Machine Learning (CS 419/519)
Assignment 02 (50 points)

Due 5:00 PM, Friday, 02 November 2018

In this assignment, you will implement a tool for building and training neural networks on some data extracted from image files. Your code will allow the user to specify the structure of one or more multi-layer feed-forward networks, and then will train those networks by back-propagation. As the tool is meant to allow a user to explore different possible data-sets and structures, it will be possible to do this process over and over again, as many times as the user chooses. The tool will also allow trained networks be saved to files and loaded again for later re-use.

For full points, your program will work as follows:

1. (*3 pts.*) The program will execute from the command line, and handle all input/output using standard system IO (in Java, e.g., this is `System.in` and `System.out`, and in C++ you would use `cin` and `cout`, etc.).

   When the program is executed, it will ask the user to choose between three options:

   (a) The user can train a new network.

   (b) The user can load a previously trained network from a data-file. You can assume that this file, if it exists, is contained in the same directory as the operating program.

   (c) The user can quit the program.

   Note: The assignment folder contains a sample runs of the program so you can see what is expected in this step and later steps; your output should be much the same, if not identical.

2. (*30 pts.*) When the user chooses to train a new network, the program will prompt the user for a data-resolution value. This will be some integer chosen from the set $\{5, 10, 15, 20\}$, corresponding to one of the four pairs of training and testing data supplied with this document. The program will then infer the size of the input and output layers of the network from the data-files chosen; the input layer size depends upon the resolution of the data, while the output layer size is always fixed.

   Next, the program will prompt the user for the number of hidden layers to be added to the network. You can assume this is a non-negative integer no greater than 10; note that it may in fact be 0 (resulting in a single-layer perceptron network). The program will then ask the user for the size of each layer, in order (top-down from input to output). You can assume that each layer-size value will be a positive integer no greater than 500.

   Once all layers are specified, the program will construct a fully-connected feed-forward network. Initially all weights on connections in the network, and the bias weight on each neuron, will be set to some random value in $[0.0, 1.0]$. The network will then use back-propagation to train the network as well as possible on the training set that has been selected. When implementing back propagation:

   - Use the standard logistic (Sigmoid) function for activation (output of a neuron).
   - Treat the α control parameter as equal to 1 (i.e., you can forget about it, since it won't factor into computations).

- Terminate the algorithm after either:
  (a) The number of iterations through the training set reaches 1,000.
  (b) The maximum error seen on any single pass through the data set is less than 0.01. Recall that error $(y_j - a_j)$ is calculated for each output neuron j, so this means that we have every neuron close to correct for every element of the training set.

  If you need to, you can play around with these stopping conditions to make the network train more quickly, or more slowly. Note that stopping too fast can reduce accuracy of the resulting network.

After the network is trained, it will test its accuracy by running each data-item from the corresponding training set through the network and assigning it to the category corresponding to the output node that has highest output value for that input. This result will then be checked against the correct output (where correctness means that the highest-valued output neuron is the one corresponding to the sole 1 in the output vector given). Accuracy will be reported on a percentage basis, accurate to a single decimal digit.

After testing, the program will ask the user if they wish to save the trained network to a file for possible re-use. If the user chooses to do so, the program will write data about the network to a file (see the next step for more information about this). Once the data is saved (or not, if the user chooses not to save it), the program repeats over again, once more prompting to train or load a network, or quit.

3. (*15 pts.*) If the user chooses to load a network, the program will prompt them for the name of a file containing a network specification; again, you may assume that any such file, if it exists, will reside in the same directory as your program.

   When handling this component (and the ability to save to a file mentioned above), you can decide upon the format of the file for yourself. A text-based format is fine, although you may choose a binary format if you like. Whatever format you choose, you must ensure that it contains all the information necessary to re-construct a learned network. That is, it must contain data about the size of each layer in the network, and about the learned values for all network connection and bias weights. It will need to be possible to read in this data and re-construct the corresponding network. When loading the network, the program should display the sizes of the various layers that comprise it.

   Once a previously learned network is loaded from data, it will be tested on the appropriate test set (this can be inferred by the program from the size of the network input layer). Accuracy will again be reported on a percentage basis, accurate to a single decimal digit. The program will then repeat over again, once more prompting to train or load a network, or quit.

4. (*2 pt.*) When the user chooses to quit the program, the process of building and testing networks is over. The program should wish the user goodbye, as this is the least one can do.

You will hand in a single compressed archive containing the following:

1. Complete source-code for your program. This can be written in any language you choose, but should follow proper software engineering principles for said language, including meaningful comments and appropriate code style. Code should be in its own folder within the archive, to keep it organized and separate from the rest of the submission.

2. A file containing a neural network trained and saved using your program, that can then be re-loaded by the program.

3. A `README` file that provides instructions for how the program can be run, and the name of the file containing the pre-trained network, along with the accuracy you achieved using that network on the supplied testing data.