Abstract

Neural networks have demonstrated considerable success in a wide variety of real-world problems. However, the presence of adversarial examples - slightly perturbed inputs that are misclassified with high confidence - limits our ability to guarantee performance for these networks in safety-critical applications. We demonstrate that, for networks that are piecewise affine (for example, deep networks with ReLU and maxpool units), proving no adversarial example exists - or finding the closest example if one does exist - can be naturally formulated as solving a mixed integer program. Solves for a fully-connected MNIST classifier with three hidden layers can be completed an order of magnitude faster than those of the best existing approach. To address the concern that adversarial examples are irrelevant because pixel-wise attacks are unlikely to happen in natural images, we search for adversaries over a natural class of perturbations written as convolutions with an adversarial blurring kernel. When searching over blurred images, we find that as opposed to pixelwise attacks, some misclassifications are impossible. Even more interestingly, a small fraction of input images are provably robust to blurs: every blurred version of the input is classified with the same, correct label.

1. Introduction

Neural networks have seen significant success on a range of problems. In particular, within computer vision, they represent the state-of-the-art for classification of images [8, 17] and object localization in images [15]. However, recent work has shown that many of the same networks that perform so well on these datasets are vulnerable to adversarial examples: input $x'$ that is generated by slightly perturbing some regular input $x$ in a way that leads to radically different output. In the context of image classification, the perturbed input is often indistinguishable from the original input, but can lead to misclassifications into any target category chosen by the adversary [16]. This phenomenon makes it difficult to provide performance guarantees for safety-critical applications such as autonomous cars.

One approach to defending against adversarial examples is to train networks to be more robust to them [14]. Another approach is to attempt to detect adversarial examples by identifying some properties on which they differ from normal input [1, 9]. However, both specially-trained networks and detection approaches are often soon defeated by new targeted attacks that iteratively construct adversarial examples [4] or artificially reduce the magnitude of the input to the softmax function [5].

Carlini et al. introduce the idea of using ground-truth adversarial examples [3] to design future-proof defenses. For a given network $f(\cdot)$, the ground-truth adversarial example for some input $x$ is defined as the closest $x'$ such that $x$ and $x'$ are categorized differently. The algorithm relies on iteratively calling a Satisfiability Modulo Theories (SMT) solver [11] to determine whether there exists an adversarial example within a selected distance $\delta$. Carlini et al. argue that ground-truth adversarial examples can be used to estimate the robustness of a network, and successfully demonstrate that these examples can be used to compare the effectiveness of defense techniques.

We develop a faster way to find these ground-truth adversarial examples by formulating the optimization problem as a mixed-integer programming (MIP) model. This formulation allows us to directly determine the closest $x'$ (or prove that no such $x'$ exists) in a single solve. More importantly, explicitly specifying an objective (namely, finding the closest adversarial example) provides additional information that can inform and guide the branch-and-bound search used to solve the MIP. These two factors combine to provide an improvement in solve times of one to two orders of magnitude over the median solve time of two to three days for the SMT solver-based approach (Guy Katz, personal communication, October 13, 2017).

Many people dismiss adversarial examples as irrelevant because pixel-wise attacks are unlikely to happen in natural images. We address this objection by searching over a natural class of input perturbations written as convolutions with
an adversarial blurring kernel. We discover an interesting phenomenon: while there still are misclassified versions of the original image that are virtually indistinguishable from the original, for the vast majority of input, there is at least one category that no blurred version of the image is classified in. Furthermore, a small fraction of the test data is provably robust: every blurred version of the image is classified in the same, correct category. This suggests that we can train networks to be provably robust to restricted families of perturbations that go beyond the $L_1$ norm bounded perturbations that Kolter et al. have demonstrated success on [12].

Our contributions can be summarized as follows: (i) we present an algorithm to identify ground-truth adversarial examples that is significantly faster than the state-of-the-art; (ii) we apply this algorithm to blurring perturbations and discover that we can prove robustness for a significant subset of pairs of input images and target categories. (iii) due to improved solved times, we are able to fully quantify the adversarial attacks on a large sample of the training set, and in some cases confirm that the network is fully robust to an entire class of attacks.

2. Preliminaries

2.1. Notation

Consider a neural network $N = f(\cdot; \theta) : \mathbb{R}^m \rightarrow \mathbb{R}^n$ that is parameterized by a (fixed) vector of weights $\theta$. The output of the final layer contains a single node for each target class the network is designed to predict.

Most of the existing literature considers the problem of finding an adversarial example where we allow every pixel of the original image to be modified independently (what we call an additive adversarial example). For a given test input $x$, this corresponds to finding an $\epsilon$ such that $f(x+\epsilon) \in \mathcal{Y}_{adv}$, where $\mathcal{Y}_{adv}$ denotes the set of adversarial target outputs$^1$. Finding the ground-truth additive adversarial example simply corresponds to solving the optimization problem

$$\min_{\epsilon} \|\epsilon\|_d$$

subject to

$$f(x+\epsilon) \in \mathcal{Y}_{adv}$$

$$0 \leq x_i + \epsilon_i \leq 1 \quad \forall \ i = 1, 2, \ldots, m$$

$$x' = x + \epsilon$$

More generally, we can parameterize the search space of adversarial images to a particular family of perturbations $g(\cdot, \gamma) : \mathbb{R}^m \times \mathbb{R}^p \rightarrow \mathbb{R}^n$, where $\gamma$ is the parameter. Additive perturbations form one such family. However, an objection to the additive family is that it can transform any image into any other image; a threat model that may be unrealistic. We can address this concern by selecting $g(\cdot)$ to better reflect what perturbations are possible in natural images.

