Verification of Neural Networks with Mixed Integer Programming

MEEN 689 Convex Optimization for Control Systems
Final Project Report Fall 2019

Venkata Sameer Kumar Betana Bhotla
UIN:728009992

December 2019

1 Motivation

We start exploring this problem by asking why need to verify neural networks? Neural networks trained only to optimize for training accuracy have been shown to be vulnerable to adversarial examples, with small perturbations to input potentially leading to large changes in the 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.

There is now a large body of work proposing defense methods to produce classifiers that are more robust to adversarial examples. However, as long as a defense is evaluated only via attacks that find local optima, we have no guarantee that the defense actually increases the robustness of the classifier produced.

Fortunately, we **can** evaluate robustness to adversarial examples in a principled fashion. One option is to determine (for each test input) the minimum distance to the closest adversarial example, which we call the **minimum adversarial distortion**. The second option is to determine the **adversarial test accuracy**, which is the proportion of the test set for which no bounded perturbation causes a misclassification. An increase in the mean minimum adversarial distortion or in the adversarial test accuracy indicates an improvement in robustness.

Determining the minimum adversarial distortion for some input (or proving that no bounded perturbation of that input causes a misclassification) corresponds to solving an optimization problem. For piecewise-linear neural networks, the optimization problem can be expressed as a mixed-integer linear programming (MILP) problem.

1.1 Introduction

As mentioned above neural networks trained only to optimize for training accuracy have been shown to be vulnerable to adversarial examples: perturbed inputs that are very similar to some regular input but for which the output is radically different [1]. There is now a large body of work proposing defense methods to produce classifiers that are more robust to adversarial examples. However, as long as a defense is evaluated only via heuristic attacks (such as the Fast Gradient Sign Method (FGSM) [2][4], we have no guarantee that the defense actually increases the robustness of the classifier produced. Defense methods thought to be successful when published have often later been found to be vulnerable to a new class of attacks.

For instance, multiple defense methods are defeated in [5] by constructing defense-specific loss functions and in [6] by overcoming obfuscated gradients. Fortunately, we can evaluate robustness to adversarial examples in a precise manner. One option is to determine (for each test input) the minimum distance to the

closest adversarial example, in literature it is called the minimum adversarial distortion [3]. Alternatively, we can determine the adversarial test accuracy [7], which is the proportion of the test set for which no perturbation in some bounded class causes a misclassification. An increase in the mean minimum adversarial distortion or in the adversarial test accuracy indicates an improvement in robustness. (The two measures are related: a solver that can find certificates for bounded perturbations can be used iteratively (in a binary search process) to find minimum distortions.)

This project was inspired from the paper [9], which presents an efficient implementation of a mixed-integer linear programming (MILP) verifier for properties of piecewise-linear feed-forward neural networks. Their tight formulation for non-linearities and novel pre-solve algorithm can be combined to minimize the number of binary variables in the MILP problem and dramatically improve its numerical conditioning. Optimizations in their MILP implementation improve performance by several orders of magnitude when compared to a naive MILP implementation, and are two to three orders of magnitude faster than the state-of-the-art Satisfiability Modulo Theories (SMT) based verifier [8].

1.2 Literature Survey

[9] work relates to other peoples work on verification of piecewise-linear neural networks; [13] provides a good overview of this area. Authors in [9] categorize verification procedures as complete or incomplete. To understand the difference between these two types of procedures, we consider the example of evaluating adversarial accuracy.

As in [12], adversarial polytope is known as the exact set of all final-layer activations that can be achieved by applying a bounded perturbation to the input. Incomplete verifiers reason over an outer approximation of the adversarial polytope. As a result, when using incomplete verifiers, the answer to some queries about the adversarial polytope may not be decidable. In particular, incomplete verifiers can only certify robustness for a fraction of robust input; the status for the remaining input is undetermined. In contrast, complete verifiers reason over the exact adversarial polytope. Given sufficient time, a complete verifier can provide a definite answer to any query about the adversarial polytope. In the context of adversarial accuracy, complete verifiers will obtain a valid adversarial example or a certificate of robustness for every input. When a time limit is set, complete verifiers behave like incomplete verifiers, and resolve only a fraction of queries. However, complete verifiers do allow users to answer a larger fraction of queries by extending the set time limit.

