paper/2018/file/485843481a7edacbfce101ecb1e4d2a8-Paper.pdf`.

Chaoqi Wang, Guodong Zhang, and Roger Baker Grosse. Picking winning tickets before training by preserving gradient flow, 2020. Preprint at `https://arxiv.org/abs/2002.07376`.

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 Jennifer Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 5276–5285. PMLR, 10–15 Jul 2018. URL `https://proceedings.mlr.press/v80/weng18a.html`.

Eric Wong and Zico Kolter. Provable defenses against adversarial examples via the convex outer adversarial polytope. In Jennifer Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 5286–5295. PMLR, 10–15 Jul 2018. URL `https://proceedings.mlr.press/v80/wong18a.html`.

Shaokai Ye, Kaidi Xu, Sijia Liu, Hao Cheng, Jan-Henrik Lambrechts, Huan Zhang, Aojun Zhou, Kaisheng Ma, Yanzhi Wang, and Xue Lin. Adversarial robustness vs. model compression, or both? In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 111–120, 2019.

Huan Zhang, Tsui-Wei Weng, Pin-Yu Chen, Cho-Jui Hsieh, and Luca Daniel. Efficient neural network robustness certification with general activation functions. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018a. URL `https://proceedings.neurips.cc/paper_files/paper/2018/file/d04863f100d59b3eb688a11f95b0ae60-Paper.pdf`.

Huan Zhang, Pengchuan Zhang, and Cho-Jui Hsieh. Recurjac: An efficient recursive algorithm for bounding jacobian matrix of neural networks and its applications. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 5757–5764, 2019. doi: 10.1609/aaai.v33i01.33015757.

Tianyun Zhang, Shaokai Ye, Kaiqi Zhang, Jian Tang, Wujie Wen, Makan Fardad, and Yanzhi Wang. A systematic dnn weight pruning framework using alternating direction method of multipliers. In *Proceedings of the European conference on computer vision (ECCV)*, pp. 184–199, 2018b.

A Appendix

A.1 Proofs

Lemma 1. *If for some non-negative δ we have $f(\tilde{\boldsymbol{x}}^0) = g(\tilde{\boldsymbol{x}}^0)$ for all $\tilde{\boldsymbol{x}}^0 \in \mathcal{B}_\delta(\boldsymbol{x}^0)$ and*

1. *$\delta \geq \epsilon$, then $\mathcal{V}_f(\epsilon) \geq 0$ if $\mathcal{V}_g(\epsilon) \geq 0$,*
2. *$\delta \leq \epsilon$, then $\mathcal{V}_f(\epsilon) \geq 0$ if $\mathcal{V}_g(\epsilon) \geq 2(L_f + L_g)(\epsilon - \delta)$,*

where L is the Lipschitz constant of the corresponding function on $\mathcal{B}_\epsilon(\boldsymbol{x}^0)$.

Proof. 1 follows directly from the fact that $f = g$ on the set $\mathcal{B}_\delta(\boldsymbol{x}^0)$ that includes the admissible set of the optimization task 2, which is $\mathcal{B}_\epsilon(\boldsymbol{x}^0)$. To prove 2, assume $\boldsymbol{z}^0 \in \mathcal{B}_\epsilon(\boldsymbol{x}^0)$ and let $\tilde{\boldsymbol{z}}^0 = \frac{\epsilon}{\delta}\boldsymbol{z}^0$ be the projection of $\boldsymbol{z}^0$ on $\mathcal{B}_\delta(\boldsymbol{x}^0)$, then