The search for the ground-truth adversarial example over the family $g(\cdot)$ then corresponds to a natural generalization of the optimization problem in Equations 1-4.

$$\min_\gamma \|x' - x\|_d$$

subject to

$$f(x') \in \mathcal{Y}_{adv}$$

$$0 \leq x_i' \leq 1 \quad \forall \ i = 1, 2, \ldots, m$$

$$x' = g(x, \gamma)$$

Finally, for conciseness, we define the notion of perturbing into a target class: we say that an input image has been perturbed into a target class if the perturbed image is classified by the neural network in that class.

3. Model Setup

Translating the optimization problem in Equations 5-8 to an MIP model boils down to efficiently expressing the constraints between the input $x$ and the output $f(x)$, as well as the constraints in Equations 6-8.

To understand how to generate the constraints that link the input and output of each layer, let us consider an example $k$-layer feed-forward network $N_0 = f(\cdot)$ that uses ReLU:

$$f(x'; \theta) = \hat{z}_{k-1}$$

$$\hat{z}_i = W_i z_{i-1} + b_i, \quad i = 1, 2, \ldots, k-1$$

$$z_i = \max(\hat{z}_i, 0), \quad i = 1, 2, \ldots, k-2$$

$$z_0 = x'$$

Decision Variables. The MIP model consists of four different groups of decision variables: (i) continuous variables corresponding to the (perturbed) input $x'$ to the network; (ii) continuous variables corresponding to the activations for each of the layers of the neural network; (iii) continuous variables corresponding to the parameter $\gamma$; (iv) binary variables corresponding to the choice of the active set for piecewise linear constraints (e.g. ReLU, maxpool). In the case of $N_0$, we introduce one continuous variable for each of the elements of $z_1, \hat{z}_i$ for all $i$. In addition, we introduce a binary variable for each ReLU as discussed in Section 3.1.1.

Constraints. The MIP model consists of four different groups of constraints on decision variables: (i) constraints on the output ensuring that it is in the target set $\mathcal{Y}_{adv}$ (Equation 6); (ii) constraints on the perturbed input to ensure that its value stays within bounds (Equation 7); (iii) constraints between the parameter $\gamma$ and the perturbed input $x'$ (Equation 8); (iv) constraints between the value of the input and output of the network. The constraint between the value of

---

$^1$For example, if the target is to have the perturbed input be classified with category $c'$, we would choose $\mathcal{Y}_{adv} = \{y \mid y \in \mathbb{R}^n, y_{c'} \geq y \forall j \in [1, n]\}$. 

---
the input and output of the network is enforced by successively adding constraints between the input and output of each layer of the network. Again, using $N_0$ as an example, linear relationships between decision variables (10, 12) can be represented as a single linear equality constraint in the model. However, each of the piecewise linear input-output relationships between decision variables (11) are only expressible as a set of linear constraints.

We provide an example of all the decision variables and constraints for a simple net in the Appendix.

### 3.1. Expressing Piecewise Linear Relationships

We demonstrate how to express the input-output relationship for both rectified linear and max-pooling units. Bounds on the input decision variables are necessary for these expressions to work. We will first proceed by assuming that some bounds exist, but will show how to get tight bounds in Section 3.2.

#### 3.1.1 Expressing ReLU in an MIP Model

Let $y = \text{max}(x, 0)$, and $l \leq x \leq u$. If $u \leq 0$, we have $z \equiv 0$; similarly, if $l \geq 0$, we have $z \equiv x$. Otherwise, we introduce an indicator decision variable $a = I_{x \geq 0}$. The ReLU constraint is then equivalent to the following set of linear constraints:

\begin{align*}
    y & \leq x - l(1 - a) \quad (13) \\
    y & \geq x \quad (14) \\
    y & \leq u \cdot a \\
    y & \geq 0 \\
    a & \in \{0, 1\} \quad (17)
\end{align*}

To see that this formulation works, consider what happens when $a = 0$. The constraints in Equation 15 and 16 are binding, and together imply that $y = 0$. The other two constraints are not binding, since Equation 14 is no stricter than Equation 16 when $x < 0$, while Equation 13 is no stricter than Equation 15 since $x - l \geq 0$. We thus have $a = 0 \implies y = 0$. A similar line of reasoning allows us to conclude that $a = 1 \implies y = x$.\footnote{We note that this formulation for rectified linearity is the best possible given the bounds on $x$. This is the case since relaxing the integrality constraint on $a$ leads to $(x, y)$ being restricted to an area that is the convex hull of $y = \text{max}(x, 0)$.}

