CSE 251A - ML: Learning Algorithms

Project 3 — Adversarial Examples for Neural Networks

It has been seen that many classifiers, including neural networks, are highly susceptible to what are called *adversarial examples* – small perturbations of the input that cause a classifier to misclassify, but are imperceptible to humans. For example, making a small change to an image of a stop sign might cause an object detector in an autonomous vehicle to classify it as a yield sign, which could lead to an accident.

In this assignment, we will see how to construct adversarial examples for neural networks, and you are given a 3- hidden layer perceptron trained on the MNIST dataset for this purpose.

Download from this announcement post on Piazza a zip file named `pa3.zip`. This file contains the test portion of the MNIST dataset; each row of this file contains a vector representation of a 28×28 image, followed by its label (between 0 and 9). It also contains code for a neural network, already trained on MNIST, that you will run experiments on (i.e. attempt to discover adversarial examples for). A description of the file formats, an iPython notebook for loading the neural network and the data, as well as some documentation on how to compute the gradients is also available in the zip file we provide.

On the due date, upload Gradescope a **typewritten** (i.e. using Latex) report containing the following elements (each labeled clearly). Please also append the additional code you developed (no need to include the code we already provide) at the end of your report.

1. *Fast Gradient Sign Method.*

   We begin with deriving a simple way of constructing an adversarial example around an input (x, y). Supppose we denote our neural network by a function $f : X \to \{0, \ldots, 9\}$.

   Suppose we want to find a small perturbation Δ of x such that the neural network f assigns a label different from y to $x + \Delta$. To find such a Δ, we want to increase the cross-entropy loss of the network f at (x, y); in other words, we want to take a small step Δ along which the cross-entropy loss increases, thus causing a misclassification. We can write this as a gradient ascent update, and to ensure that we only take a small step, we can just use the sign of each coordinate of the gradient. The final algorithm is this:

   $$\tilde{x} = x + \epsilon \cdot sign(\nabla L(f, x, y)),$$

   where L is the cross-entropy loss, and it is known as the Fast Gradient Sign Method (FGSM).

   First, implement the Fast Gradient Sign Method (FGSM) for the neural network given to you. Then, evaluate and report the accuracy of the neural network on adversarial examples. This is computed as follows – for each test example $x^{(i)}$, generate an adversarial example $\tilde{x}^{(i)}$ for $\epsilon = 0.1$. The neural network is correct if it predicts $y^{(i)}$ on $\tilde{x}^{(i)}$ and wrong otherwise.

2. *New Method and Pseudocode.*

   Next, design your own method to find adversarial examples. Your algorithm should take as input a labeled example (x, y) and a perturbation amount ϵ, and output an adversarial example $\tilde{x}$ such that the infinity norm of the difference between x and $\tilde{x}$, $\|x - \tilde{x}\|_\infty$, is at most ϵ.

   Describe your algorithm, and write down clear pseudocode for it. Once again, clarity and conciseness are of the essence.

3. *Experimental results.*

   A (clearly labeled) table or graph of results showing the accuracy of the given neural network on adversarial examples generated by your algorithm vs. the fast gradient sign method on the given MNIST test set as a function of ϵ. Report the accuracy on adversarial examples for $\epsilon = 0.05, 0.1, 0.15, 0.2$. For any strategy with randomness, you should do several experiments and give error bars – give all relevant details, including the formulas you used for computing confidence intervals.

   The pseudocode and experimental details must contain all information needed to reproduce the results.

4. *Critical evaluation.*

   Is your method a clear improvement over Fast Gradient Sign Method? Is there further scope for improvement? What would you like to try next?