$$\begin{aligned}
f_y(\boldsymbol{z}^0) - f_{\hat{y}}(\boldsymbol{z}^0) = & \underbrace{f_y(\boldsymbol{z}^0) - f_y(\tilde{\boldsymbol{z}}^0)}_{\geq -L_f\|\boldsymbol{z}^0-\tilde{\boldsymbol{z}}^0\|} + \underbrace{f_y(\tilde{\boldsymbol{z}}^0) - g_y(\tilde{\boldsymbol{z}}^0)}_{=0} + \underbrace{g_y(\tilde{\boldsymbol{z}}^0) - g_y(\boldsymbol{z}^0)}_{\geq -L_g\|\boldsymbol{z}^0-\tilde{\boldsymbol{z}}^0\|} + \\
& + \underbrace{g_y(\boldsymbol{z}^0) - g_{\hat{y}}(\boldsymbol{z}^0)}_{\geq \mathcal{V}_g(\epsilon)} + \\
& + \underbrace{g_{\hat{y}}(\boldsymbol{z}^0) - g_{\hat{y}}(\tilde{\boldsymbol{z}}^0)}_{\geq -L_g\|\boldsymbol{z}^0-\tilde{\boldsymbol{z}}^0\|} + \underbrace{g_{\hat{y}}(\tilde{\boldsymbol{z}}^0) - f_{\hat{y}}(\tilde{\boldsymbol{z}}^0)}_{=0} + \underbrace{f_{\hat{y}}(\tilde{\boldsymbol{z}}^0) - f_{\hat{y}}(\boldsymbol{z}^0)}_{\geq -L_f\|\boldsymbol{z}^0-\tilde{\boldsymbol{z}}^0\|} \\
& \geq \mathcal{V}_g(\epsilon) - 2(L_f + L_g)(\epsilon - \delta)
\end{aligned}$$

That means $\mathcal{V}_g(\epsilon) \geq 2(L_f + L_g)(\epsilon - \delta)$ indeed implies $f_y(\tilde{\boldsymbol{x}}^0) \geq f_{\hat{y}}(\tilde{\boldsymbol{x}}^0)$ for all $\tilde{\boldsymbol{x}}^0 \in \mathcal{B}_\epsilon(\boldsymbol{x}^0)$, or in other words $\mathcal{V}_f(\epsilon) \geq 0$. □

A.2 Runtime and Complexity

To understand what affects the complexity of the problem 2, we reformulate it as follows (here for a single target label $\tilde{y}$).

$$\min_{\tilde{\boldsymbol{x}},\tilde{\boldsymbol{z}}} (e_{\tilde{y}} - e_y)^T \left(\boldsymbol{W}^L \tilde{\boldsymbol{x}}^{L-1} + \boldsymbol{b}^L\right), \text{ s.t. } \|\boldsymbol{x}^0 - \tilde{\boldsymbol{x}}^0\| \leq \epsilon \text{ and for } l \in [L-1] \tag{3}$$

$$\tilde{\boldsymbol{x}}^l \leq \boldsymbol{W}^l \tilde{\boldsymbol{x}}^{l-1} + \boldsymbol{b}^l - (\boldsymbol{1} - \boldsymbol{z}^l)\underline{\boldsymbol{a}}^l \tag{4}$$

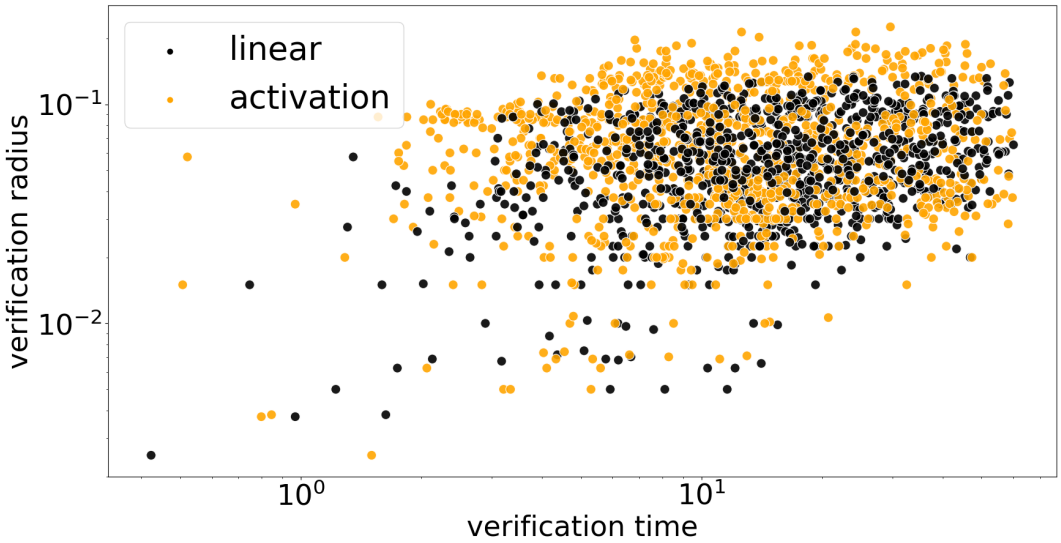
$$\tilde{\boldsymbol{x}}^l \geq \boldsymbol{W}^l \tilde{\boldsymbol{x}}^{l-1} + \boldsymbol{b}^l \tag{5}$$

$$\tilde{\boldsymbol{x}}^l \leq \boldsymbol{z}^l \bar{\boldsymbol{a}}^l, \ \tilde{\boldsymbol{x}}^l \geq 0 \tag{6}$$

