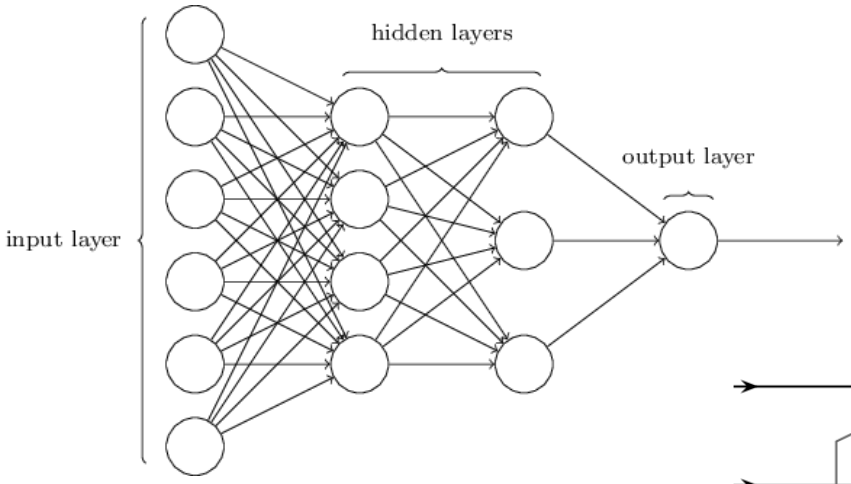
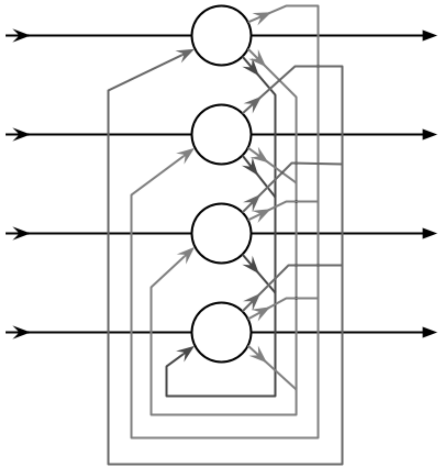
MNIST Digit Recognition

A case study of Perceptron & Hopfield Artificial Neural Networks





Chaise Brown

Will Carhart

May 18th, 2017

Table of Contents

Project Assertion

Project Background

Modern communication is founded in handwriting; humans have used handwriting for thousands of years to illustrate their thoughts. Even though there is a general form for each character or number in an alphabet, each human has a seemingly unique form of writing each. Despite this vast variety of handwritings, most humans can determine the difference between an '8' and a '2' without much thought.

In order for computers and machines to continue their advancements in intelligence, they will need to be able to communicate with humans. One of these key forms of communication is written characters. Thus, it is important for computers to be able to read different styles and forms of handwriting, yet it is accomplished by most children during elementary school.

We have developed an artificial neural network to recognize numbers written in different handwritings, just as a human brain does. Our project implements an application of image recognition, utilizing deep learning. At a high level, we create an Artificial Neural Network (ANN) that successfully classifies handwritten digits [0, 9]. Using the MNIST Dataset (see *Data*), we feed the ANN a series of images. These images, which are recorded using a grid of numbers representing the shade of each pizel, form a 28x28 matrix of input pixel values, and are processed by an array of 784 numbers. Once processed, the ANN will produce ten different outputs, with each output corresponding to a likelihood that the image is a specific digit. By having a separate output for each type of object we want to recognize, we can use an ANN to classify objects into groups. Thus, our training of the neural network will involve reading in the pixel arrays and classifying them as *0's* and *not 0's*, *1's* and *not 1's*, etc., so it learns to tell them apart. Ultimately, the ANN should be able to accept an image input vector, classify it as a digit in the rage [0, 9], and return a probability of the image being classified within the range – i.e. a given image of a handwritten *8* could be classified as 80% *8* and 20% *5*.

Purpose

The purpose of this project is to explore the applications of image recognition it two different machine learning algorithms: the Hopfield ANN and Perceptron ANN. We hope to determine the performance and architectural differences between Hopfield and Perceptron ANNs. We will build these ANNs to classify targets of the MNIST Dataset (see *Data*).

Correctness

To the extent of our knowledge after rigorous testing and experimentation, the implementation of our project is correct and accurate, per the guidelines and requirements of the assignment description.

Completeness

To the extent of our knowledge after rigorous testing and experimentation, the implementation of our project is complete, per the requirements specified in the assignment prompt. Our program's user interface and error handling is robust; there are no missing components of the project.

Data

Dataset Description

We used data from the MNIST Dataset provided by Yann LeCun, a professor at New York University, Corinna Cortes, a research scientist at Google Labs, and Christopher J.C. Burges, a research analyst at Microsoft. The MNIST Dataset contains 60,000 training samples and 10,000 testing samples of handwritten numbers. The numbers are in the range [0, 9], and have been collected from 500 different authors. In addition, all of the numbers in the data set have been self-normalized (grayscale, where 0 represents entirely white and 1 represents entirely black) and centered in a fixed-size image (28x28 pixels). The data set can be accessed at yann.lecun.com/exdb/mnist/ (see *References*).

Dataset Format

The MNIST Dataset consists of four files, as shown in Table 1.