Incomplete verifiers for evaluating network robustness employ a range of techniques, including duality [12], layer-by-layer approximations of the adversarial polytope, discretizing the search space, abstract interpretation, bounding the local Lipschitz constant [10], or bounding the activation of the ReLU with linear functions [10].

Complete verifiers typically employ either MILP solvers [9] [14] or SMT solvers [3]. [9] approach improves upon existing MILP-based approaches with a tighter formulation for non-linearities and a novel presolve algorithm that makes full use of all information available, leading to solve times several orders of magnitude faster than a naively implemented MILP-based approach. When compared to the state-of-the-art SMT-based approach (Reluplex) on the task of finding minimum adversarial distortions, [9]'s verifier is two to three orders of magnitude faster. Crucially, these improvements in performance allow their verifier to verify a network with over 100,000 units—several orders of magnitude larger than the largest MNIST classifier previously verified with a complete verifier.

A complementary line of research to verification is in robust training procedures that train networks designed to be robust to bounded perturbations. Robust training aims to minimize the "worst-case loss" for each example— that is, the maximum loss over all bounded perturbations of that example [12]. Since calculating the exact worst-case loss can be computationally costly, robust training procedures typically minimize an estimate of the worst-case loss: either a lower bound as is the case for adversarial training [2], or an upper bound as is the case for certified training approaches [12]. Complete verifiers such as [9] can augment robust training procedures by resolving the status of input for which heuristic attacks cannot find an adversarial example and incomplete verifiers cannot guarantee robustness, enabling more accurate

comparisons between different training procedures.

2 Problem Formulation

Problem formulation discussed here is based on [9] and [13].

2.1 Problem Statement

Using the notation from [9] we will denote neural network by $f(.;\theta) : \mathbb{R}^m \rightarrow \mathbb{R}^n$ parameterized by a (fixed) vector of weights θ. For a classifier, the output layer has a neuron for each target class.

Verification as solving an MILP: The general problem of verification is to determine whether some property P on the output of a neural network holds for all input in a bounded input domain $\mathcal{C} \subseteq \mathbb{R}^m$. For the verification problem to be expressible as solving an MILP, P must be expressible as the conjunction or disjunction of linear properties $P_{i,j}$ over some set of polyhedra $\mathcal{C}_i$ where $\mathcal{C} = \cup \mathcal{C}_i$.

Here we assume that $f(.)$ must be composed of piecewise-linear layers i.e., we assume that layers with linear transformations like fully-connected, convolution, and average-pooling layers or layers with linear activation function like ReLU or maximum-pooling layers.

2.2 Formulating Robustness Evaluation of Classifiers as MILP

Here we will see how to **evaluate adversarial accuracy** (based on [9]). Let $\mathcal{G}$ denote the region in the input domain corresponding to all allowable perturbations of a particular input x. In general, perturbed inputs must also remain in the domain of valid inputs $\mathcal{X}_{valid}$, example for normalized images with pixel values ranging from 0 to 1, $\mathcal{X}_{valid} = [0,1]^m$. We say that a neural network is robust to perturbations on x if the predicted probability of the true label $\lambda(x)$ exceeds that of every other label for all perturbations:

$$\forall x^{'} \in (\mathcal{G}(x) \cap \mathcal{X}_{valid}) : \text{argmax}_i(f_i(x^{'})) = \lambda(x) \tag{1}$$

Equivalently, the network is robust to perturbations on x if and only if below equation is infeasible for $x^{'}$:

$$(x^{'} \in (\mathcal{G}(x) \cap \mathcal{X}_{valid})) \wedge \left(f_{\lambda(x)}(x^{'}) < max_{\mu \in [1,n]\ \{\lambda(x)\}} f_{\mu}(x^{'})\right) \tag{2}$$

where $f_i(.)$ is the i^{th} output of the network. In the paper they use the following notation: x is robust with respect to the network if $f(.)$ is robust to perturbations on x. If x is not robust, we call any $x^{'}$ satisfying the constraints a valid adversarial example to x. The adversarial accuracy of a network is the fraction of the test set that is robust; the adversarial error is the complement of the adversarial accuracy.