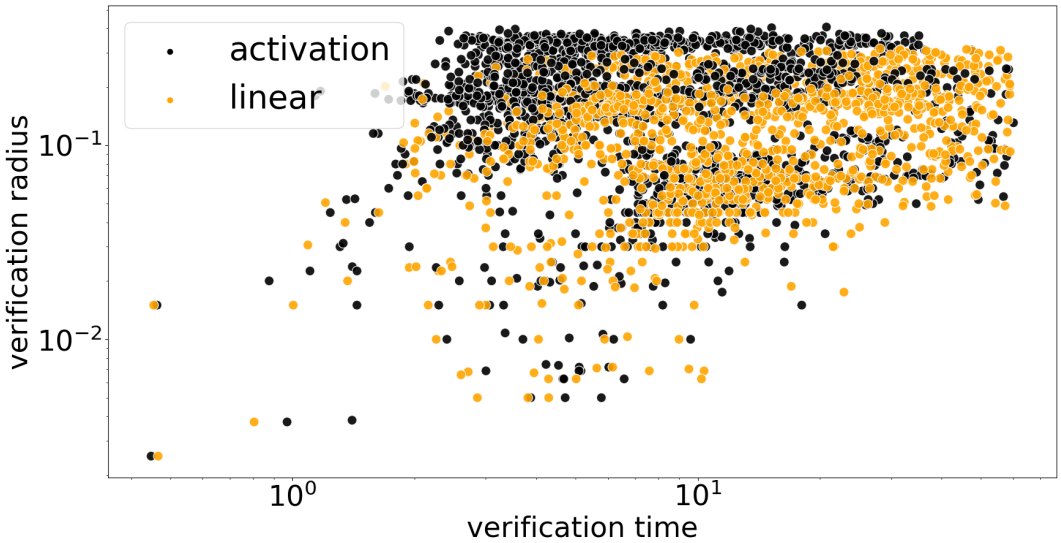
$$x_i^l \in \mathbb{R}, z_i^l \in \{0, 1\}$$

Note, that once we prune a neuron (l, i) as described in Section 3, the corresponding binary variable $\boldsymbol{z}_i^l$ disappears from the formulation together with the constraints 4–6 and we get a simple linear constraint $\tilde{\boldsymbol{x}}_i^l = \boldsymbol{W}_{i,:}^l \tilde{\boldsymbol{x}}^{l-1} + \boldsymbol{b}_i^l$ instead. Here, $\boldsymbol{W}_{i,:}^l$ and $\boldsymbol{b}_i^l$ define our new linear transformation instead of the non-linear ReLU activation. By pruning, for example, 99% of the hidden neurons, we reduce the number of integer variables in equation 1 by the same amount. The standard approach to solve a MILP task is branch-and-bound, where we divide the whole set of the admissible points by relaxing the integrality constraints (in our case $\boldsymbol{z}_i^l \in \{0, 1\}$) one by one and bound the optimal value on the resulting sub-domains. Fewer binary variables mean less branching and a faster problem solution.

On the other hand, smaller ϵ also contributes to a faster solution. Now, instead of affecting the ReLU-constraints 4–6, ϵ directly controls the size of $\mathcal{B}_\epsilon(\boldsymbol{x}^0)$ via the constraint $\|\boldsymbol{x}^0 - \tilde{\boldsymbol{x}}^0\| \leq \epsilon$. Again, smaller ϵ results in a smaller admissible set that gets explored faster by the solver. To demonstrate this correlation, in Figures 3a and 3b, we show the time of each optimization run solving problem 2 that completed within the time limit of 60 seconds on the x-axis. The ϵ that was verified is shown on the y-axis. Each point represents the optimization tasks solved a network pruned either using *linear adversarial preserving* or *activation pattern preserving* method indicated by the color of the points. For Figure 3a we use the sparsity parameter $\gamma = 0.9$ and for Figure 3b $\gamma = 0.99$. That means within one plot, the only parameter affecting the optimization task's complexity is ϵ.



(a) $\gamma = 0.9$



(b) $\gamma = 0.99$

Figure 3: Runtime and verified ϵ for the optimization tasks.