Table 1: Files of MNIST Dataset

Filename	Size (bytes)	Description
`train-images-idx3-ubyte.gz`	9912422	Set of training images
`train-labels-idx1-ubyte.gz`	28881	Set of training targets
`t10k-images-idx3-ubyte.gz`	1648877	Set of testing images
`t10k-labels-idx1-ubyte.gz`	4542	Set of testing targets

Each of the four files uses an *.idx* format, which adheres to the following guidelines, as shown in Fig. 1.

magic number
size in dimension 0
size in dimension 1
size in dimension 2
……
size in dimension N
data

Figure 1: Example of .idx file format

The *magic number* is an integer (4 bytes, Big Endian, MSB first). The first two bytes are always 0. The third byte codes the type of data for the *.idx* file, as shown in Table 2.

Table 2: Magic Number 3rd Byte Value Mapping

Magic Number 3rd Byte Value	Data Type
0x08	unsigned byte
0x09	signed byte
0x0B	short (2 bytes)
0x0C	int (4 bytes)
0x0D	float (4 bytes)
0x0E	double (8 bytes)

The 4th byte of the *magic number* codes the number of dimensions in the input vector. In addition, the size in each dimension is an integer (4 bytes, Big Endian, MSB first).

Dataset Indexing

Each of the four files are indexed in the following fashion, as shown in Tables 3-6.

Table 3: `train-images-idx3-ubyte` *indexing*

Offset	Type	Value	Description
0000	32bit integer	*0x0000 0803 (2051)*	magic number
0004	32bit integer	*0x0000 EA60 (60000)*	number of images
0008	32bit integer	*0x0000 001C (28)*	number of rows
0012	32bit integer	*0x0000 001C (28)*	number of columns
0016	unsigned byte	*0x??*	pixel
0017	unsigned byte	*0x??*	pixel
……	……	……	……
xxxx	unsigned byte	*0x??*	pixel

Table 4: `train-labels-idx1-ubyte` *indexing*

Offset	Type	Value	Description
0000	32bit integer	*0x0000 0801 (2049)*	magic number
0004	32bit integer	*0x0000 EA60 (60000)*	number of items
0008	unsigned byte	*0x??*	label
0009	unsigned byte	*0x??*	label
……	……	……	……
xxxx	unsigned byte	*0x??*	label

Table 5: `t10k-images-idx3-ubyte` *indexing*

Offset	Type	Value	Description
0000	32bit integer	*0x0000 0803 (2051)*	magic number
0004	32bit integer	*0x0000 2710 (10000)*	number of images
0008	32bit integer	*0x0000 001C (28)*	number of rows
0012	32bit integer	*0x0000 001C (28)*	number of columns
0016	unsigned byte	*0x??*	pixel
0017	unsigned byte	*0x??*	pixel
……	……	……	……
xxxx	unsigned byte	*0x??*	pixel

Table 6: `t10k-labels-idx1-ubyte` *indexing*

Offset	Type	Value	Description
0000	32bit integer	*0x0000 0801 (2049)*	magic number
0004	32bit integer	*0x0000 2710 (10000)*	number of images
0008	unsigned byte	*0x??*	label
0009	unsigned byte	*0x??*	label
……	……	……	……
xxxx	unsigned byte	*0x??*	label

Dataset Conversion

In order to reference our data in a more easily accessible format, we converted the *.idx* format to *.csv*, or a ***C****omma* ***S****eparated* ***V****alues* file, which, as the name portrays, separates each data point with a comma.

We used the algorithm provided by Joseph Redmon, which can be found at pjreddie.com/projects/mnist-in-csv/ (see *References*). Redmon's algorithm transforms our data into the following format, as shown in Fig. 2.

Label1	*Pixel1,1*	*Pixel1,2*	*Pixel1,3*	……	*Pixel1,784*
Label2	*Pixel2,1*	*Pixel2,2*	*Pixel3,2*	……	*Pixel2,784*
……	……	……	……	……	……
Label60000	*Pixel60000,1*	*Pixel60000,2*	*Pixel60000,3*	……	*Pixel60000,784*

Figure 2: Example of .csv file format

The source code for this algorithm is listed in full in *Appendix, Source Code, IDX-to-CSV Converter*.

Implementations Details

Hopfield Artificial Neural Network

Architecture Description

A Hopfield ANN is an auto-associative neural network popularized by John Hopfield in 1982. Hopfield ANNs mimic the memorization abilities of the human brain, but with either binary or bipolar nodes. While testing on a given input pattern, Hopfield ANNs are guaranteed to converge, but will not always converge to a known pattern.

Implementation Steps

At a high level, our implementation adheres to the following steps:

1. Read in all training settings from the user.
2. Read in the training data from a user-specified input file.
3. Train the Hopfield ANN using the Hopfield learning method, which entails constructing a weighted matrix and altering its values based on each training input.
4. During training, compute and save the average input matrix for each of the training targets.
5. Once training has been completed, save the final weighted matrix to an external file for later use.
6. Upon user input, test specific cases using the trained Hopfield ANN, which entails starting with the ANN's saved weighted matrix and altering its values based on each testing input.
7. Once training has converged, classify test cases based on the result of the Hopfield ANN and saved average input matrices.
8. Output classification results to a user-specified file.