#### 3.1.2 Expressing Max-Pooling in an MIP Model

Let $y = \max(x_1, x_2, \ldots, x_m)$, and $l_i \leq x_i \leq u_i$. Define $l_{\text{max}} = \max(l_1, l_2, \ldots, l_m)$. For efficiency, we first eliminate all $x_i$ where $u_i \leq l_{\text{max}}$, since we know that $y \geq l_{\text{max}}$ (and thus we would have $y \geq u_i \geq x_i$). Assume without loss of generality that we do not need to eliminate any of $x_i$. We introduce an indicator decision variable $a_i$ for each of our input variables, where $a_i = 1 \implies y = x_i$. Furthermore, we define $u_{\text{max},-i} = \max_{j \neq i}(u_j)$. The max-pooling constraint is then equivalent to the following set of linear constraints:

\begin{align*}
    y & \leq x_i + (1 - a_i)(u_{\text{max},-i} - l_i) \quad \forall i \quad (18) \\
    y & \geq x_i \quad \forall i \quad (19) \\
    \sum_{i=1}^{m} a_i & = 1 \quad (20) \\
    a_i & \in \{0, 1\} \quad (21)
\end{align*}

To see that this formulation works, we first note that Equation 20 ensures that exactly one of the $a_i$ is 1.\footnote{If more than one $x_i$ is equal to the maximum value, we will simply select one; $y = x_i$ does not necessarily imply $a_i = 1$.} It thus suffices to consider the value of $a_i$ for a single variable.

When $a_i = 1$, Equations 18 and 19 are binding, and together imply that $y = x_i$. We thus have $a_i = 1 \implies y = x_i$.

When $a_i = 0$, we simply need to show that the constraints involving $x_i$ are never binding regardless of the values of $x_1, x_2, \ldots, x_m$. Equation 19 is not binding since $a_i = 0$ implies $x_i$ is not the (unique) maximum value. Furthermore, we have chosen the coefficient of $a_i$ such that Equation 18 is not binding, since $x_i + u_{\text{max},-i} - l_i \geq u_{\text{max},-i} \geq y$. This completes our proof.

### 3.2. Getting Tight Bounds on Decision Variables

In practice, obtaining the tightest possible bounds on each decision variable that is an input to a non-linear relationship is crucial to ensure that the translated MIP model has a tight convex relaxation. This improves the numerical conditioning of the resulting optimization problem, and renders it more tractable for the solver. Simply obtaining tighter bounds on these decision variables can provide improvements of several orders of magnitude for solve times.

Since the perturbed input is always bounded between 0 and 1 (Equation 7), it is always possible to determine some bounds on each of the decision variables corresponding to the activation of some layer of the network. If the family of perturbations we choose and the input image we are attempting to perturb has the right structure, we may be able to improve on these bounds further. We discuss this observation in more detail in Section 5.2.

Interval arithmetic\cite{13} allows us to derive reasonable bounds on variables efficiently. The approach involves calculating bounds for the output of each layer of the network based on bounds to its input, beginning at the input layer and proceeding until we reach the layer containing the variable of interest.
While the bounds derived from interval arithmetic are sufficient for smaller networks, even tighter upper (lower) bounds can be found by solving the partially-constructed MIP model with the objective set to maximizing (minimizing) the value of the variable of interest. The bounds derived from solving the MIP are often much tighter than the bounds derived from interval arithmetic, leading to a significant speed-up in the main algorithm that finds adversarial examples. Again, we provide an example of this process of tightening bounds in the Appendix.

3.3. Solving the Model

When solving the MIP model, we always have available (1) a candidate ground-truth adversarial example at a distance of 
\[ \delta_+ = \|x' - x\|_\infty, \]
which is simply the closest we have found during our search so far, and (2) a certificate proving that there exists no solution within a distance \( \delta_- \) of \( x \). The solver begins by relaxing all the integer constraints in the MIP, discovering a lower bound \( \delta_- \) that holds for the relaxed LP. This bound is precisely the one found when solving the LP in [12]. The bounds is progressively tightened via a branch-and-bound search, and the solve terminates when the MIP gap \( \delta_+ / \delta_- - 1 \) falls below a selected threshold. In practice, we can set this threshold such that the gap \( \delta_+ / \delta_- - 1 \) is exactly 0, meaning that the candidate ground-truth adversarial example we have found is provably optimal.

4. Evaluation

Choice of Neural Network. We evaluate the performance of our mixed-integer verification framework on two neural networks. Both networks are relatively small, but achieve a high classification accuracy relative to the number of parameters they use. \( N_1 \) is a network with two fully-connected hidden layers containing 40 nodes and 20 nodes respectively, and a final output layer with 10 nodes. The classification accuracy on the MNIST test set is 96.95%. \( N_2 \) is a network with three fully-connected hidden layers containing 24 nodes each, and a final output layer with 10 nodes. The classification accuracy on the test set is 97.02%. \( N_2 \) matches the size and reported accuracy of the network found in [3].

Input Image. For each network, we select the input to be perturbed from the subset of images where the predicted category matches the provided label.

Distance Metric. The MIP framework supports three distance metrics for the distance in Equation 5: \( L_1 \) and \( L_\infty \) (as an MILP) and \( L_2 \) (as a MIQP). We only present results for the \( L_1 \) norm, but performance for the \( L_2 \) and \( L_\infty \) norms are comparable.