Important thing to note is that As long as $\mathcal{G}(x) \cap \mathcal{X}_{valid}$ can be expressed as the union of a set of polyhedra, the feasibility problem can be expressed as an MILP. Training procedures designed to be robust to l_∞ norm, they can be expressed as set of linear constraints [12][9].

Evaluating Mean Minimum Adversarial Distortion Let $d(.,.)$ denote a distance metric that measures the perceptual similarity between two input images. The minimum adversarial distortion under d for input x with true label $\lambda(x)$ corresponds to the solution to the optimization:

$$\min_{x^{'}} d(x, x^{'}) \tag{3}$$

$$\text{subjected to} \quad \text{argmax}_i(f_i(x^{'})) \neq \lambda(x) \tag{4}$$

$$x^{'} \in \mathcal{X}_{valid} \tag{5}$$

We can target the attack to generate an adversarial example that is classified in one of a set of target labels T by replacing with $argmax_i(f_i(x^{'})) \in T$. The most prevalent distance metrics in the literature for generating adversarial examples are the l_1, l_2, and l_∞ norms [1][2][4]. As explained in the Boyd's book [11] we can express the objective without adding any additional integer variables to the model.

GETBOUNDSFORRELU(x, fs)

$\triangleright$ fs are the procedures to determine bounds, sorted in increasing computational complexity.
$l_{best} = -\infty$; $u_{best} = \infty$ $\triangleright$ initialize best known upper and lower bounds on x
for f in fs: $\triangleright$ carrying out progressive bounds tightening
  do $u = f(x, boundType = upper)$; $u_{best} = \min(u_{best}, u)$
    if $u_{best} \leq 0$ **return** (l_{best}, u_{best}) $\triangleright$ Early return: $x \leq u_{best} \leq 0$; thus $\max(x, 0) \equiv 0$.
    $l = f(x, boundType = lower)$; $l_{best} = \max(l_{best}, l)$
    if $l_{best} \geq 0$ **return** (l_{best}, u_{best}) $\triangleright$ Early return: $x \geq l_{best} \geq 0$; thus $\max(x, 0) \equiv x$
return (l_{best}, u_{best}) $\triangleright$ x could be either positive or negative.

Figure 1: Pseudocode demonstrating how to efficiently determine bounds for the tightest possible formulations for the ReLU

2.3 Formulating Piecewise-Linear Functions as MILP

Tight formulations of the ReLU and maximum functions are critical to good performance of the MILP solver. Here I have provided the formulation based on [9] but for proof please refer the paper [9].

Formulating ReLU Let $y = max(x, 0)$, and $l \leq x \leq u$. We have following three possibilities of ReLU

- If $u \leq 0$ we have $y \equiv 0$. This inactive phase.
- If $l \geq 0$ we have $y \equiv x$. This active phase.
- Otherwise it is known as unstable.

For unstable unit we write it with the help of indicator decision variable $a = \mathbb{1}_{x \geq 0}$. Now we can write ReLU as set of integer and linear constraints

$$(y \leq x - l(1 - a)) \wedge (y \geq x) \wedge (y \leq u.a) \wedge (y \geq 0) \wedge (a \in \{0, 1\}) \tag{6}$$

Note that we can relax the binary constraint on a to express it as a convex formulation.