Method Details

Our implementation was broken down in to the following methods, which are all controlled and called from `main()`. The method details are as follows:

```
public static String[] train(String imageTrainingFile, String
savedWeightsFile)
```

The `train()` method implements the training of the Hopfield ANN. The method is `static` because it is called from `main()`. The method returns `void` because it trains the Hopfield ANN and does not need to return anything. The parameter `String imageTrainingFile` refers to the name of the input file that contains the training data for the Hopfield ANN and the parameter `String savedWeightsFile` refers to the name of the output file to which the computed weighted matrix will be saved. The implementation of the method adheres to the following steps:

1. Open an input stream to `imageTrainingFile`.
2. Read in the dimensionality of the image vectors and the total number of training image vectors t.
3. For each of the t training image vectors, construct a matrix m to match the input data.

4. Compute the matrix ω, such that:

$$\omega = \sum_{i=1}^{t} m_i \cdot m_i^T$$

5. Set the diagonal of ω to 0, such that for each element ω_{ij} of ω, $\omega_{ij} = 0$ if $i = j$.
6. For each of the n_i training targets, compute the average matrix for each training target α_i, such that:

$$\alpha_i = \frac{1}{n_i} \sum_{i=1}^{n_i} m_i \cdot m_i^T$$

7. Open an output stream to each of the n_i average matrix output files and save each α_i.
8. Open an output stream to `savedWeightsFile` and save the computed weighted matrix ω to `savedWeightsFile`.
9. Close all input and output streams.

```
public static void test(String imageTestingFile, String
savedWeightsFile, String resultsFile, double steepness, double
errorThreshold)
```

The `test()` method implements the testing of the trained Hopfield ANN. The method is `static` because it is called from `main()`. The method returns `void` because it tests the trained Hopfield ANN and does not need to return anything. The parameter `String imageTestingfile` refers to the name of the input file that contains the testing data for the trained Hopfield ANN, the parameter `String savedWeightsFile` refers to the input file that contains the saved weighted matrix from the training of the Hopfield ANN, the parameter `String resultsFile` refers to the name of the output file to which the results of the testing classification will be written, the parameter `double steepness` represents the steepness parameter used in the activation function, and the parameter `double errorThreshold` refers to the value of acceptable error to declare the ANN as converged during testing. The implementation of this method adheres to the following steps:

1. Open an input stream to `imageTestingFile`.
2. Read in the dimensionality of the image vectors n and the total number of testing vectors t.
3. Read the set of testing matrices as W_{test}.
4. Create an array of integers A from 0 to $n - 1$.
5. Call `weightMatrix()` to acquire the saved weighted matrix ω.
6. For each $\omega_{test} \in W_{test}$, iterate through each $\omega_{ij,test} \in \omega_{test}$ and record each $\omega_{ij,test}$ into a two-dimensional array.
7. Check to see if the product matrix ω has converged. If it has converged, continue, else, update the weights in ω using the following steps:
    a. Select a random element $a_i \in A$.
    b. Iterate through each element $m_{ij} \in \omega$. Compute s using the following formula:

$$s = \omega_{ij,test} * m_{ij} * a_i$$

c. Compute y_{in} using the following formula:

$$y_{in} = \omega_{a_i,test} + s$$

d. Compute y using the following formula:

$$y = f_A(y_{in}) = \frac{2}{1 + e^{-\sigma y_{in}}} - 1$$

8. Compute the number of epochs required for testing to converge.
9. Save results to `resultsFile` by calling `saveResults()`.
10. Close the input stream.

```
public static void saveWeights(double[][] weightMatrix, String
savedWeightsFile)
```

The `saveWeights()` method saves the weighted matrix of the trained Hopfield ANN to a user-specified output file. The method is `static` because it is called from `main()`. The method returns `void` because it saved the computed weighted matrix and does not need to return anything. The parameter `double[][] weightMatrix` represents the weighted matrix for the Hopfield ANN and the parameter `String savedWeightsFile` refers to the name of the output file to which the computed weight matrix will be saved. The implementation of this method adheres to the following steps:

1. Open an output stream to `savedWeightsFile`.
2. Iterate though each element ω_{ij} of `weightMatrix` and write it to `savedWeightsFile`.
3. Close the output stream.

```
public static double[][] weightMatrix(String savedWeightsFile, int
image_dim)
```

The `weightMatrix()` method copies the weights from a user-specified input file to a two-dimensional array, which is then used to represent the weighted matrix for the Hopfield ANN. The method is `static` because it is called from `main()`. The method returns a `double[][]` that represents the saved weighted matrix from the trained Hopfield ANN. The parameter `String savedWeightsFile` refers to the name of the input file that contains the saved weighted matrix from the trained Hopfield ANN and the parameter `int image_dim` refers the dimensionality of the weighted matrix. The implementation of this method adheres to the following steps:

1. Open an input stream to `savedWeightsFile`.
2. Iterate through each element in `savedWeightsFile` and store it as a weighted matrix, represented by a two-dimensional array of size `image_dim`.
3. Close the input stream.

```
public static double activationFunction(double yin, double steepness)
```

The `activationFunction()` method computes the activation for a given neuron, using a *Bipolar Sigmoid Activation Function*. The method is `static` because it is called from `main()`. The method returns a `double` that represents the computed activation value for the given input. The parameter `double yin` represents the input to the given neuron and the parameter `double steepness`

represents the steepness parameter used in the *Bipolar Sigmoid* function. The implementation of this method adheres to the following step:

1. Compute the activation with the following function:

$$y = f_A(y_{in}) = \frac{2}{1 + e^{-\sigma y_{in}}} - 1$$

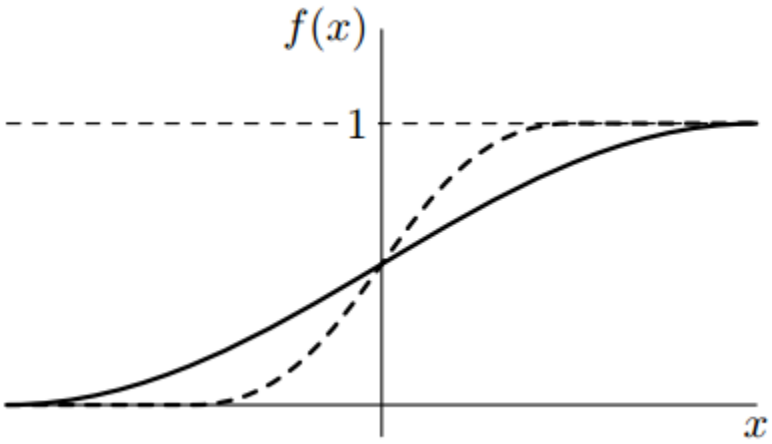


Figure 3: Plot of Binary Sigmoid Function

`public static void displayUserMenu()`

The `displayMenu()` method displays the user interface menu. The method is `static` because it is called from `main()`. The method returns `void` because it displays the menu and does not need to return anything. The method has no input parameters. The implementation of this method adheres to the following step:

1. Display the user interface menu.

`public static void saveResults(String resultsFile, ArrayList<Double> y, int count, double errorThreshold)`

The `saveResults()` method saves the classified results from the trained Hopfield ANN to a user-specified output file. The method is `static` because it is called from `main()`. The method returns `void` because it writes the output to a file and does not need to return anything. The parameter `String resultFile` refers to the name of the file to which the results will be written, the parameter `ArrayList<Double> y` represents the calculated classification results of the trained ANN, the parameter `int count` represents the index of the training sample that is currently being saved, and the parameter `double errorThreshold` represents the error threshold. The implementation of this method adheres to the following steps:

1. If count is 1, open an output stream to `resultsFile` and clear the file.
2. Close the output stream, re-open an output stream to `resultsFile`.
3. For each test case, open an input stream to the specific average matrix α_i for that test case.

4. Compute the amount of error for each pixel of the given test case $\beta_i \in y_i$ by computing:

$$Error = abs(\beta_i - \alpha_{i,i})$$

5. If this error is less then `errorThreshold`, then add one to a count for that given testing target.
6. After all pixel errors have been computed, output the number of correct pixels for each test as a percentage, with the total number of pixels being 784 (28x28).
7. Output classification results to `resultsFile`.
8. Close all input and output streams.

Perceptron Artificial Neural Network

Architecture Description

A Perceptron ANN is a supervised learning algorithm that can interpret an input vector and classify it as an output vector. The algorithm was invented in 1957 by Frank Rosenblatt. The algorithm works with a specific number of artificial input neurons, which map to a specific number of artificial output neurons. The algorithm can also contain multiple layers of hidden (internal) artificial neurons, but it is not always required.

Implementation Steps

At a high level, our implementation adheres to the following steps:

1. Read in all training settings from the user.
2. Read in training data from a user-specified input file.
3. Train the artificial neural network using the Perceptron learning method.
4. Once training has converged, save final neuron weights to an external file for later use.
5. Upon user input, read in testing data from a user-specified input file.
6. Test specific cases using the trained Hopfield ANN.
7. Classify test cases based on the trained Hopfield ANN.
8. Output classification results to a user-specified results file.

Method Details

Our implementation was broken down into the following methods, which are all controlled and called from `main()`. The method details are as follows:

```
public static void train(String trainingfile)
```

The `train()` method implements the training of the ANN. The method is `static` because it is called from `main()`. The method returns `void` because it trains the ANN and does not need to return anything. The parameter `String trainingfile` refers to the name of the input training file. The implementation of this method adheres to the following steps:

1. Read in the size of the data set, output dimension, and number of training samples from a user-specified input file.
2. Initialize all neuron weights and other initial values per user-specified values.

3. Read in each training sample one at a time, compute the value y_{in} using the following equation:

$$y_{in} = w_b + \sum_{i=1}^{n} w_i x_i$$

4. Compute the value y based on the following activation function:

$$y = f_A(y_{in})$$

$$f_A = \begin{cases} 1, & if\ y_{in} > \theta \\ 0, & if\ y_{in} = \theta \\ -1, & if\ y_{in} < \theta \end{cases}$$

5. Read in each of the targets that correspond for each training sample.
6. If the output of the activation function $y = f_A(y_{in})$ does not equal the corresponding target, then update the neuron weights using the following formulae:

$$w_i(new) = w_i(old) + \alpha t_i x_i$$

$$w_b(new) = w_b(old) + \alpha t_i$$

7. Continue until an epoch is completed without any neuron weights being updated, or if the number of epochs exceeds the maximum specified by the user.
8. Final neuron weights are then saved to a user-specified output file.

```
public static void test(String trainedweights)
```