Family of Perturbations. We consider two families of perturbations \( g(\cdot) \): standard additive perturbations in Section 5.1, with \( g(x) = x + \epsilon \), and blurring perturbations in Section 5.2, with \( g(x) = x + h \), where \( h \) is a \( 5 \times 5 \) convolution kernel with \( 0 \leq h_{i,j} \leq 1, \sum h_{i,j} = 1 \). These blurring perturbations are meant to mimic blur from camera shake [7].

Each combination of neural network \( f(\cdot) \), input \( x \), perturbation family \( g(\cdot) \), and target category (which determines the region \( Y_{adv} \)) corresponds to a unique instance of the optimization problem in Equations 5-8.

We construct the MIP models corresponding to the optimization problems in Julia [2] using JuMP [6], with the solve carried out by the commercial solver Gurobi 7.0.2 [10] on a desktop with a Intel Core i7-6700 CPU (with 4 cores) clocked at 3.40GHz.

5. Experimental Results

For each image considered, we attempted to find the minimum perturbation that would cause the image to be classified as an instance of each of the nine possible target categories.\(^4\)

5.1. Additive Perturbations

We split the MNIST test set by the true label of the images, selecting the first image in each of the ten label categories.\(^5\) The perturbed images for \( N_1 \) are shown in Figure 2, and statistics on solve times and distances for both networks are presented in Tables 1 and 2. Solve times are significantly lower for \( N_1 \) when compared to \( N_2 \). This is likely to be the result of a combination of the smaller number of hidden units (60 for \( N_1 \) vs. 72 for \( N_2 \)) and the increased depth of the network (2 hidden layers for \( N_1 \) vs. 3 for \( N_2 \)). Nevertheless, for \( N_2 \), even the worst-case solve time is significantly faster than the median solve time of two to three days reported for the SMT-based solver (Guy Katz, personal communication, October 13, 2017) for a network of the same size and architecture. As discussed in the introduction, we believe that this improvement in solve times is a result of our ability to specify an explicit objective to guide the branch-and-bound search.

While the worst-case solve time is still high, Figure 1 demonstrates that the candidate ground-truth adversarial examples found early in the solve are usually close to optimal. Let \( \delta^* \) denote the distance between the true ground-truth adversarial example and the original image. In practice, the gap between \( \delta_+ \) and \( \delta^* \) decreases rapidly even if the solve takes a long time to complete. If we are simply interested in obtaining an adversarial example with a 'good-enough' value of \( \delta_+ \), we can simply set a time limit for the search.

\(^4\) We exclude the case where the target category matches the actual label, since the minimum perturbation would simply be zero.

\(^5\) For both networks we evaluate, the indexes for these images in the test set are (one-indexed) 4, 3, 2, 31, 5, 9, 12, 1, 62, and 8 for labels 0 to 9. Neither \( N_1 \) nor \( N_2 \) classifies the first ‘3’ (index 19) correctly, and we thus ignore this image.
Figure 1: Solve progress for worst-case solves for network $N_2$. We select the ten solves that take the longest time of the ninety experiments we ran for additive perturbation. Each line corresponds to the progress of an MIP model solve for an input image with the specified index in the test set, and the target category for the perturbation. The multiplicative gap $\mu$ is defined as $\mu = \delta_+ / \delta^- - 1$, and reaches 0 when the solve is completed. Better candidate adversarial examples are found relatively infrequently, but significantly improve on the multiplicative gap when they are found, leading to the large step-decreases in the value of the gap. For our samples, the multiplicative gap falls below 0.1 within 300s for all samples even when the slowest solve takes more than 30,000s to complete.

and retrieve the candidate ground-truth adversarial example.

5.2. Blurring Perturbations

As with the additive perturbations in Section 5.1, we split the MNIST test set by the true label of the images. However, since solve times for blurring perturbations are significantly lower, we were able to run experiments on the first fifty images in each of the ten label categories.

In contrast to additive perturbations of unbounded norm, restricted families of perturbations may not always be able to perturb every image to every target label. We introduce the notion of achievability: we call an input image-target category pair $(x, c')$ achievable under blurring for a given network $N = f(\cdot)$ if there is some blurred version $x'$ of the input that is classified as an instance of $c'$ by $N$.

While many blurred images that are misclassified are virtually indistinguishable from the original images (see, for instance, the ground-truth adversarial examples for target labels 4 and 5 for the input image with label 9 in Figure 3a), not all image-category pairs are achievable.

A sample of the perturbed images for $N_1$ are shown in Figure 3, and statistics on solve times and distances for both networks are presented in Tables 3 and 4. The average number of image-category pairs that are achievable under blurring is less than half: 0.426 for $N_1$, and 0.466 for $N_2$.

Our solver is able to find the ground-truth adversarial ex-
Neural Net | Count | Solve Time / s Mean | Min | Median | Max |
--- | --- | --- | --- | --- | --- |
$N_1$ | 90 | 161.5 | 2.6 | 17.2 | 3638.6 |
$N_2$ | 90 | 3081.2 | 10.3 | 1115.9 | 39293.8 |

Table 1: Solve times for finding ground-truth additive adversarial examples, minimizing over the $L_1$ norm for the perturbation. We perturb a total of ten images - one for each true label - and attempt to cause the perturbed images to be classified in each of the nine target categories.