Formulating the Maximum Function Let $y = max(x_1, x_2, \ldots x_m)$, and $l_i \leq x_i \leq u_i$. Let $l_{max} \triangleq max(l_1, l_2, \ldots, l_m)$. We can eliminate from consideration all x_i where $u_i \leq l_{max}$, since we know that $y \geq l_{max} \geq u_i \geq x_i$. Using an indicator variable a_i for each of our input variables where $a_i = 1 \implies y = x_i$. We define $u_{max,-i} \triangleq max_{j \neq i}(u_j)$. Using all this we can represent max function as a set of linear and integer constraints

$$\bigwedge_{i=1}^{m} ((y \leq x_i + (1 - a_i)(u_{max,-i} - l_i)) \wedge (y \geq x_i)) \wedge (\sum_{i=1}^{m} a_i = 1)(a \in \{0, 1\}) \tag{7}$$

2.4 Progressive Bounds Tightening

In the paper they also discuss how to progressively tighten the bounds. I have not explored this section of the paper in detailed because of the time constraint but I have tried to implement it.

Previously we assumed that we had some element-wise bounds on the inputs to non-linearities. In practice, we have to carry out a presolve step to determine these bounds because

- it is important for tractability and
- it is important for improving solve times
- loose bounds on input to some unit will propagate downstream, leading to units in later layers having looser bounds.

For instance, if we can prove that the phase of a ReLU is stable, we can avoid introducing a binary variable. In the paper they discuss two different methods: Interval Arithmetic (IA), and the slower but tighter Linear Programming (LP) approach. Implementation details are provided in the paper.

Note:

```
GETBOUNDSFORMAX(xs, fs)
▷ fs are the procedures to determine bounds, sorted in increasing computational complexity.
d_l = {x : −∞ for x in xs}
d_u = {x : ∞ for x in xs}
▷ initialize dictionaries containing best known upper and lower bounds on xs
l_max = −∞      ▷ l_max is the maximum known lower bound on any of the xs
a = {xs}
▷ a is a set of active elements in xs that can still potentially take on the maximum value.
for f in fs:    ▷ carrying out progressive bounds tightening
    do for x in xs:
        if d_u[x] < l_max
          then a.remove(x)      ▷ x cannot take on the maximum value
          else u = f(x, boundType = upper)
               d_u[x] = min(d_u[x], u)
               l = f(x, boundType = lower)
               d_l[x] = max(d_l[x], l)
               l_max = max(l_max, l)
return (a, d_l, d_u)
```

Figure 2: Pseudocode demonstrating how to efficiently determine bounds for the tightest possible formulations for the Max

- Faster procedures achieve efficiency by compromising on tightness of bounds i.e., tradeoff between higher build times (to determine tighter bounds to inputs to non-linearities), and higher solve times (to solve the main MILP problem in Equation 2 or Equation 3-5)
- For piecewise-linear non-linearities, there are thresholds beyond which further refining a bound will not improve the problem formulation.

Based on this [9] provide a coarse bounds using fast procedures and only spend time refining bounds using procedures with higher computational complexity if doing so could provide additional information to improve the problem formulation.

Please refer to the **Fig 2**. *GETBOUNDSFORMAX* finds the tightest bounds required for specifying the constraint $y = max(x)$. Also we stop tightening the bounds on a variable if its maximum possible value is lower than the minimum value of some other variable. *GETBOUNDSFORMAX* returns a tuple containing the set of elements in x that can still take on the maximum value, as well as a dictionary of upper and lower bounds. Similarly please refer to **Fig 1** for pseudocode which shows how to tighten the bound if we are using ReLU. For proofs behind this algorithms as mentioned earlier please refer to the paper.

2.5 Expressing the norms as the objective of an MILP model

Here I have showed how to cast the problem when we use l_∞ and l_2 norm. For l_1 norm please refer to the paper.