The `test()` method implements the deployment/testing of the trained Hopfield ANN. The method is `static` because it is called from `main()`. The method returns `void` because it implements deployment/testing and does not need to return anything. The parameter `String trainedWeights` refers to the name of the input file that contains the saved trained weights from a previous training. The implementation of this method adheres to the following steps:

1. Read in the saved trained weights from `trainedweights`.
2. Read in the name of the testing input file from the user.
3. Read in the testing and deployment settings from the user.
4. Open an input stream from the testing input file and read in each of the testing samples one at a time.
5. Read in the expected target that corresponds to each testing sample.
6. For each testing sample, compute the value y_{in} using the following equation:

$$y_{in} = w_b + \sum_{i=1}^{n} w_i x_i$$

7. Compute the value y based on the following activation function:

$$y = f_A(y_{in})$$

$$f_A = \begin{cases} 1, & if\ y_{in} > \theta \\ 0, & if\ y_{in} = \theta \\ -1, & if\ y_{in} < \theta \end{cases}$$

8. Store the value of the output of the activation function $y = f_A(y_{in})$ to the user-specified results file.

```
public static void output(int[] results, int[] targets, int
output_dim, String testingresults, int num)
```

The `output()` method outputs the classification results from trained ANN to a user-specified results file. The method is `static` because it is called from `main()`. The method returns `void` because it outputs to a results file and does not need to return anything. The parameter `int[] results` refers to the classification results from the trained artificial neural network, the parameter `int[] targets` refers to the expected output of the testing set, the parameter `int output_dim` refers to the number of output dimensions, the parameter `String testingresults` refers to the name of the output file to which the method will write, and the parameter `int num` refers to what testing index the method is currently writing to the output file. The implementation of this method adheres to the following steps:

1. If `num` is zero, clear the `testingresults` file. If `num` is not zero, open a new output stream to `testingresults`.
2. Iterate through results to determine which of the elements is a one, which denotes a successful classification. Store the index and use a switch block to correctly classify the result as a letter.
3. Iterate through target to determine which of the elements is a one, which denotes the correct letter. Store the index and use a switch block to correctly identify the letter.
4. Write the classified and actual results to the user-specified results file, with the proper formatting.
5. Close the output stream.

```
public static void statistics(String testingresults)
```

The `statistics()` method computes the testing statistics of the trained ANN after testing has been completed. The method is `static` because it is called from main(). The method returns void because it computes and outputs the testing statistics and does not need to return anything. The parameter `String testingresults` refers to the name of the input file from which the testing classification results are read. The implementation of this method adheres to the following steps:

1. Open an input stream to `testingresults`.
2. Iterate through `testingresults` and calculate the number of correctly classified results.
3. Output the calculated statistics to the console.
4. Close the input stream.

```
public static int activationFunction(int yin, double threshold)
```

The `activationFunction()` method computes the output of the bipolar activation function based on the given input. The method is `static` because it is called from `main()`. The method returns an `int` that represents the output of the activation function. The parameter `int yin` refers to the input to the activation function and the parameter `double threshold` refers to the threshold value that when achieved will cause the neuron to fire. The implementation of this method adheres to the following step:

1. Return a value based on following activation function:

$$f_A = \begin{cases} 1, & if\ y_{in} > \theta \\ 0, & if\ y_{in} = \theta \\ -1, & if\ y_{in} < \theta \end{cases}$$

```
public static void saveWeights(int input_dim, int output_dim,
double[][] weights, String destination)
```

The `saveWeights()` method saves the finalized neuron weights after training has converged to a user-specified output file. The method is `static` because it is called from `main()`. The method returns `void` because it writes the neuron weights to an output file and does not need to return anything. The parameter `int input_dim` represents the size of the data array, the parameter `int output_dim` represents the dimensions of the output, the parameter `double[][] weights` refers to the two-dimensional array that represents the finalized neuron weights (the first dimension is for each output dimension, the second dimension is for the list of neuron weights for each output dimension), and the parameter `String destination` refers to the desired output file. The implementation of this method adheres to the following steps:

1. Open an output stream to the user-specified output file.
2. Write the parameters of the edge weights, including the size of the data array and the number of output dimensions, to the output file.
3. Iterate through `weights[][]` and write each of the edge weights to the output file.
4. Close the output stream and return.

How do Hopfield ANNs differ from Perceptron ANNs?

Hopfield ANN

- Trained all at once
- Convergence at testing
- Guaranteed to converge
- Can converge to false pattern
- Matrix-based implementation
- Trained via *Hebb Learning Rule*
- Used primarily for *Pattern Association (PA)*

Perceptron ANN

- Trained neuron-by-neuron
- Convergence at training
- Not guaranteed to converge
- If converges, will converge to known pattern
- Neuron-based implementation
- Trained via *Hebb Learning Rule*
- Used primarily for *Pattern Classification (PC)*

Compilation & Execution Instructions

Hopfield Artificial Neural Network

Compilation

To compile the *Hopfield ANN* program, use the following command from the command line:

```
javac fpHopfield.java
```

To run the compiled program, use the following from the command line:

```
java fpHopfield.java
```

The program can also be compiled and run with an IDE, such as Eclipse. We used Eclipse to write and test the program, but it can also be tested and used in a Linux environment.

