CHAPTER 7

Overview of neural network verification

7.1 Robustness verification versus adversarial attack

To introduce the concept of robustness verification, we will first mathematically define the notion of adversarial robustness and discuss the relationship between robustness verification and attack.

We consider a classification model $F:\mathbb{R}^d \to \{1,\dots,K\}$, where K is the number of classes. For simplicity, we consider the most simple ℓ_p norm threat model, where an adversarial example aims to change the prediction with minimum ℓ_p norm perturbation. In this case, for an input example $\boldsymbol{x}_0$, the *minimum adversarial perturbation* can be mathematically defined as

$$\epsilon^* = \min_{\boldsymbol{\delta}} \|\boldsymbol{\delta}\|_p \quad \text{s.t.} \quad F(\boldsymbol{x}_0+\boldsymbol{\delta}) \neq F(\boldsymbol{x}_0). \tag{7.1}$$

The value of ϵ^* defines the distance to the closest decision boundary and also gives a "safe region", which guarantees that the prediction remains unchanged within $\mathcal{B}_{\boldsymbol{x}_0}(\epsilon^*) := \{\boldsymbol{x} \mid \|\boldsymbol{x}-\boldsymbol{x}_0\| < \epsilon^*\}$. This is also the shortest distance from $\boldsymbol{x}_0$ to the decision boundary of the model F, as illustrated in Fig. 7.1.

Unfortunately, computing ϵ^* is NP-complete even for a simple ReLU network (Katz et al., 2017). Therefore we can only expect an efficient way for computing upper or lower bounds of ϵ^*.

Adversarial attacks are developed for finding a feasible solution $\bar{\boldsymbol{\delta}}$ of (7.1), which corresponds to a counterexample for the robustness property. The norm of $\bar{\boldsymbol{\delta}}$ can then serve as an *upper bound of* ϵ^**, denoted by* $\bar{\epsilon}$. For neural networks, several attacking algorithms have been proposed (see Chapter 2), including L-BFGS (Szegedy et al., 2014), FGSM (Goodfellow et al., 2015), I-FGSM (Kurakin et al., 2016), C&W (Carlini and Wagner, 2017b), and PGD attack (Madry et al., 2018). All of them try to solve (7.1) or its reformulations by a gradient-based optimizer. However, due to the non-convexity of neural networks, attacks may not converge to the minimal adversarial perturbation, so the robustness measurement computed by an attack may not represent the true adversarial robustness. There are many examples that a model was considered "robust" under standard attack but

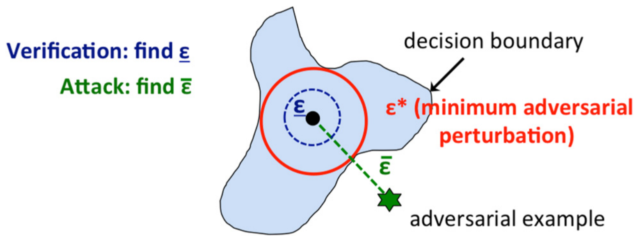
Verification: find $\underline{\epsilon}$
Attack: find $\bar{\epsilon}$
decision boundary
$\underline{\epsilon}$
ϵ^* (minimum adversarial perturbation)
$\bar{\epsilon}$
adversarial example

Figure 7.1 Illustration of adversarial robustness, attack and verification.

later on was broken by more carefully designed attacks (Athalye et al., 2018; Carlini and Wagner, 2017a; Carlini et al., 2019a).

Robustness verification. In comparison, robustness verification methods aim to give a reliable measurement of robustness. Finding the exact value of ϵ^* is regarded as *sound and complete verification*, which requires exponential computational complexity. For instance, several branch-and-bound algorithms were developed to split the problem into an exponential number of sub-regions and solve each of them separately. They usually only scale to less than 100 neurons and couldn't handle most of the practically used neural networks.

