COM 312 - Neural Networks and Deep Learning

Assignment 1 – Learning MNIST Digits Using the Perceptron

In this Assignment, you will implement 10 'perceptrons' to learn to recognize ten hand-written digits. Each perceptron will learn to detect one of ten sample digits from the MNIST database. Perceptron learning for this assignment should use the *delta rule* and the *sigmoid squashing function* as described in class (and see below). The MNIST database is a standard database of images (hand written digits from zero to nine) used in comparing the results of learning across a variety of neural network models. Each of the 70,000 images from the database is composed of 28x28 pixels of continuous values. Figure 1 illustrates some sample images from the MNIST database.

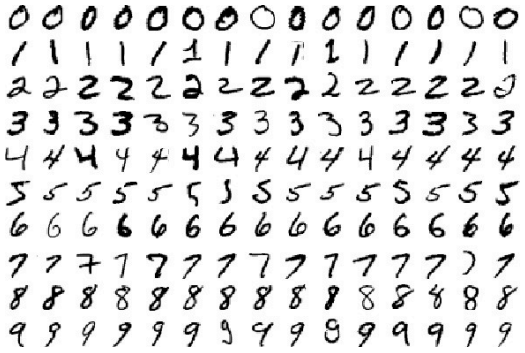

Figure 1: example images from the MNIST database

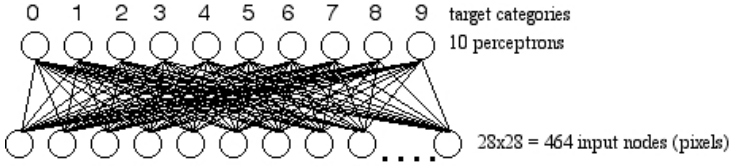

Figure 2: 10 perceptrons to learn digit categories based on 28x28 images (784 pixels)

Output for each perceptron is found following Equation 1. Activation for the output node or perceptron *j* is found as the sum of input to the perception, scaled by the output node's bias, and then passed through a squashing function $\sigma(\ )$. The bias for a perceptron is typically implemented as an extra input node always set to 1.0 passed through an extra weight. For convenience, the MNIST library function included for this assignment 'get_input()' returns a vector of pixel values plus an extra value at the beginning of the vector that is always set to one (1+784 = 785 values in a digit vector).

(1) $$o_j = \sigma(\sum o_i \cdot w_{ij}) \qquad \sigma(x) = \frac{1}{1+e^{-x}}$$

Once the output for each perceptron is found, the weights to each perceptron can be updated using the *delta rule* (Equation 2). The delta rule seeks to reduce the error, of a perceptron by changing the weights to the perceptron slightly per training iteration, depending on the learning rate η. The error of a perceptron is found as the difference between the Target, *T,* (what the perceptron should have output) and the actual output o_j produced by the perceptron based on its current input weights.

(2) $$\Delta w_{ij} = \eta \cdot o_i \cdot Error_j \qquad Error_j = T_j - o_j$$

Problem 1

The assignment is divided up into two problems: this first problem is to get a program working and results analyzed. We therefore will start with a simple dataset of ten images to be learned – one image per digit category (use loadType = 2 in the mnistLoad() library function for this). Then, using the library function 'get_input(…)', you will load the vector *inputVec* for the digit *j* from the *zData* repository per training example with different levels of noise. Given no noise, you should be able to train your network to learn the samples fairly quickly. After doing this, you should adjust your learning rate to be optimal. Then experiment with other noise levels and learning rates.

Deliverables:

1) Turn in your code (don't modify the rand.h or nmist.h libraries. Your code should compile and run on the classroom machines) – 30 points

2) Analyze results given six different noise conditions: {noise = 0, .1, .2, .3, .4, .5}. Average each result over 20 training simulations. What seems to be the optimal learning rate per noise condition? what seems to be the lowest error rate that can be achieved in each condition? Plot some graphs to exemplify your results. – 30 points

Problem 2

Train the neural network you built in Problem 1 on the full 60,000 training images (loadType = 0). After every training epoch, test your network on the 10,000 testing images (loadType = 1) and inspect your results during the course of training. Use three different learning rates { η = .01, .001, .0001} and one more learning rate of your choosing – this fourth learning rate should be what you analyze to be the optimal learning rate.

Deliverables:

1) Turn in your code (it should be very similar or the same as Problem 1). Again, it should compile and run on the classroom machines – 10 points

2) Plot the results of learning over time using the four learning rates, averaged over five simulations per learning rate. What is your optimal learning rate? What was the lowest amount of error you achieved and about many training cycles did it take to achieve? – 30 points