The neural network designed for this assignment has its input layer organized in a 2D array, each neuron corresponding to one of the 8 by 8 input array elements. The 64 input neurons were each connected to a variable number of hidden neurons. The hidden neurons were each connected to the 10 output neurons, each corresponding to one of the 10 digit classifications. Roughly 90% classification was typical using 16-32 input node, not having tried more. Node thresholds were set to 0.5, half the possible output of any sigmoidal input function, to be distributed amongst the nodes’ input weights.

The network class has two methods which explain its functionality: Feedforward and Backpropagate. Feedforward brings the network into equilibrium with an input image, so its classification can be read from it. The inputs nodes’ inputs are set to the reduced dimensionality pixel values, in the range of 0-16. They pass these values to each hidden node, which adds all of its weighted inputs and computes its sigmoidal activation function. The output nodes’ input values are set to the weighted sum of the hidden nodes’ activation potential values. The output nodes compute their sigmoidal activation potentials, bringing the network to equilibrium. Backpropagate compares the network output to the known correct classification value and adjusts the weights accordingly. The formulas derived from the calculus relating the change in error with respect to weight presented in the lectures and textbook were used to update the weights.

The classification mean-squared-error (MSE) is calculated in a devoted training function associated with the network class. When a new minimum MSE is found, the network object is copied and thus saved. If too many training epochs pass without improvement on the minimum MSE, or if the current MSE become too large relative to the minimum, the network is restored to the best known configuration. Its weights are then perturbed by feeding-forward a random input image, and backpropagating to change the weights. The training process then resumes. The test function iterates through the set of test data and writes output to “output.txt”.

Upon creation of the network’s nodes, all weight values are randomly initialized in the range of -1–1. They are expected to decrease, for how many inputs the nodes had and the uniformly initialized threshold values of 0.5, but were allowed to change according to the formulas. As such, it is unlikely that performance is very dependent on the initial weights.

The error used to change the weight values during backpropagation took into account an error offset, so that weights would drive inputs beyond the threshold in the appropriate direction rather than stop only when correct classification had barely been achieved. The error offset value was set to 0.45 on either side of the threshold. The error offsets were implemented as an attribute of the network class.

A learning rate of 0.15 was typically used to fine tune the network during many training epochs, but it performance was not terribly dependent on any thing but unreasonably large (> 1) learning rates. Convergence was usually achieved rapidly for hidden layers of size 16-32 nodes, in that the sense that it resets to the best known configuration, as mentioned above, repeatedly, without being able to find better weight values often. “Rapidly”, meaning within tens of training epochs. The learning rate was implemented as an attribute of the network.

To keep the network weight changes going in some guess of what a good direction is, and to avoid getting stuck in local minima, a momentum factor was used during training. 10% of the last weight change was usually added to the current weight change, although other reasonable (<100%) weight values appeared to perform well. The momentum is controlled by an attribute of the network class.

The program can be run either by building the source in Asg2\_10006197/Asg2\_10006197/main.cpp, or executing the executable at Asg2\_10006197/Debug/Asg2\_10006197.exe; both directories contain the necessary data files. The network’s performance can be demonstrated by running the program with only several training epochs; greater than 85% classification can be achieved with just 2 training epochs. However, allowing more training epochs produces better performance. The number of training epochs and hidden nodes are controlled by global constants at the top of the source code. Over 96% classification was achieved using 64 hidden nodes, and 500 training epoch. An ‘output.txt’ file is left with the source code, but not with the executable, should you wish to generate one from scratch. The program uses only libraries standard to C++ with Visual Studio 2013 Community. As such, everything necessary to run the program should be included with this submission.