COM 312 - Neural Networks and Deep Learning

Assignment 2 – Training an Auto-Associator on the MNIST Database

In this Assignment, you will build on the first assignment to implement the Generalized Delta Rule (also called "back-propagation"). Using the same database as in Assignment 1 (the MNIST database), the connection architecture should follow Figure 1. Again, input to the network will come from 784 input nodes corresponding to the pixels of pictures of ten hand-drawn digit categories. The hidden layer may have a variable number of nodes, but ten is a good size to start with. The output layer of the network will have 785 nodes. This output layer is trained to reproduce the input to the network on 784 nodes (called an 'auto-associator,' with an extra node (depicted in gray) that will be trained to predict whether the network is looking at an odd or even number (assume 0 to be an even number, where the remainder of 0/2 = 0).

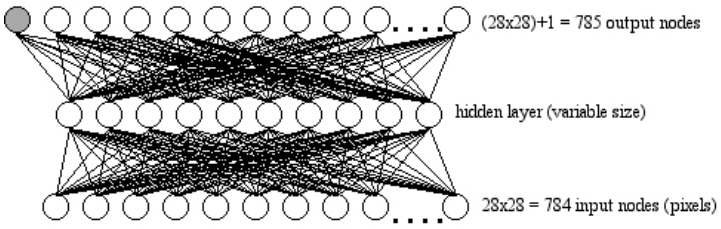

Figure 1: 'Auto-Associator' for the MNIST database

Start by initializing the weights of the network to small random values. Then train the network through training epochs composed of training cycles. A training cycle is where the network's weights are updated in response to an input stimulus. A training epoch is complete once each of the ten training cycles (or 60,000) have been trained on the network, one time each. Because a single training epoch will usually not result in the error being reduced to zero, multiple training epochs need to be run to reduce the error to as low as it will seem to go. Return or test the error after each training epoch to track error over training epochs.

During a training cycle, first calculate output for the network. Output for each node, *o*, is found following Equation 1, where *i* indicates node indexes in the input layer, *j* indicates node indexes of the hidden layer, and *k* indicates node indexes for the output layer. Again all nodes besides the input nodes need to include a bias term, θ. And again, a node's bias can be treated in the same way as a weight to the node with input that is always set to 1.0. Use the same squashing function, σ(), as in Assignment 1.

(1) $o_i = pixelValue$ $\quad o_j = \sigma\left(\theta_i + \sum w_{ij} \cdot o_i\right)$ $\quad o_k = \sigma\left(\theta_j + \sum w_{jk} \cdot o_j\right)$

Once output for the network is found, the error for the nodes in the hidden and output layers should be determined – following Equation 2 below. Like in Assignment 1, error for a unit in the output layer is found as the difference between the target, T, and actual output of the node. This error is then 'un-squashed' using the partial derivative of the squashing function, as: (o)*(1-o). The error is back-propagated to the hidden layer through the output weights, and then the error at the hidden layer is 'un-squashed' in the same way as at the output layer.

(2) $Err_k = (T_k - o_k) \cdot o_k \cdot (1 - o_k)$ $\quad Err_j = o_j \cdot (1 - o_j) \cdot \sum Err_k \cdot w_{jk}$

Now that error is found for all processing nodes, the weights to the nodes can be updated as before, based on the error.

(3) $\Delta w_{jk} = \eta \cdot o_j \cdot Err_k$ $\quad \Delta w_{ij} = \eta \cdot o_i \cdot Err_j$

What to do:

Code the network depicted in Figure 1 following the equations above. Train the network on the 60,000 training samples of the MNIST database per training epoch (loadType = 0). After each training epoch, run a testing epoch on the testing data (loadType = 1) and return the results per epoch. A testing epoch should sum the error for the 10,000 testing samples of the MNIST database, and then divide by 10,000 to give the average error. Run enough training-testing epochs that the average error per training-testing epoch reaches asymptote. As you are developing and debugging your code, you may want to use just the 10 training samples of Problem 1 in Assignment 1.

Deliverables:

1) Turn in your code (do not modify the rand.h or nmist.h libraries). Your code should compile and run on the classroom machines running linux. DO NOT provide a copy of rand.h or nmist.h or the MNIST database. (50 points)

2) Provide a written analysis of your results including graphs. Your analysis should not include noise on the input patterns (set noise to zero). You should explore changes with the learning rate and with using a different number of nodes in the hidden layer. You should also explore the influence of the extra odd-even category node. Specifically:

    - Try at least two different learning rates and plot graphs for those two different learning rates, using 10 hidden nodes. Report how low you were able to get the error to go, and report how many training epochs it took to achieve that level of learning.
      (20 points)

    - Try the two learning rates above but using a different number of hidden nodes (you decide how many hidden nodes to use). Did the different number of hidden nodes improve performance? Provide graphs to compare with the resulting graphs from above where you used 10 hidden nodes.
      (20 points)

    - What is the influence of the odd-even category node? Using the optimal performing network in your above simulations, what happens if you train that network without using the odd-even category node?
      (10 points)

Compress all of your files into a single .zip file that includes your name, the name of the class, and reference to Assignment 2. For example: COM312_Almaz_Ass2.zip. Also include you name in the header of your code and your name on the top of your written report.

DO NOT SUBMIT SEPARATE FILES AND DO NOT SUBMIT A .RAR FILE OR OTHER COMPRESSION FORMATS