To address the computational issue, algorithms for *sound but incomplete* neural network verification has become increasingly important. Those verification algorithms aim to compute a lower bound $\underline{\epsilon} \leq \epsilon^*$ of minimum adversarial perturbation. This guarantees that no adversarial example exists within an $\underline{\epsilon}$-ball around x no matter which attack is used. Fig. 7.1 illustrates the relationships between sound verification, attack, and minimum adversarial perturbation. The word "soundness" comes from the formal verification community, since with $\underline{\epsilon}$ available, we can provably state that some points are ϵ-robust (when $\epsilon \leq \underline{\epsilon}$), which is the property of soundness. However, when $\epsilon > \underline{\epsilon}$ we do not know whether it is robust, so the verification is "incomplete".

In addition to viewing robustness verification as finding the lower bound $\underline{\epsilon} \leq \epsilon^*$, we can also formulate it as the following problem:

$$\text{Decide whether there exists } \boldsymbol{x}' \in \{\boldsymbol{x} \mid \|\boldsymbol{x} - \boldsymbol{x}_0\| \leq \epsilon\} \text{ such that } F(\boldsymbol{x}_0) \neq F(\boldsymbol{x}'). \tag{7.2}$$

If we can answer this decision problem exactly, then a binary search can immediately give an accurate ϵ^* value. Instead of solving this decision problem exactly, sound (but incomplete) verification algorithms will answer the question with yes if the robustness of the point can be verified within ϵ perturbation, but will not give a certified answer for the rest of the points.

7.2 Formulations of robustness verification

In this section we briefly show how to formulate robustness verification problems mathematically and discuss the challengings of solving it.

We first consider a binary classification neural network, where the output layer only has one neuron $f(x)$ and the label prediction is sign($f(x)$). In this case, if we denote the perturbation set as $S_\epsilon(\boldsymbol{x}_0) := \{\boldsymbol{x} \mid \|\boldsymbol{x} - \boldsymbol{x}_0\|_p \leq \epsilon\}$ and assume the original example $\boldsymbol{x}_0$ is predicted as $+1$, then problem (7.2) is equivalent to solving

$$\min f(x) \quad \text{s.t.} \quad x \in S_\epsilon(\boldsymbol{x}_0). \tag{7.3}$$

When the solution of (7.3) is negative, there exists a perturbation that can change the prediction from positive to negative, so the model is not robust on $\boldsymbol{x}_0$; otherwise if the solution of (7.3) is positive, then the model is provably robust on $\boldsymbol{x}_0$. A sound but incomplete verification will produce a lower bound of (7.3); when the lower bound is positive, the model is provably robust on $\boldsymbol{x}_0$; otherwise if the lower bound is negative, the robustness is not certified.

Therefore, we can see that robustness verification is equivalent to solving an optimization problem. To gain more intuition why this optimization problem is difficult when $f(x)$ is a neural network, let's further expand the function $f(x)$. For simplicity, let us assume that $f(\boldsymbol{x})$ is an L-layer feedforward network:

$$f(\boldsymbol{x}) := \boldsymbol{z}^{(L)} = W^{(L)}\boldsymbol{x}^{(L)} + \boldsymbol{b}^{(L)}, \qquad \boldsymbol{x}^{(l)} = \sigma^{(l)}(W^{(l)}\boldsymbol{x}^{(l)} + \boldsymbol{b}^{(l)}), \qquad l \in [L], \tag{7.4}$$

where $\boldsymbol{x}^{(0)} = \boldsymbol{x}$ is the input, $W^{(l)}$ and $\boldsymbol{b}^{(l)}$ are the weight matrix and bias vector of the lth layer, $\sigma^{(l)}$ is a (nonlinear) activation function, and $z^{(L)}$ in the binary classification case will be a single output neuron corresponding to the value of $f(x)$. With these definitions, the optimization problem (7.3) can be further expanded as