Execution

Once the program begins execution, it will prompt the user for a variety of information, including:

1. *Initial input* – a number that represents the desired menu choice:
    a. 1: Train the Hopfield ANN
    b. 2: Test the trained Hopfield ANN
    c. 3: Exit the program
2. For training:
    a. *The name of the training data file*
    b. *The name of the file to which weights will be saved*
3. For testing:
    a. *The name of the testing data file*
    b. *The name of the saved weights file*
    c. *The name of the file to which results will be written*
    d. *Sigma* – the steepness parameter
    e. *The error threshold*

Supplemental Files & Resources

The program needs a few files to complete its computations:

1. *mnist_train.csv* – A set of training samples to train the Perceptron ANN
2. *mnist_test.csv* – A set of testing examples to test the trained Perceptron ANN
3. *weights.txt* – An empty *.txt* file to store the computed neuron weights from training
4. *results.txt* – An empty *.txt* file to store the classification results of the trained Perceptron ANN

Perceptron Artificial Neural Network

Compilation

To compile the *Perceptron ANN* program, use the following command from the command line:

```
javac fpPerceptron.java
```

To run the compiled program, use the following from the command line:

```
java fpPerceptron.java
```

The program can also be compiled and run with an IDE, such as Eclipse. We used Eclipse to write and test the program, but it can also be tested and used in a Linux environment.

Execution

Once the program begins execution, it will prompt the user for a variety of information, including:

1. *Initial input* – a number that represents the desired menu choice:
    a. 1: Train the Perceptron ANN using a data file
    b. 2: Train the Perceptron ANN using a trained weight settings data file
2. For training with a data file:
    a. *The name of the training data file*
    b. *A choice to initialize weights to 0 or a random number* $\in [-0.5, 0.5]$.
    c. *The maximum number of training epochs*
    d. *The name of file to which computed weights will be saved*
    e. *Alpha* – the learning rate
    f. *Theta* – the activation threshold
    g. *An input to continue with testing or quit*
    h. *The name of the testing data file*
    i. *The name of the file to which results will be saved*
    j. *Theta* – the activation threshold
3. For training from a saved weight settings data file
    a. *An input to continue with testing or quit*
    b. *The name of the testing data file*
    c. *The name of the file to which results will be saved*
    d. *Theta* – the activation threshold

Supplemental Files & Resources

The program needs a few files to complete its computations:

5. *mnist_train.csv* – A set of training samples to train the Perceptron ANN
6. *mnist_test.csv* – A set of testing examples to test the trained Perceptron ANN
7. *weights.txt* – An empty *.txt* file to store the computed neuron weights from training
8. *results.txt* – An empty *.txt* file to store the classification results of the trained Perceptron ANN

Testing & Experimentation

Hopfield Artificial Neural Network

Error Threshold & Activation Steepness Parameter Testing

We will vary the error threshold and the activation steepness parameter (*sigma*), and measure how the Hopfield ANN's performance responds.

Perceptron Artificial Neural Network

General Testing

We will test the general performance of the Perceptron ANN, using optimal values for alpha and theta to maximize the ANN's memorization abilities.

Identity Testing

To prove the Perceptron ANN's correctness, we will train and test using the same set.

Alpha & Theta Variance Testing

We will vary the learning rate (*alpha*) and the activation threshold (*theta*), and measure how the Perceptron ANN's performance responds.

Results

Hopfield Artificial Neural Network

Error Threshold & Activation Steepness Parameter Testing

Table 7: Error Threshold & Activation Steepness Parameter Variance Testing Data

Target Value	*Activation Steepness*	*Error Threshold*	*Classification Success Rate*
0	0.01	0.01	0.0 %
		0.1	0.0 %
		0.5	0.0 %
		1.0	57.99 %
	0.1	0.01	8.0 %
		0.1	0.0 %
		0.5	18.0 %
		1.0	84.0 %
	1.0	0.01	48.0 %
		0.1	70.0 %
		0.5	46.0 %
		1.0	47.99 %
1	0.01	0.01	0.0 %
		0.1	0.0 %
		0.5	0.0 %
		1.0	86. 0 %
	0.1	0.01	0.0 %
		0.1	0.0 %
		0.5	0.0 %
		1.0	92.0 %
	1.0	0.01	0.0 %
		0.1	0.0 %
		0.5	100.0 %
		1.0	86.0 %
2	0.01	0.01	100.0 %
		0.1	100.0 %
		0.5	100.0 %
		1.0	64.0 %
	0.1	0.01	80.0 %
		0.1	100.0 %
		0.5	100.0 %
		1.0	72.0 %
	1.0	0.01	18.0 %
		0.1	68.0 %
		0.5	26.0 %
		1.0	64.0 %
		0.01	0.0 %