$\boldsymbol{l_\infty}$: When $d(x^{'}, x) = \|x^{'} - x\|_\infty$, we introduce the auxiliary variable ϵ, which bounds the l_∞ norm: $\epsilon \geq x_j^{'} - x_j$ and $\epsilon \geq x_j - x_j^{'}$. The optimization in Equation 3-5 is equivalent to

$$min_{x^{'}} \epsilon \tag{8}$$

$$\text{subjected to} \quad \text{argmax}_i(f_i(x^{'})) \neq \lambda(x) \tag{9}$$

$$x^{'} \in \mathcal{X}_{valid} \tag{10}$$

$$\epsilon \geq x_j^{'} - x_j \tag{11}$$

$$\epsilon \geq x_j - x_j^{'} \tag{12}$$

l_2: When $d(x^{'}, x) = \|x^{'} - x\|_2$, the objective becomes quadratic, and we have to use a Mixed Integer Quadratic Program (MIQP) solver. However, no auxiliary variables are required: the optimization in Equation 3-5 is simply equivalent to

$$min_{x'} \sum_j (x_j^{'} - x_j)^2 \tag{13}$$

$$\text{subjected to} \quad \text{argmax}_i(f_i(x^{'})) \neq \lambda(x) \tag{14}$$

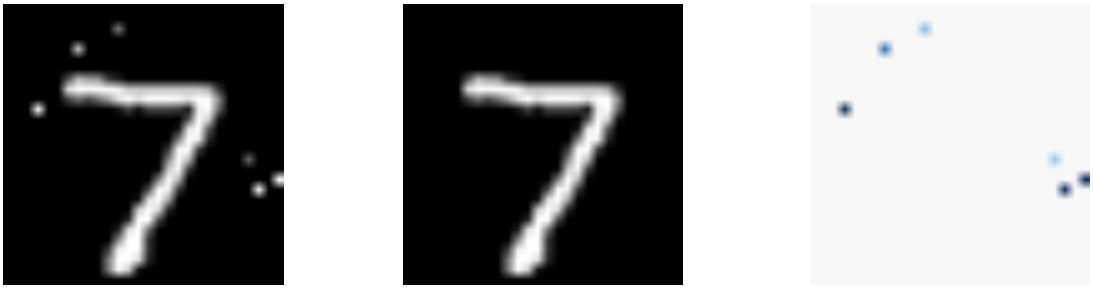
$$x^{'} \in \mathcal{X}_{valid} \tag{15}$$

3 Results

Using with the help of Gurobi [16] and MILP-verifier package provided in [9], I have tried create to adversarial examples for a neural network trained on MNIST dataset. Neural network architecture I had was three layers with 40 and 20 ReLU units in first two layers and 10 SoftMax units in the last layer. In the **Fig. 3b** we can see we have image of digit 7 and we have generated an adversarial sample (**Fig. 3a**) with the help of l_1 norm which when given as input to the neural network it will be classified as 9. Note that the test accuracy of our neural network on test dataset was over 96%. This show how vulnerable is supervised learning. Difference between the pixel values for adversarial and true image is shown in **Fig. 3c**.

I have performed similar analysis for norms l_∞ and l_2 as shown in **Fig. 4** and **Fig. 5** respectively. But we can see the adversarial sample generated is very subtle and still neural network fails to classify them. We can apply same procedure to compute adversarial sample for different input images and we can also generate a adversarial samples for same digit that will be missclassified as anything but the true label. Similarly, we can also use blurring kernel to generate images that will be blurred images of input images and they will be missclassified. Please refer to the **Fig. 6**. I have also generated an interesting case where input image is 7 but it due to slight variation it will be classified as 4. Please refer **Fig. 7**. I have also generated an another case where input image is 2 but it due to slight variation it will be classified as 9. Please refer **Fig. 8**.

With the help of their tutorial I have tested their bound tightening algorithm. Depend the on how many adversarial samples we generate and which norm we use we can different ranges of solve time. On an average for generating one set adversarial examples it took me 160 seconds. Please refer to the code for more detailed implementation.