$$\min_{\boldsymbol{x}^{(0)} \in S_\epsilon} \quad z^{(L)} \text{ s.t. } \boldsymbol{z}^{(l)} = W^{(l)}\boldsymbol{x}^{(l)} + \boldsymbol{b}^{(l)}, \tag{7.5}$$

$$\boldsymbol{x}^{(l+1)} = \sigma^{(l)}(\boldsymbol{z}^{(l)}), \tag{7.6}$$

$$l \in [L]. \tag{7.7}$$

In this minimization problem, constraint (7.5) is just a linear equality constraint, but constraint (7.6) is a non-linear and non-convex constraint when $\sigma^{(l)}$ is a nonlinear function such as ReLU. Therefore, it is theoretically difficult to solve this optimization problem exactly. The idea in sound but incomplete verification aims to relax constraint (7.6) to make the problem tractable while being able to compute a lower bound of the solution. We will discuss how to achieve a lower bound and the exact solution of this problem in the next two chapters, corresponding to algorithms for incomplete and complete verifications.

Before delving into the actual verification algorithms, we will first discuss the generalized form of (7.3). In fact, neural network verification can be used to verify many other properties in addition to adversarial robustness. Given a neural network $f : \mathbb{R}^d \to \mathbb{R}^K$ that maps input to output neurons, verification aims to find a lower bound or exact solution of an affine function within an input region S:

$$\min \langle c, f(\boldsymbol{x})\rangle \text{ s.t. } \boldsymbol{x} \in S, \tag{7.8}$$

where c is called a linear specification, which corresponds to the property we want to verify. In the setting of binary classification, as in formulation (7.3), $f(x)$ is just a single value and we can set $c = 1$ for verification. For multi-class classification problem, by setting c as a vector with 1 on the positive class and -1 on the opposite class (e.g., class t), the sign of the solution of (7.8) indicates whether there exists any adversarial example predicted as class t. If we verify there exists no adversarial example for each class t, then the robustness can be certified. In addition to robustness verification, this framework is also used to guarantee safety in many real-time control systems, such as aircraft controls (Julian et al., 2019; Katz et al., 2017).

7.3 Applications of neural network verification

As mentioned before, neural network verification can verify not only robustness, but also other safety specifications as long as the specification can be written as an affine function locally. Here we introduce some important applications for neural network verification.

Safety-critical control systems

Verification has been used in many control systems. Among them, the safety verification problem of next-generation airborne collision avoidance system for unmanned aircraft (ACAS Xu) has been used as the benchmark for this field (Katz et al., 2017; Julian et al., 2019), and several other practical systems are also considered in the previous verification work such as a LiDAR object detection system (Sun et al., 2019a), a perception network for automated driving (Cheng, 2019), and a fuel injecting system (Wong et al., 2020b). Taking the ACAS Xu system as an example, it contains 45 neural networks with hundreds of neurons, and the goal is to verify whether these networks satisfy some predefined safety specifications for collision avoidance.

Natural language processing

The robustness of natural language models have become an important problem in natural language processing. It has been shown that semantic preserved perturbations (e.g., replacing a word by its synonym) can alter the model's prediction in many NLP tasks. Certified defense methods have been applied in (Jia et al., 2019; Huang et al., 2019) by using neural network verification methods and extending to the discrete perturbation setting.

Machine learning interpretability

Neural network verification is also an important tool for understanding the model's behavior. For instance, many feature-based explanations (Hsieh et al., 2020) are based on the sensitivity of models – how much perturbation to a feature can change the model's prediction – which can be easily answered by verification tools. Also, verification methods can be used to find an important set of features as well as identifying unimportant features that are provably having no effect on the model.

7.4 Extended reading

- A survey on neural network verification in (Liu et al., 2019a) gives an overview on both incomplete and complete verifications.
- The International Verification of Neural Network Competition (VNN-Comp) that takes place each year has been a place for teams demonstrating state-of-the-art tools on neural network verification on various problems.