3	0.01	0.1	0.0 %
		0.5	0.0 %
		1.0	56.00 %
	0.1	0.01	0.0 %
		0.1	0.0 %
		0.5	0.0 %
		1.0	80.0 %
	1.0	0.01	14.0 %
		0.1	36.0 %
		0.5	48.0 %
		1.0	56.00 %
4	0.01	0.01	0.0 %
		0.1	0.0 %
		0.5	0.0 %
		1.0	74.0 %
	0.1	0.01	0.0 %
		0.1	0.0 %
		0.5	0.0 %
		1.0	84.0 %
	1.0	0.01	32.0 %
		0.1	32.0 %
		0.5	74.0 %
		1.0	74.0 %
5	0.01	0.01	0.0 %
		0.1	0.0 %
		0.5	0.0 %
		1.0	62.0 %
	0.1	0.01	20.0 %
		0.1	0.0 %
		0.5	0.0 %
		1.0	70.0 %
	1.0	0.01	14.0 %
		0.1	40.0 %
		0.5	14.0 %
		1.0	62.0 %
6	0.01	0.01	0.0 %
		0.1	0.0 %
		0.5	0.0 %
		1.0	48.0 %
	0.1	0.01	0.0 %
		0.1	0.0 %
		0.5	0.0 %
		1.0	80.0 %
	1.0	0.01	38.0 %
		0.1	8.0 %
		0.5	66.0 %

		1.0	48.0 %
7	0.01	0.01	0.0 %
		0.1	0.0 %
		0.5	0.0 %
		1.0	66.0 %
	0.1	0.01	0.0 %
		0.1	0.0 %
		0.5	0.0 %
		1.0	78.0 %
	1.0	0.01	18.0 %
		0.1	2.0 %
		0.5	56.0 %
		1.0	66.0 %
8	0.01	0.01	0.0 %
		0.1	0.0 %
		0.5	0.0 %
		1.0	46.0 %
	0.1	0.01	0.0 %
		0.1	0.0 %
		0.5	0.0 %
		1.0	70.0 %
	1.0	0.01	22.0 %
		0.1	4.0 %
		0.5	34.0 %
		1.0	46.0 %
9	0.01	0.01	0.0 %
		0.1	0.0 %
		0.5	0.0 %
		1.0	46.0 %
	0.1	0.01	0.0 %
		0.1	0.0 %
		0.5	0.0 %
		1.0	74.0 %
	1.0	0.01	2.0 %
		0.1	0.0 %
		0.5	60.0 %
		1.0	46.0 %

Perceptron Artificial Neural Network

General Testing

Table 8: General tests showing artificial neural network classification success

Initial weights setting	*Learning Rate (Alpha)*	*Activation Threshold (Theta)*	*# of Trials*	*# of Successes*	*Success Rate*	*Convergence Speed (# of Epochs)*
0	1	0	500	400	80.00 %	120
Random, $\in [-0.5, 0.5]$	1	0	500	425	approx. 85.00 %	136 - 140

Identity Testing

Table 9: Training and testing the artificial neural network with the training set

Initial weights setting	*# of Trials*	*# of Successes*	*Success Rate*
0	500	500	100.00 %
Random, $\in [-0.5, 0.5]$	500	500	100.00 %

Alpha & Theta Variance Testing

Table 10: Alpha & theta variance tests with weights initialized to zero

Learning Rate (Alpha)	*Activation Threshold (Theta)*	*Convergence Speed (# of Epochs)*	*# of Trials*	*# of Successes*	*Success Rate*	*Average Success Rate*
0.25	0.00	170	500	372	74.40 %	76.33 %
	0.25	181	500	392	78.40 %	
	1.00	214	500	366	73.20 %	
	5.00	175	500	367	73.40 %	
	10.00	191	500	399	79.80 %	
	50.00	186	500	394	78.80 %	
0.50	0.00	149	500	386	77.20 %	78.78 %
	0.25	141	500	424	84.50%	
	1.00	149	500	399	79.80 %	
	5.00	149	500	385	77.00 %	
	10.00	145	500	394	78.80 %	
	50.00	144	500	377	75.40 %	
0.75	0.00	125	500	413	82.60 %	81.40 %
	0.25	137	500	415	83.00 %	
	1.00	153	500	414	82.80 %	
	5.00	128	500	410	82.00 %	
	10.00	137	500	394	78.80 %	
	50.00	149	500	397	79.40 %	
1.00	0.00	120	500	400	80.00 %	83.33 %
	0.25	131	500	430	86.00 %	
	1.00	130	500	421	84.20 %	
	5.00	140	500	413	82.60 %	
	10.00	138	500	419	83.80 %	
	50.00	135	500	417	83.40 %	

Table 11: Alpha & theta variance tests with weights initialized to a random value $\in [-0.5, 0.5]$

Learning Rate (Alpha)	*Activation Threshold (Theta)*	*Convergence Speed (# of Epochs)*	*# of Trials*	*# of Successes*	*Success Rate*	*Average Success Rate*
0.25	0.00	134	500	359	71.80 %	75.63 %
	0.25	139	500	403	80.6 %	
	1.00	145	500	364	72.80 %	
	5.00	125	500	381	76.20 %	
	10.00	115	500	395	79.00 %	
	50.00	117	500	367	73.40 %	
0.50	0.00	151	500	395	79.00 %	77.10%
	0.25	150	500	387	77.4 %	
	1.00	136	500	380	76.0 %	
	5.00	155	500	383	76.6 %	
	10.00	138	500	408	81.6 %	
	50.00	149	500	360	72.0 %	
0.75	0.00	142	500	406	81.20 %	81.13 %
	0.25	151	500	403	80.60 %	
	1.00	165	500	410	82.00 %	
	5.00	129	500	413	82.60 %	
	10.00	149	500	408	81.60 %	
	50.00	140	500	394	78.80 %	
1.00	0.00	127	500	425	85.00 %	83.83 %
	0.25	143	500	419	83.80 %	
	1.00	156	500	417	83.40 %	
	5.00	149	500	418	83.60 %	
	10.00	153	500	421	84.20 %	
	50.00	132	500	415	83.00 %	

