Assignment 3 – Networks Networks
CS 2400 – Fall 2020
Due: Friday, November 6

Requirements

In this assignment you will write a program that creates a perceptron and uses backpropagation to train it to recognize discretized images of handwritten digits. We will fix the learning rate at 0.01 and the number of epochs of training at 1000. We have a set of 5,620 examples. Those examples need to be divided into a training set and a test set. I want you to run experiments to gauge the effect of the proportion of that split. I want you to try nine possible splits: train on 10% of the examples and test on the remaining 90%, train on 20% of the examples and test on the remaining 80%, up to training on 90% of the examples and test on the remaining 10%.

The number of input nodes and the number of output nodes are fixed by the structures of the examples (see below). Your code needs to be able to create a perceptron, read the examples, divide them up, train the perceptron on the training set, and test the perceptron on the test set. So, in particular, your perceptron needs to be able to:

1. evaluate an input, i.e. process a given input vector through the network and produce the output vector, and

2. use backpropagation to train the network given an input/output pair.

We will define performance in terms of the average error on the test set where the error on a test item is the Euclidean distance between the target vector t and the actual output vector o:

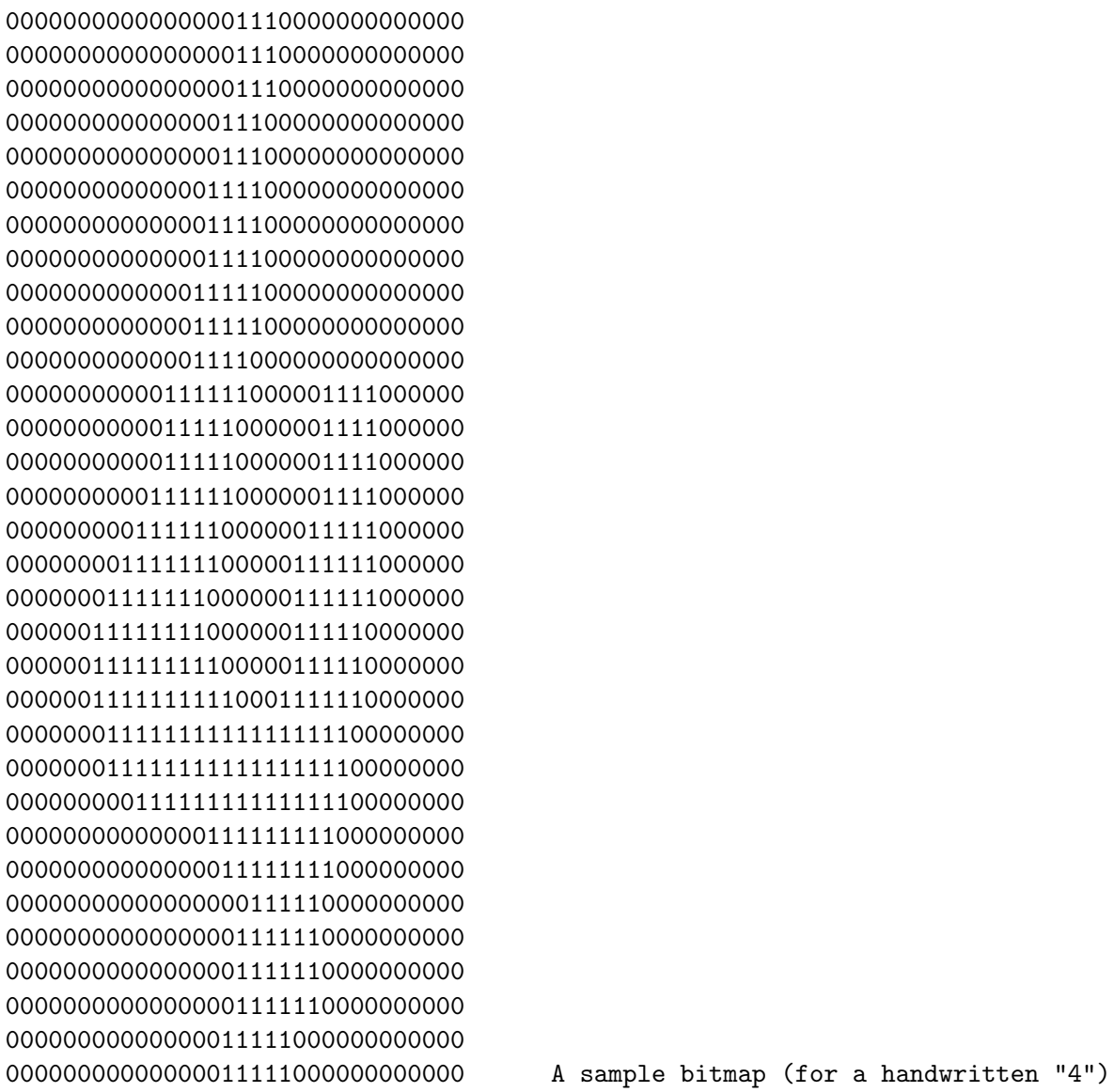
$$\sqrt{\sum_{i=1}^{n}(t_i - o_i)^2}$$

where n is the number of elements in each vector. So, the question we are trying to answer is what impact the training/test set proportion has on the measured accuracy of the resulting network.

Digit Recognition Examples

The original examples were 32 x 32 bitmap of 0s and 1s representing the image of the digit (see an example on the next page). The examples I am giving you are an 8 x 8 down-sampled images, which were created by dividing the 32 x 32 bitmap into 64 non-overlapping 4 x 4 blocks and creating a new 8 x 8 matrix whose elements are real values indicating the fraction of 1s (gray-scale value) in the corresponding 4 x 4 blocks in the original, i.e. the number of 1s divided by 16. I have given you a text file containing 5,620 examples. An example consists of two lists of numbers, each of which begins with an open parenthesis followed by a space and ends with a close parenthesis preceded by a

space. The first list consists of the 64 real numbers, described above, that represent the gray-scale values for the 8×8 discretization of the image. The second lists consists of 10 real values, exactly one of which is 1.0 (the rest are 0.0) that indicates which digit this discretization is an example of. If you think of this as an array, the index of the 1.0 is the digit in the image.



Notes

Code

Normally, I would insist that you code this in the object oriented paradigm. So, you might have a `Node` class, a `NeuralNet` class, an `Example` class, etc. Of course, you're welcome to do this, but you don't need to. I'm even fine with just one class! And arrays (or `ArrayList`s, if you're programming in Java), work fine for the weights, activation levels, etc.

Bias Node

You may need to have a *bias node* in the input layer. The bias node is an extra input node whose activation level is always 1. It should be connected to every output node and the weights on those connections should be trained just like the weights on connections between the other input nodes and output nodes. In the examples in the class slides, where there are two inputs and a single output, if you added a bias node, you could think of the weight on the edge from that node to the output node as the y-intercept for the line the perceptron is learning. (The weights indicate the line's slope.)

Initial Weights

Although the initial weights in the neural network should be random, I found that the range of those weights has a significant impact on the trainability of the network. I got good results initializing the weights to random doubles between -1.0 and 1.0, but students in the past have found it necessary to start with weights as small as -0.15 to 0.15.

Learning Rate and Number of Epochs

A learning rate of 0.01 and 1,000 epochs should give you good results on the more reasonable training-set/test-set splits (e.g. 70% for training and 30% for testing). If not, feel free to vary these.

Submitting Your Work

When you have completed the assignment, you should submit the following through Blackboard:

1. your code (all the files I need to run your program should be in a directory that includes a `README` file containing clear instructions about how to run your program), and

2. an electronic copy of a brief report describing your results.