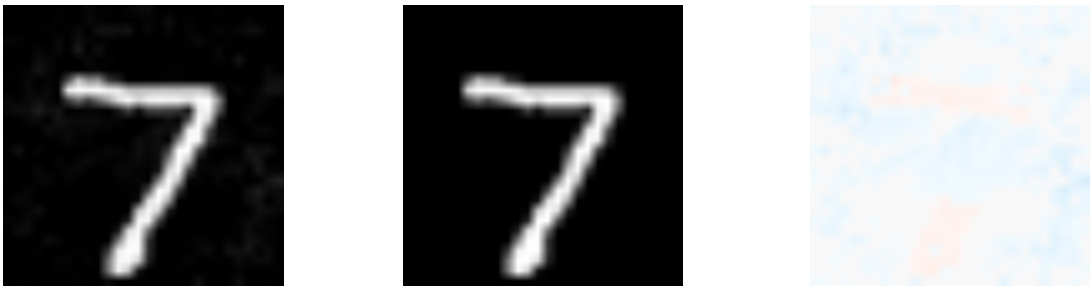


(a) Input Image of 7 Classified as 9 (b) Input Image of 7 Classified as 7 (c) Difference between the images

Figure 3: Plots showing the distance between an adversarial example for image of digit 7. Distance metric is l_1 norm

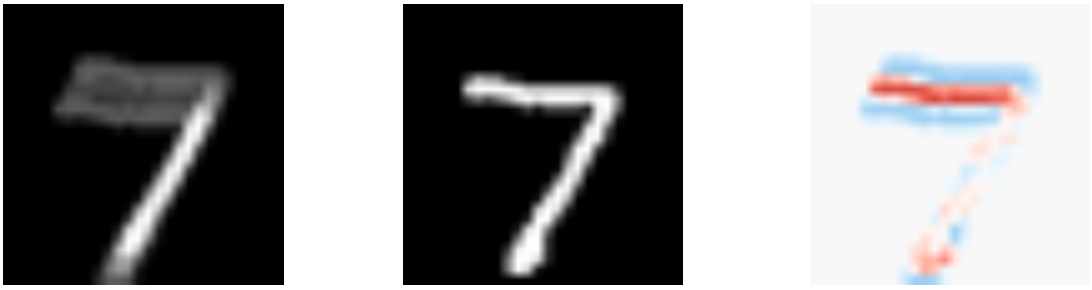
(a) Input Image of 7 Classified as 9 (b) Input Image of 7 Classified as 7 (c) Difference between the images

Figure 4: Plots showing the distance between an adversarial example for image of digit 7. Distance metric is l_∞ norm



(a) Input Image of 7 Classified as 9 (b) Input Image of 7 Classified as 7 (c) Difference between the images

Figure 5: Plots showing the distance between an adversarial example for image of digit 7. Distance metric is l_2 norm



(a) Input Image of 7 Classified as 9 (b) Input Image of 7 Classified as 7 (c) Difference between the images

Figure 6: Here we similar analysis of as mentioned but the difference is that we have blurred image as our adversarial example. Distance metric is l_1 norm

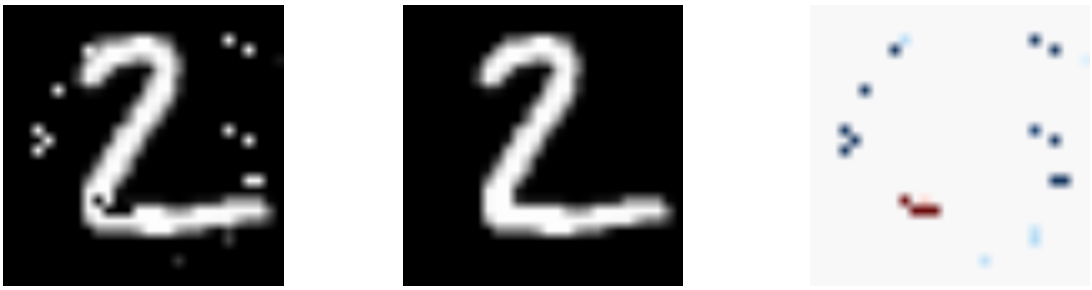
4 Conclusion

Here we have seen an efficient complete verifier for piecewise-linear neural networks. It is focused on evaluating networks on the class of perturbations they are designed to be robust to, defining a class of perturbations that generates images perceptually similar to the original remains an important direction of research. The verifier is able to handle new classes of perturbations (such as convolutions applied to the original image) as long as the set of perturbed images is a union of polytopes in the input space.

