10-707: Advanced Deep Learning **Spring 2020**

Lecture 22: April 20

Lecturer: Elan Rosenfeld *Scribes: Nick Roberts*

Note: *LaTeX template courtesy of UC Berkeley EECS dept.*

22.1 Introduction

When training a neural network, we perform forward propagation of an input, compute a loss, compute the gradient of the loss with respect to the parameters of the network, and then we update the parameters of the network via gradient descent (taking a step in the negative direction). What if we instead took the gradient *with respect to the input, and took a step in the positive direction?*

Doing this (when the step is sufficiently large) causes a network to produce incorrect predictions. Here, the updated inputs are called **adversarial examples**. Even worse, these attacks generalize to other datasets and models and they are *completely imperceptible to humans*. Furthermore, it isn't clear why this is happening. Some theories suggest that this has to do with the nonlinear decision boundaries learned by deep neural networks.

22.2 Part 1: Attacks

22.2.1 Threat models

In computer secutity literature, it is important to define a threat model, or a model of your adversary. In the context of adversarial examples, threat models fall into one of two categories:

1. **White box:** adversary has full access to model and gradients
2. **Black box:** adversary only has query access to the model

In general, most of the literature concerns white box attacks (so we will focus on these).

22.2.2 Methods

The most popular attack is the **fast gradient sign method (FGSM)**, and it is written as:

$$\delta = \epsilon sign(\nabla_x L(x; \theta))$$

The current SOTA attack is the **projected gradient descent** attack. In this attack, we apply the following procedure:

1. Take a step in the direction of the gradient
2. Project back onto the ℓ_p norm ball
3. Repeat (the more iterations, the better)

22.3 Part 2: Defenses

22.3.1 Methods

There are many proposed defenses against adversarial examples. Some of which are as follows:

1. Add non-differentiability to the model after training
2. Add randomness at test time
3. Evaluate extremely deep networks with repeated computation such that the gradient vanishes

However, none of these methods, called obfuscated gradients, are effective, according to a 2018 paper.

One method, however, seems to be effective (so far, at least). That is, we adversarially perturb examples during training. This is very slow, unforunately.

The security community favors *provable defenses* over empirical ones. Two examples of provable defenses are as follows:

1. **Convex Outer Adversarial Polytope** As we pass the input through the network, maintain a convex envelope of possible activations caused by a perturbation. Then train the network to minimize this envelope. *Unfortunately, this is very slow and grows with the size of the network.*
2. **Interval Bound Propagation** As we pass the input through the network, maintain an axis-aligned polytope of possible activations caused by a perturbation. Use this to certify the worst possible case. *Still slow and doesn't scale to large datasets.*
3. **Randomized Smoothing** Define a new network g whose output is f convolved with Gaussian noise. This new network g can be certified as a function of the margin with which the largest softmax class wins. *Scales well, but noise must have high variance if the adversary uses infinity norm perturbations.*

22.4 Part 3: Moving forward

22.4.1 The cost of adversarial robustness

For large datasets, adversarial training seems to result in a large drop in standard accuracy.

22.4.2 One last theory

Adversarial examples are not bugs, they are features! [Ilyas et al. 2019] suggest that neural networks purely learn from statistical properties of the data. This implies that there are two categories of features:

1. **robust features:** features which may be salient to a human (features might be "ears" or "snout")
2. **non-robust features:** features which are not salient to a human (pixel at (2, 76) is #FFF23D)

We can actually test this by adversarially perturbing the training inputs at training time to the label+1, and leaving the labels as-is at test time. Sure enough, the network still generalizes to the test set.