Neural Net | Count | $L_1$ distance Mean | Min | Median | Max |
--- | --- | --- | --- | --- | --- |
$N_1$ | 90 | 7.362 | 0.397 | 6.37 | 38.384 |
$N_2$ | 90 | 7.741 | 1.126 | 7.518 | 16.436 |

Table 2: Minimum $L_1$ distances for ground-truth additive adversarial examples. The distance is the sum of the absolute value of the difference between the brightness of the pixel in the original and perturbed image. Pixel values range from 0 (white) to 1 (black), and the image has a total of 784 pixels.

ample far more quickly when the search space is restricted to only blurred versions of the original image. Comparing Tables 1 and 3, we find that the solve times for blurring adversarial examples are one to two orders of magnitude faster than those for additive adversarial examples. For instance, the median solve time to find the ground-truth adversarial example for $N_2$ falls from 1120s to just 4.90s.

Two main factors contribute to this reduction in solve times. Firstly, the number of parameters in the search space is reduced. For the MNIST dataset, additive perturbations have $784 = 28 \times 28$ parameters, while the blurring kernel only has $24 = 5 \times 5 - 1$. More importantly, we are able to exploit the structure of the family of perturbations and input image to obtain better lower and upper bounds on the perturbed input. For example, since our blurring kernel is of size $5 \times 5$, the maximum value a pixel in the perturbed image is simply the maximum value among all pixels in the $5 \times 5$ window centered at that pixel. The same reasoning holds true for the minimum value in the perturbed image. The importance of having tight bounds on decision variables is discussed in greater detail in Section 3.2.

5.2.1 Achievability of Misclassifications

To better understand the distribution of achievable pairs, we begin by aggregating the data on achievability based on the true category of the image and the target category we are attempting to perturb the image into. Let $\alpha(x, c'; N)$ be an indicator variable indicating whether $(x, c')$ is an image-

![Images with minimal blurring perturbations. The original images can be found along the main diagonal. Gray boxes represent image-category pairs not achievable under blurring.](image1)

![Difference between perturbed and original images. A red pixel indicates that that pixel was darkened, while a blue pixel indicates that the pixel was brightened. This view of the data emphasizes how small many of the ground-truth perturbations are, even though we are restricting ourselves to blurring perturbations. Gray boxes represent image-category pairs not achievable under blurring.](image2)

Figure 3: Ground-truth blurring adversarial examples for $N_1$. A single image - the same one as in Figure 2 - was selected from the MNIST test set for each true label category. For each image, we applied our MIP solver to find the minimum blurring perturbation required to cause $N_1$ to classify the image as an instance of each of the possible target categories.
Table 3: Solve times for finding ground-truth blurring adversarial examples, minimizing over the $L_1$ norm for the perturbation. We group the times by whether the target category was achievable (Y) or not achievable (N) for the given input. We perturb a total of 500 images - 50 for each true label - and attempt to cause the perturbed images to be classified in each of the nine target categories.

| Neural Net | Count | Mean | Min | Med. | Max |
|------------|-------|------|-----|------|-----|
| N          |       |      |     |      |     |
| N          | 2585  | 1.62 | 0.01| 1.70 | 16.73|
| Y          | 1915  | 2.17 | 0.32| 1.82 | 13.77|
| T          | 4500  | 1.72 | 0.01| 1.78 | 16.73|
| N          | 2400  | 17.55| 0.01| 3.31 | 2748.67|
| Y          | 2100  | 16.55| 0.92| 7.16 | 533.37|
| T          | 4500  | 17.08| 0.01| 4.90 | 2748.67|

Table 4: Minimum $L_1$ distances for ground-truth blurring adversarial examples where the target category was achievable (Y). This is the same distance metric used in Table 2. Note that the median $L_1$ distance is much larger than that of the additive adversarial examples, which is what we would expect. However, the minimum distance is still very small given that there are a total of 784 pixels that can change.

| Neural Net | Count | $L_1$ distance | Mean | Min | Med. | Max |
|------------|-------|----------------|------|-----|------|-----|
| N          |       |                |      |     |      |     |
| N          | 1915  | 66.60          | 0.09 | 64.91| 195.36|
| Y          | 2100  | 70.88          | 0.31 | 67.64| 190.07|

category pair achievable under blurring for the network $N$. For each pair of categories $c, c'$, we can evaluate an empirical adversary success rate:

$$r_{c,c'}(N) = \frac{\sum_{l(x) = c} a(x, c'; N)}{\sum_{l(x) = c} 1}$$

where $l(x)$ is the true label for $x$. The success rates for $N_1$ for each pair of categories are visualized in Figure 4, and is summarized by true category and target category in Tables 5 and 6.

One key observation is that the adversary success rate is not necessarily symmetric when we interchange the true label and target label. For instance, all of the images with true label ‘1’ in our sample have a perturbed version that is classified as an ‘8’, but only one of the fifty images with true label ‘8’ has a perturbed version that is classified as a ‘1’.

Another key observation is that there are significant differences between labels in terms of the false positive and false negatives rates. The rows of Figure 4 show that there are some labels (such as ‘2’ and ‘8’) for which the fraction of false negatives generated is significantly below average.

