CSCI-580 Fall 2014 Project 3 Due: November 19-21

In this project you will practice implementation of the *back-propagation* algorithm for artificial neural networks.

Input:

1. A structure of a *feed-forward neural network*.
2. Weights of links.
3. Training input data set and the corresponding output.
4. Testing input data set and the corresponding output.
5. The number of iterations to run the back-propagation algorithm.

Output:

1. The learned weights of the links from the first node of the input layer to the nodes in the hidden layer printed to the screen in this format
   <weight><space><weight><space>….<weight><space><endl>
   Use the following code to output weight values, so that your output will match mine:
   cout << showpoint << fixed << setprecision(12) << weight_value << " ";

2. The results of classification of testing data points printed to the standard output. For each data point in the testing data set, use the output vector $\boldsymbol{y} = [y_0, y_1, \dots, y_{n-1}]$ (the activation functions of the nodes in the output layer) to calculate the Euclidean distance between $\boldsymbol{y}$ and the encodings of the digits (see below) and classify the data point according to the minimum Euclidean distance (in case of ties, choose the smallest digit). The output of the classification are the digits that your program assigned to the testing data points. Output is in this format:
   <digit><endl>
3. The ***accuracy*** of the classifier printed to the screen and calculated by the following formula:

$$\frac{The\ number\ of\ correctly\ classified\ testing\ data\ points}{Total\ number\ of\ testing\ data\ points}$$

   The output format is:
   <accuracy><endl>
   Use this code to output accuracy:
   cout << showpoint << fixed << setprecision(12) << accuracy << endl;

Input files for your program:

- *train_input.txt* contains input vectors of the training data set;
- *train_output.txt* contains class labels for the data points of the training data set;
- *test_input.txt* contains input vectors of the testing data set;
- *test_output.txt* contains class labels for the data points of the testing data set;
- *structure.txt* contains the number of nodes (neurons) in each layer including the input and output layer;
- *weights.txt* contains the initial weights of all links of the neural networks between the nodes of any two consecutive layers (does not contain the dummy weights w_{0i}; assume that $w_{0i} = 0.01$ for each node *i*; assume that the nodes in the input layer do not have the dummy weights).

Format of the files:

train_input.txt (or *test_input.txt*)

```
0.9 0.1 0.1 0.9 0.9 0.9 0.9
0.9 0.9 0.1 0.1 0.1 0.9 0.9
0.9 0.9 0.1 0.1 0.1 0.9 0.9
0.9 0.9 0.1 0.1 0.1 0.9 0.9
```

train_output.txt (or *test_output.txt*)

```
2
3
5
4
```

Each row of the *train_input.txt (test_input.txt)* corresponds to an image of a digit; 0.9 corresponds to a white pixel and 0.1 corresponds to a black pixel. Thus, each row corresponds to an input vector of a single data point. The total number of numbers (0.1 or 0.9) in each row equals to the number of nodes in the input layer of a given neural network.

Each row of the *train_output.txt (test_output.txt)* corresponds to a digit. The total number of nodes in the output layer of a given neural network corresponds to the total number of possible digits that the neural network recognizes (starting from 0, 1, 2, 3, …).

Use the following encoding for the digits:

Digit	Output vector $\boldsymbol{y} = [y_0, y_1, y_2, y_3, y_4, y_5, y_6, y_7, y_8, y_9]$
0	$\boldsymbol{y} = [0.1,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9]$
1	$\boldsymbol{y} = [0.9,0.1,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9]$
2	$\boldsymbol{y} = [0.9,0.9,0.1,0.9,0.9,0.9,0.9,0.9,0.9,0.9]$
3	$\boldsymbol{y} = [0.9,0.9,0.9,0.1,0.9,0.9,0.9,0.9,0.9,0.9]$
4	$\boldsymbol{y} = [0.9,0.9,0.9,0.9,0.1,0.9,0.9,0.9,0.9,0.9]$
5	$\boldsymbol{y} = [0.9,0.9,0.9,0.9,0.9,0.1,0.9,0.9,0.9,0.9]$
6	$\boldsymbol{y} = [0.9,0.9,0.9,0.9,0.9,0.9,0.1,0.9,0.9,0.9]$
7	$\boldsymbol{y} = [0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.1,0.9,0.9]$
8	$\boldsymbol{y} = [0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.1,0.9]$
9	$\boldsymbol{y} = [0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.1]$

structure.txt

```
7
2
3
```

weights.txt

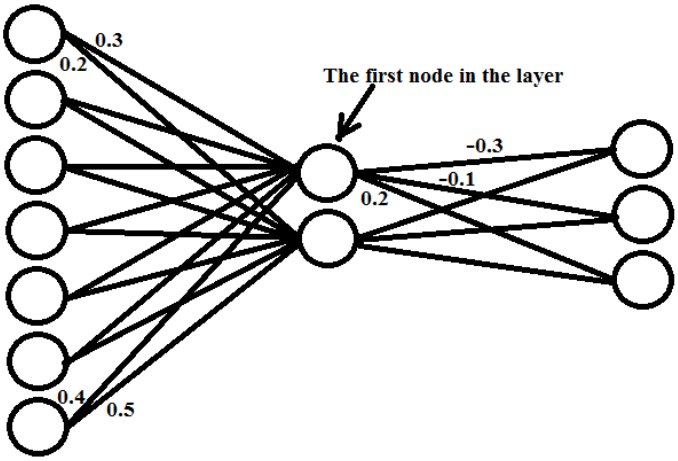
```
0.3   0.2
0.1   -0.1
-0.2  0.4
0.1   0.5
0.3   0.1
0.2   -0.1
0.4   0.5
-0.3  -0.1  0.2
0.4   0.3   -0.1
```

from the input layer's nodes to the hidden layer's nodes

from the hidden layer's nodes to the output layer's nodes

In this example, the given neural network has 7 nodes in the input layer, 2 nodes in a single hidden layer and 3 nodes in the output layer, which means that the given neural network can recognize 3 digits: 0, 1 and 2. This means that the output vector ***y*** has three attributes [0.1, 0.9, 0.9] to represent the digit 0; [0.9, 0.1, 0.9] to represent 1 and [0.9, 0.9, 0.1] to represent 2.

The numbers in each row in *weights.txt* correspond to the weights of the out-going links from a single node (see Figure below for ordering of the nodes and weights).



Command line to run your executable program ann:

./*ann* train_input.txt train_output.txt test_input.txt test_output.txt structure.txt weights.txt k

where *k* is an integer, the number of iterations to run the back-propagation algorithm.

Overall requirements of your program:

1. You need to write a class *ann* (artificial neural network; has two files: ann.h and ann.cpp) that stores the structure of a given neural network, has a function to run the back-propagation algorithm on the neural network (use alpha = 0.01), and has data structures to store intermediate results (activation function, errors, weights, etc.).
2. Your program must:
    - read the input files and initialize a neural network with the given structure and weights;
    - run the back-propagation algorithm to learn weights using the training data set;
    - classify the data points from the testing data set using the Euclidean distance;
    - calculate the accuracy of the classification;
    - output the results (see above).

Submission includes:

1. Submit the following files to ***turnin*** system: ann.h, ann.cpp and main.cpp.
2. Zipped code submitted via Blackboard.

Grading rubric:
1. Submission of code is 10%.
2. Program compiles and passes all tests: 90%. Program that does not compile receives 0; program that does not pass any tests receives 0.