CSE 250B: Machine learning **Winter 2019**

Programming project 4 — Adversarial examples for neural nets

Many classifiers, including neural nets, are susceptible to what are called *adversarial examples*: small perturbations of the input that are imperceptible to humans, yet cause the classifier to predict the wrong label. 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 nets. You are given a net with 3 hidden layers, trained on the MNIST dataset, for this purpose.

From the course website, download the provided zipped file that 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). This is in the `data` directory.
- A neural net that has been pre-trained on MNIST. This is also in the `data` directory.
- An iPython notebook that loads the neural net and the data.

The assignment

On the due date, upload (to `gradescope`) a **typewritten** report containing the following elements (each labeled clearly).

1. *Fast Gradient Sign Method.*

   We begin by deriving a simple way of constructing an adversarial example around an input (x, y). Denote the neural net by a function $f : X \to \{0, \ldots, 9\}$.

   Suppose we want to find a small perturbation Δ to x such that the neural net f assigns point $\tilde{x} = x + \Delta$ a label different from y. 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:

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

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

   First, implement the Fast Gradient Sign Method (FGSM) for the neural net given to you. Then, evaluate and report the accuracy of the neural net 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.

   To turn in:

   - Error rate of the neural net on the original data set and on the perturbed version.
   - An example of an error due to perturbation: the original image, the perturbed version, and the two labels.

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.

   To turn in:

   - A high-level description of your method.
   - Unambiguous pseudocode.

   As always, clarity and conciseness are of the essence.

3. *Experimental results.*

   To turn in:

   - 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.*

   To turn in:

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

Notes on gradient calculation

Here is a brief explanation of the structure and the forward propagation process of the provided neural net, along with details of the gradient computation for the cross-entropy loss with respect to the input image.

Structure and forward propagation

The neural net is a fully-connected multi-layer perceptron with three hidden layers. The hidden layers contain 2048, 512 and 512 hidden nodes respectively. We use ReLU as the activation function at each hidden node. The last intermediate layer's output is passed through a softmax function, and the loss is measured as the cross-entropy between the resulted probability vector and the true label.

We use the following notation.

1. x: the input image vector with dimension 1×784.
2. y: the true class label of x.
3. z^i: the value of the i-th intermediate layer *before* activation, with dimension 1×2048, 1×512, 1×512 and 1×10 for $i = 1, 2, 3, 4$.

4. h^i: the value of the i-th intermediate layer *after* activation, with dimension 1×2048, 1×512 and 1×512 for $i = 1, 2, 3$.

5. p: the predicted class probability vector after the softmax function, with dimension 1×10.

6. W^i: the weights between the $(i-1)$-th and the i-th intermediate layer. For simplicity, we use h^0 as an alias for x. Each W^i has dimension $\ell_{i-1} \times \ell_i$, where ℓ_i is the number of nodes in the i-th layer. For example, W^1 has dimension 784×2048.

7. b^i: the bias between the $(i-1)$th and the i-th intermediate layer. The dimension is $1 \times \ell_i$.

The forward propagation rules are as follows.

$$z^i = h^{i-1}W^i + b^i \quad \text{for} \quad i = 1, 2, 3, 4$$

$$h^i = ReLU(z^i) \quad \text{for} \quad i = 1, 2, 3$$

$$p = \text{softmax}(z^4)$$

Gradient calculation

Let L denote the cross-entropy loss of an image-label pair (x, y). We are interested in the gradient of L w.r.t. x, since we will move x in the direction of (the sign of) the gradient to increase L. If L becomes large, the new image x_{adv} will likely be misclassified.

We use the chain rule for gradient computation. Again, let h^0 be the alias of x. We have

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial h^0} = \frac{\partial L}{\partial z^4} \frac{\partial z^4}{\partial h^3} \prod_{i=1}^{3} \frac{\partial h^i}{\partial h^{i-1}} = \frac{\partial L}{\partial z^4} \frac{\partial z^4}{\partial h^3} \prod_{i=1}^{3} \left(\frac{\partial h^i}{\partial z^i} \frac{\partial z^i}{\partial h^{i-1}} \right).$$

The intermediate terms can be computed as follows.

$$\frac{\partial L}{\partial z^4} = p - y$$

$$\frac{\partial z^i}{\partial h^{i-1}} = (W^i)^T$$

$$\frac{\partial h^i}{\partial z^i} = \text{diag}(1(h^i > 0)).$$