Figure 4: Success rate of blurring perturbations for $N_1$. Each cell of the heatmap summarizes the results of 50 experiments. The adversary success rate is the fraction of input images with a particular true label that have a blurred version that is classified with a selected target label. Note the significant heterogeneity in success rates - the success rates cover the full range from 0 to 1.

| target label | adversary success rate | true label | adversary success rate |
|--------------|------------------------|------------|------------------------|
| 0            | 0.285                  | 0          | 0.374                  |
| 1            | 0.052                  | 1          | 0.774                  |
| 2            | 0.552                  | 2          | 0.292                  |
| 3            | 0.233                  | 3          | 0.493                  |
| 4            | 0.203                  | 4          | 0.420                  |
| 5            | 0.247                  | 5          | 0.531                  |
| 6            | 0.747                  | 6          | 0.320                  |
| 7            | 0.626                  | 7          | 0.403                  |
| 8            | 0.879                  | 8          | 0.277                  |
| 9            | 0.536                  | 9          | 0.386                  |

Table 5: Adversary success rate for $N_1$, aggregated by target label of perturbation. Entries correspond to the average value of a column in Figure 4.

Table 6: Adversary success rate for $N_1$, aggregated by true label of image. Entries correspond to the average value of a row in Figure 4.

Similarly, the columns of Figure 4 show that there are some labels (such as ‘1’ and ‘3’) for which the fraction of false positives generated is significantly below average. While the ultimate goal would be to eliminate all adversarial examples, an intermediate step could be to tune the network to a point that optimizes, based on the cost context, the trade-
off between the false negative rate and the false positive rate for each label. For example, in the context of a self-driving car, designers may choose to accept a small amount of false positives in detecting pedestrians as long as missed detections are kept at a minimum.

5.2.2 Provable Robustness of Images

Interestingly, we find that there are certain images that are provably robust to blurring for a given network \( N \): every blurred version of the image is classified in the same, correct category. To formalize this, we define the number of achievable incorrect target categories \( n_i(x; N) \) as

\[
n_i(x; N) = \sum_{c' \in C, c' \neq l(x)} a(x, c'; N)
\]

where \( C \) denotes the set of all categories. Provably robust images have \( n_i(x; N) = 0 \), while images that can be perturbed into any incorrect category have \( n_i(x; N) = |C| - 1 \).

The distribution of values of \( n_i \) for \( N_1 \) is shown in Figure 5, and the distribution for \( N_2 \) is similar (albeit with a higher mean). While there is no particular reason to expect that any images would be provably robust, a small fraction of the images are, and they are shown in Figure 6a. The fact that we have provably robust images even though we made no attempt to train the network for robustness bodes well for having provable robustness over a larger subset of test input if we train a network with robustness to blurs as an explicit goal.

6. Conclusion

Having performance guarantees on neural networks would give us greater confidence when applying these networks in safety-critical applications such as autonomous vehicles. Our efficient implementation of the verification algorithm enables the nearest adversarial example to be determined more efficiently, and would allow quicker evaluation of the robustness increase for defenses as done in [3].

In addition, to the best of our knowledge, this is the first work to demonstrate the existence of provably robust inputs for a family of perturbations where every provably robust input will be identified. Quick solves (a median of 4.90s for blurs and 1120s for additive perturbations for a neural network with three hidden layers) also enable us to better understand statistical properties of the test set in relation to a given neural network.

One potentially fruitful direction for future work would be to consider different choices of the family of perturbations applied to the input image. A wide range of families are supported by our MIP formulation, including modifications to the contrast and brightness of the image, thresholding operations on pixel intensities, and unsharp masks.

Figure 5: Distribution of number of achievable categories for blurring perturbations for \( N_1 \) for a sample of 500 input images. The input images in the leftmost bin are provably robust to blurring for \( N_1 \): all blurred versions of the image are classified by \( N_1 \) in the same, correct category. The input images in the rightmost bin exhibit the polar opposite behavior: there is a blurred version of the image that is classified by \( N_1 \) in each of the ten possible categories.

Figure 6: A comparison of provably robust images with other images in the test set that have the same true label. Note that to the human eye, there is no ambiguity surrounding the label for any of these images (other than possibly the ‘8’ on the bottom right).

In addition, there may be opportunities to exploit the structure of the neural network in order to further improve solve times. For example, the solver does not currently take advantage of the fact that the network consists of layers; using this structure to guide the branch-and-bound search could reduce the total number of nodes we need to explore during the search process.

Finally, we may be able to use the results from these robustness tests to train the networks differently, leading to networks that are robust by design.
Acknowledgements

This work was partially supported by Lockheed Martin Corporation under award number RPP2016-002. We would like to thank Robin Deits, Tobia Marcucci, Peter R. Florence, Matthew O’Kelly, Yunzhu Li and Felipe Hernandez for several useful conversations.

