CSCI-580 Spring 2017 Project 2

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 next layer (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 $\boldsymbol{a} = [a_0, a_1, \ldots, a_{n-1}]$ (the activation functions of the nodes in the output layer) to calculate the Euclidean distance between $\boldsymbol{a}$ 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 is 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:
  - each row corresponds to a single image;
  - there are 256 integers in a single row corresponding to grayscale 16 x 16 image;
  - use each integer out of 256 integers to initialize the activation function of a node in the input layer;
- *train_output.txt* contains class labels for the data points of the training data set:
  - each label corresponds to a digit from 0 to 9;
  - there are 14 examples for each digit;
  - total of images used is 140 = 14 * 10;
- *test_input.txt* contains input vectors of the testing data set:
  - each row corresponds to a single image of a digit from 0 to 9;
  - different images were used for test data set than for training data set;
  - there are six images for each digit in the test data set;
  - size of an image is 16 x 16, which results in 256 integers;
- *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 layers;

- *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).
- *encoding.txt* contains encodings for digits: one line per a digit; for all digits 0, 1, …, 9; each line contains 10 decimal values. The number of nodes in the output layer is 10 (must be same as the number of decimal values in each encoding for a digit).

Example of Format of the Files:

train_input.txt (or *test_input.txt*)

```
1 2 3 4 5 6 7
1 1 2 1 1 1 1
50 51 52 53 54 55 56
67 68 1 69 70 71 72
```

train_output.txt (or *test_output.txt*)

```
2
3
5
4
```

Each row of the *train_input.txt (test_input.txt)* corresponds to an 16 x 16 pixels image of a digit. Thus, each row corresponds to an input vector of a single data point. The total number of integers 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 (0, 1, 2, 3, 4, 5, 6, 7, 8, 9).

The total number of nodes in the output layer of a given neural network is fixed and is equal to 10.

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	$\boldsymbol{y} = [0.1,0.9,0.9,0.1,0.9,0.9,0.1,0.9,0.9,0.1]$
1	$\boldsymbol{y} = [0.9,0.1,0.9,0.9,0.1,0.9,0.9,0.1,0.9,0.9]$
2	$\boldsymbol{y} = [0.9,0.9,0.1,0.9,0.9,0.1,0.9,0.9,0.1,0.9]$
3	$\boldsymbol{y} = [0.1,0.1,0.9,0.9,0.1,0.1,0.9,0.9,0.1,0.1]$
4	$\boldsymbol{y} = [0.9,0.1,0.1,0.9,0.9,0.1,0.1,0.9,0.9,0.1]$
5	$\boldsymbol{y} = [0.9,0.9,0.1,0.1,0.9,0.9,0.1,0.1,0.9,0.9]$
6	$\boldsymbol{y} = [0.1,0.9,0.1,0.1,0.9,0.1,0.1,0.9,0.1,0.1]$
7	$\boldsymbol{y} = [0.9,0.1,0.1,0.9,0.1,0.1,0.9,0.1,0.1,0.9]$
8	$\boldsymbol{y} = [0.1,0.1,0.9,0.1,0.1,0.9,0.1,0.1,0.9,0.1]$
9	$\boldsymbol{y} = [0.1,0.9,0.9,0.9,0.1,0.1,0.1,0.9,0.9,0.9]$

structure.txt

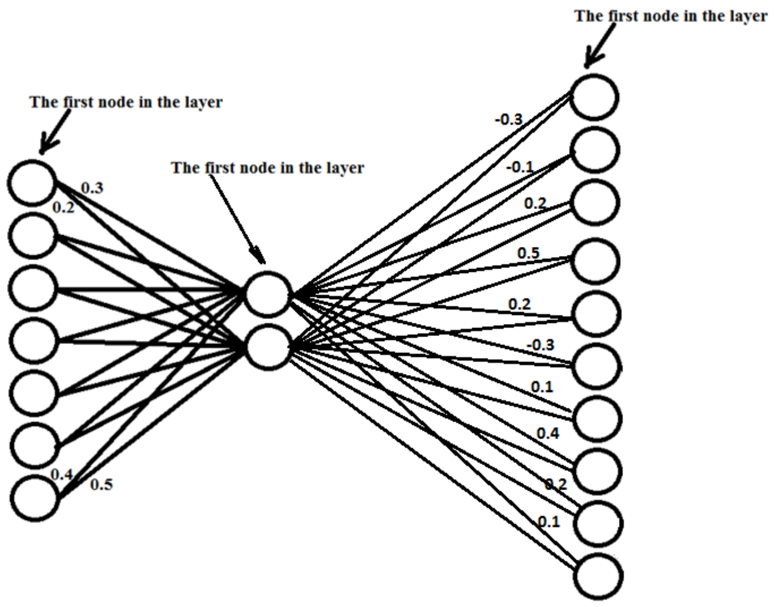
```
7
2
10
```

weights.txt

```
0.3  0.2
0.1  -0.1
-0.2 0.4
0.1  0.5     from the input layer's nodes
0.3  0.1     to the hidden layer's nodes
0.2  -0.1
0.4  0.5
-0.3 -0.1 0.2  0.5 0.2 -0.3 0.1 0.4 0.2 0.1   from the hidden layer's nodes
0.4  0.3  -0.1 0.4 0.1 -0.2 0.5 0.3 0.1 -0.2  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 10 nodes in the output layer.

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).


The first node in the layer
The first node in the layer
The first node in the layer
0.3
0.2
0.4
0.5
-0.3
-0.1
0.2
0.5
0.2
-0.3
0.1
0.4
0.2
0.1

After running a program on these files from the example above for 1 iteration, the output is the following:

```
0.299920387409 0.200029025566
5
5
5
5
0.250000000000
```

Command line to run your executable program called "run":

```
./run 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.

You can use the following commands to test your program on ecc-linux server:

```
./run $(cat tests/t01.cmd) > t01.my
diff t01.my tests/t01.out
vimdiff t01.my tests/t01.out
```

Additional information to match your output and output of the test files:

Use α = 0.01 in the back-propagation algorithm (a double).

Use ***long double*** for weights, alpha and all calculated values in 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 function(s) to run the back-propagation algorithm on the neural network, 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 for the given number of ***k*** iterations;
    - after this, the weights are fixed;
    - classify the data points from the testing data set using the Euclidean distance (do not change weights while processing testing data);
    - calculate the accuracy of classification;
    - output the results (see above).
3. Use ***long double*** in your calculations and α = 0.01.
4. Include your name in each of your files.

Submission includes:

1. Submit the following files to ***turnin*** system: ann.h, ann.cpp and main.cpp.
2. Zipped code submitted via Blackboard. Failure to submit code will result in 0pts for this project.

Grading rubric:
1. If a program does not compile on jaguar (or ecc-linux), it will get 0 pts.
2. If a program that does not pass any tests, it will receive 0 pts.
3. If program does not have the requested structure: ann.h, ann.cpp, and has most of the code placed inside main() function, 30% of the grade will be subtracted.