Basically this verifier has succeed at evaluating the robustness of larger neural networks, including those with convolutional and residual layers. Also, this formulation is fast at evaluation because ReLU

(a) Input Image of 7 Classified as 4 (b) Input Image of 7 Classified as 7 (c) Difference between the images

Figure 7: Here we see an interesting case where input image 7 is being classified as 4. Distance metric is l_1 norm



(a) Input Image of 2 Classified as 9 (b) Input Image of 2 Classified as 2 (c) Difference between the images

Figure 8: Here we see an another case where input image 2 is being classified as 9. Distance metric is l_1 norm

can be inactive many times and the predicted label is determined by the unit in the final layer with the maximum activation, proving that a unit never has the maximum activation over all bounded perturbations eliminates it from consideration.

Through this project and course work I was able to understand a new research area that exits in machine learning and optimization. I was able to understand how optimization techniques can be used to apply it on fields involving machine learning and to check the robustness of a neural network. My main of this project was learn and understand the paper [9] and [13] and see how to apply integer linear programs techniques to machine learning problem.

5 Future Work

Some ideas to improve the verification of neural networks based on [9] is

- These improvements can be combined with other optimizations in solving MILPs. For example, [13] discusses splitting on the input domain, producing two sub-MILPs where the input in each sub-MILP is restricted to be from a half of the input domain.
- Taking advantage of locally stable ReLUs speeds up verification; network verifiability could be improved during training via a regularizer that increases the number of locally stable ReLUs.
- Sparsifying weights promotes verifiability. Adopting a principled sparsification approach (for example, l1 regularization during training, or pruning and retraining [15]) could potentially further increase verifiability without compromising on the true adversarial accuracy.

- Extend these verification techniques to the field of reinforcement learning where adversarial samples will acts as unseen states that agent has not learnt. We can use this to train more robust RL agents which are trained to perform robotic tasks or self driving cars.

References

[1] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus, *Intriguing properties of neural networks*. In International Conference on Learning Representations, 2014.

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

[3] Nicholas Carlini and David Wagner, *Adversarial examples are not easily detected: Bypassing ten detection methods.* In Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security, pp. 3–14, 2017a.

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

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

[6] Anish Athalye, Nicholas Carlini, and David Wagner, *Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples*. In Proceedings of the 35th International Conference on Machine Learning, pp. 274–283, 2018.

[7] 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, pp. 2613–2621, 2016.

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

[9] Vincent Tjeng, Kai Xiao, Russ Tedrake, *Evaluating Robustness of Neural Networks with Mixed Integer Programming*. In International Conference on Learning Representations, 2019.

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

[11] Stephen Boyd and Lieven Vandenberghe, *Convex Optimization*. Cambridge University Press, 2004.

[12] J Zico Kolter and Eric Wong, *Provable defenses against adversarial examples via the convex outer adversarial polytope*. In Proceedings of the 35th International Conference on Machine Learning, 2017.

[13] Rudy Bunel, Ilker Turkaslan, Philip HS Torr, Pushmeet Kohli, and Pawan K Mudigonda, *A unified view of piecewise linear neural network verification*. In Advances in Neural Information Processing Systems, 2018.

[14] Matteo Fischetti and Jason Jo, *Deep neural networks and mixed integer linear optimization*. Constraints, 23(3):296–309, July 2018. ISSN 1383-7133 [URL].

[15] Song Han, Huizi Mao, and William J Dally, *Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding*. In Advances in Neural In International Conference on Learning Representations, 2016.

[16] Inc. Gurobi Optimization. Gurobi optimizer reference manual, 2019 [URL].