References

[1] Osbert Bastani, Yani Ioannou, Leonidas Lampropoulos, Dimitrios Vytiniotis, Aditya Nori, and Antonio Criminisi. Measuring neural net robustness with constraints. In Advances in Neural Information Processing Systems, pages 2613–2621, 2016. 1

[2] Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B Shah. Julia: A fresh approach to numerical computing. SIAM Review, 59(1):65–98, 2017. 4

[3] Nicholas Carlini, Guy Katz, Clark Barrett, and David L Dill. Ground-truth adversarial examples. arXiv preprint arXiv:1709.10207, 2017. 1, 4, 8

[4] Nicholas Carlini and David Wagner. Adversarial examples are not easily detected: Bypassing ten detection methods. arXiv preprint arXiv:1705.07263, 2017. 1

[5] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In Security and Privacy (SP), 2017 IEEE Symposium on, pages 39–57. IEEE, 2017. 1

[6] Iain Dunning, Joey Huchette, and Miles Lubin. JuMP: A modeling language for mathematical optimization. SIAM Review, 59(2):295–320, 2017. 4

[7] Rob Fergus, Barun Singh, Aaron Hertzmann, Sam T Roweis, and William T Freeman. Removing camera shake from a single photograph. In ACM transactions on graphics (TOG), volume 25, pages 787–794. ACM, 2006. 4

[8] Benjamin Graham. Fractional max-pooling. arXiv preprint arXiv:1412.6071, 2014. 1

[9] Shixiang Gu and Luca Rigazio. Towards deep neural network architectures robust to adversarial examples. arXiv preprint arXiv:1412.5068, 2014. 1

[10] Inc. Gurobi Optimization. Gurobi optimizer reference manual, 2017. 4

[11] Guy Katz, Clark Barrett, David Dill, Kyle Julian, and Mykel Kochenderfer. Reluplex: An efficient SMT solver for verifying deep neural networks. arXiv preprint arXiv:1702.01135, 2017. 1

[12] J Zico Kolter and Eric Wong. Provably defensible against adversarial examples via the convex outer adversarial polytope. arXiv preprint arXiv:1711.00851, 2017. 2, 4

[13] Ramon E Moore, R Baker Kearfott, and Michael J Cloud. Introduction to interval analysis. SIAM, 2009. 3

[14] Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. Distillation as a defense to adversarial perturbations against deep neural networks. In Security and Privacy (SP), 2016 IEEE Symposium on, pages 582–597. IEEE, 2016. 1

[15] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. International Journal of Computer Vision (IJCV), 115(3):211–252, 2015. 1

[16] 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. 1

[17] Li Wan, Matthew Zeiler, Sixin Zhang, Yann L Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In Proceedings of the 30th international conference on machine learning (ICML-13), pages 1058–1066, 2013. 1
A. Transcription to a Mixed-Integer Program

We demonstrate how to translate the problem of finding the ground-truth adversarial example to solving an MIP model. The key steps are to: (i) translate any constraints that are not linear in the decision variables to constraints that are linear in the decision variables, and (ii) to specify the objective as a linear (or quadratic) sum of decision variables. For simplicity, we will consider the case of additive adversarial examples.

As we discussed in Section 2, finding the ground-truth ($L_1$ norm) additive adversarial example for a neural net $f(\cdot)$ corresponds to solving the optimization problem

$$
\min_{\epsilon} \|\epsilon\|_1 \tag{22}
$$

subject to

$$
f(x') \in \mathcal{Y}_{\text{adv}} \tag{23}
$$

$$
0 \leq x' \leq 1 \tag{24}
$$

$$
x' = x + \epsilon \tag{25}
$$

where $x$ is some fixed input image. We collate all of the constraints associated with this optimization problem and their MIP counterparts in Table 7.

Equations 24 and 25 are clearly linear in the decision variables $x'$, $\epsilon$. However, Equation 23 is not linear in the decision variables. To see how we can translate a highly non-linear neural network $f(\cdot)$ to a set of linear constraints (with added binary variables), we consider the example of a fully-connected network $f(\cdot; \theta) : \mathbb{R}^m \rightarrow \mathbb{R}^2$ with two hidden layers and an output layer with two output neurons. We have $f(x'; \theta) = \hat{z}_3$, where

$$
\hat{z}_1 = W_1 x' + b_1 \tag{26}
$$

$$
z_1 = \max(\hat{z}_1, 0) \tag{27}
$$

$$
\hat{z}_2 = W_2 z_1 + b_2 \tag{28}
$$

$$
z_2 = \max(\hat{z}_2, 0) \tag{29}
$$

$$
\hat{z}_3 = W_3 z_2 + b_3 \tag{30}
$$

with $W_i, b_i$ denoting the matrix of weights and vector of biases for layer $i$.

We arbitrarily select the first category to be the target category; that is, we want the activation of the first neuron in the output layer to be greater than that of the second neuron. Our choice of adversarial region is thus

$$
\mathcal{Y}_{\text{adv}} = \{ y \mid y \in \mathbb{R}^2, y_1 \geq y_2 \} \tag{31}
$$

We have translated Equation 23 to Equations 26-31. Most of these equations are linear, but Equations 27 and 29 are still not. For these non-linear constraints, we will need to add binary decision variables that act as indicator variables on the value of the activation of a neuron. We denote the binary decision variable associated with the variable $v$ as $a_v$.

| Original Constraints | Linear Constraints (with Integer Variables) |
|----------------------|---------------------------------------------|
| 23 $f(x') \in \mathcal{Y}_{\text{adv}}$ | Converted to Equations 26-31. |
| 24 $0 \leq x' \leq 1$ | |
| 25 $x' = x + \epsilon$ | |
| 26 $\hat{z}_1 = W_1 x' + b_1$ | Solve optimization problem in Equation 32 to obtain bounds $l_{\hat{z}_1} \leq \hat{z}_1 \leq u_{\hat{z}_1}$. |
| 27 $z_1 = \max(\hat{z}_1, 0)$ | $z_1 \leq \hat{z}_1 - l_{\hat{z}_1}(1 - a_{\hat{z}_1})$ |
| | $z_1 \geq \hat{z}_1$ |
| | $z_1 \leq a_{\hat{z}_1} \hat{z}_1$ |
| | $z_1 \geq 0$ |
| | $a_{\hat{z}_1} \in \{0, 1\}$ |
| 28 $\hat{z}_2 = W_2 z_1 + b_2$ | Solve optimization problem in Equation 36 to obtain bounds $l_{\hat{z}_2} \leq \hat{z}_2 \leq u_{\hat{z}_2}$. |
| 29 $z_2 = \max(\hat{z}_2, 0)$ | $z_2 \leq \hat{z}_2 - l_{\hat{z}_2}(1 - a_{\hat{z}_2})$ |
| | $z_2 \geq \hat{z}_2$ |
| | $z_2 \leq a_{\hat{z}_2} \hat{z}_2$ |
| | $z_2 \geq 0$ |
| | $a_{\hat{z}_2} \in \{0, 1\}$ |
| 30 $\hat{z}_3 = W_3 z_2 + b_3$ | |
| 31 $\mathcal{Y}_{\text{adv}} = \{ y \mid y_1 \geq y_2 \}$ | $\hat{z}_3^{(1)} \geq \hat{z}_3^{(2)}$ |
| 22 $\min \|\epsilon\|_1$ | $\min \sum_i \|\epsilon_i\|_1$ s.t. $|\epsilon_i|_+ \geq \epsilon$ |
| | $|\epsilon_i|_+ \geq -\epsilon$ |

Table 7: Translating the original optimization problem to a mixed integer problem for a simple neural net with two hidden layers and two output neurons. Decision variables are vectors and inequalities operate element-wise, and we refer to the $i^{th}$ element of $x$ as $x(i)$. If $f(x') \in \mathcal{Y}_{\text{adv}}$, we would be solving a Mixed Integer Linear Program; if it is a quadratic sum, we would be solving a Mixed Integer Quadratic Program.

$x$ is not a decision variable since it is a fixed input image.

The parameter $\theta$ corresponds to all the weights and biases of the network.
The first non-linear constraint involves the output of the first hidden layer: \( z_1 = \max(\hat{z}_1, 0) \). As discussed in Section 3.1, we need to determine upper and lower bounds on the value of each of the elements of \( \hat{z}_1 \).

To find these bounds, we solve an MIP with constraints 24-26 and the appropriate objective. Consider, for example, the value of the activation of the first neuron in this layer, \( \hat{z}_1^{(1)} \). The lower bound \( l_\hat{z}_1^{(1)} \) is simply the solution to the optimization problem:

\[
\begin{align*}
\min & \quad \hat{z}_1^{(1)} \\
\text{subject to} & \quad x' = x + \epsilon \\
& \quad 0 \leq x' \leq 1 \\
& \quad \hat{z}_1 = W_1 x + b_1
\end{align*}
\]

We solve a separate optimization problem (with objective \( \max \hat{z}_1^{(1)} \)) to determine the upper bound \( u_\hat{z}_1^{(1)} \), and repeat this for each of the other neurons in the layer. Armed with these constraints, we convert the non-linear ReLU constraint in Equation 27 to linear constraints using the result in Section 3.1.

Equation 29 is the next non-linear constraint, and involves the output of the second hidden layer. To find the bounds on \( \hat{z}_2 \), we solve a larger MIP with constraints 24-28. For example, the lower bound \( l_\hat{z}_2^{(1)} \) of the value of the activation of the first neuron in the second hidden layer \( \hat{z}_2^{(1)} \) is

\[
\begin{align*}
\min & \quad \hat{z}_2^{(1)} \\
\text{subject to} & \quad x' = x + \epsilon \\
& \quad 0 \leq x' \leq 1 \\
& \quad \hat{z}_1 = W_1 x' + b_1 \\
& \quad z_1 \leq \hat{z}_1 - l_\hat{z}_1 (1 - a_{\hat{z}_1}) \\
& \quad z_1 \geq \hat{z}_1 \\
& \quad z_1 \geq u_\hat{z}_1 a_{\hat{z}_1} \\
& \quad z_1 \geq 0 \\
& \quad \hat{z}_2 = W_2 z_1 + b_2
\end{align*}
\]

These bounds that we obtain are used in row 7 of Table 7.

The penultimate constraint to impose is the choice of adversarial region. In our case, the constraint on the adversarial region is already linear. Finally, since we are attempting to minimize the \( L_1 \) norm of the perturbation, we need to have a variable corresponding to the absolute value of \( \epsilon \). The absolute value is a piecewise linear function, but we do not need to introduce any additional integer variables. Instead, the decision variable we introduce, \( |\epsilon|_+ \), has a value at least that of the absolute value of \( \epsilon \). We can then minimize over the sum of the elements of \( |\epsilon|_+ \); this works since the optimal value for this new minimization can only be achieved when \( |\epsilon|_+ = |\epsilon| \). This completes our translation process.