Conclusions

Hopfield Artificial Neural Network

Error Threshold & Activation Steepness Parameter Testing

Despite rigorous testing, there were no noticeable trends with the Hopfield ANN, regardless of what settings the ANN was tested with. We varied both the Activation Steepness and the Error Threshold, and saw no visual trend. As the Activation Steepness increased, there was no noticeable trendline with classification success. The average classification success for each activation steepness value was roughly the same, with an activation steepness of 1.0 causing a dramatic spread of success rates (See Fig. 4).

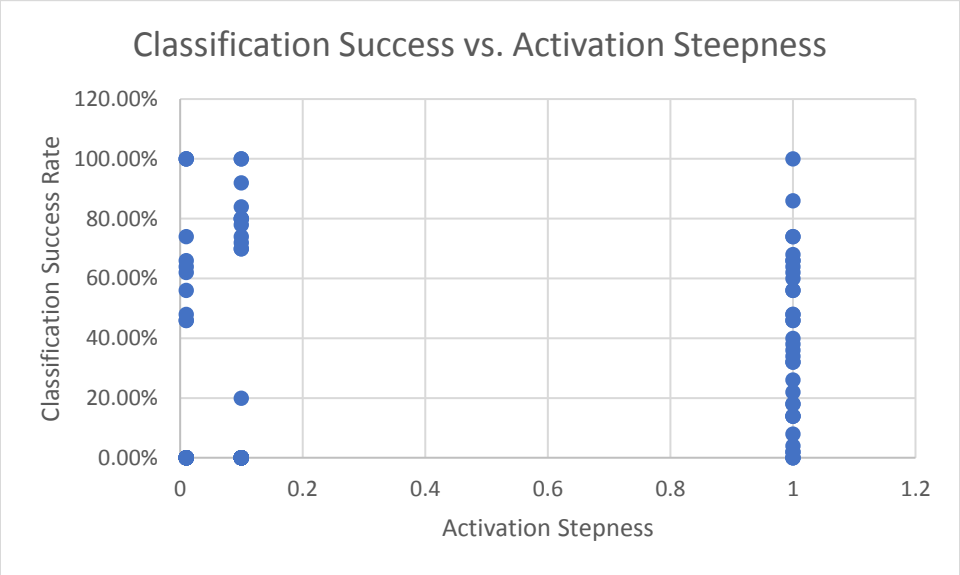


Figure 4: Hopfield ANN Classification Success vs. Activation Steepness

A similar observation can be made about the relationship between classification success and the error threshold. The accuracy of the ANN did not increase dramatically as the error threshold did, but precision increased. With an error threshold of 0, the classification success results were widely spread from 0% to 100%. However, with an error threshold of 1, the classification success results were closely packed between 45% and 90%, as showing in Fig. 5.

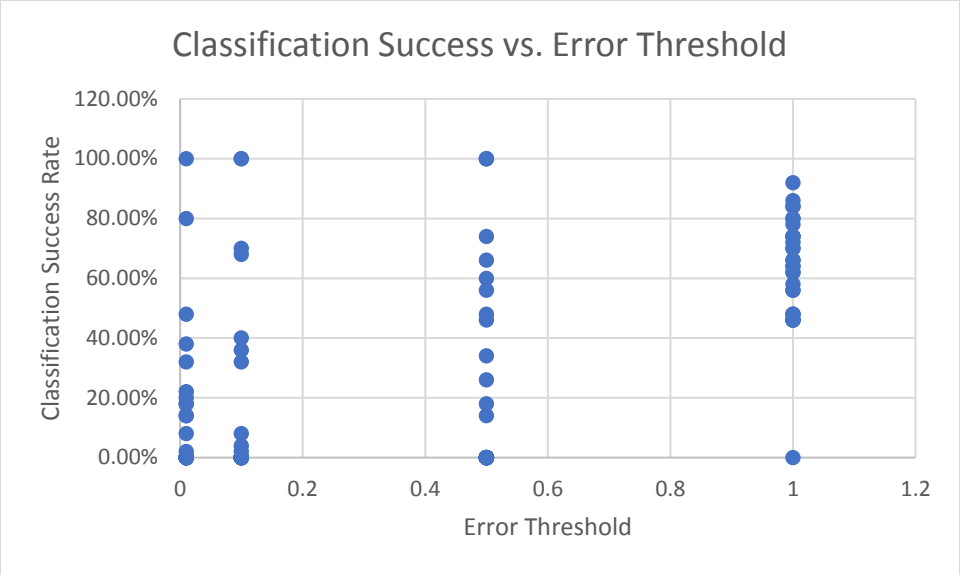


Figure 5: Hopfield ANN Classification Success vs. Error Handling

Perceptron Artificial Neural Network

General Testing

We found that the Perceptron ANN was decently accurate, with an average of 80-85% success while computing all tests.

Identity Testing

We confirmed the validity of our Perceptron ANN implementation, as the trained ANN had perfect recall of all its training samples.

Alpha & Theta Variance Testing

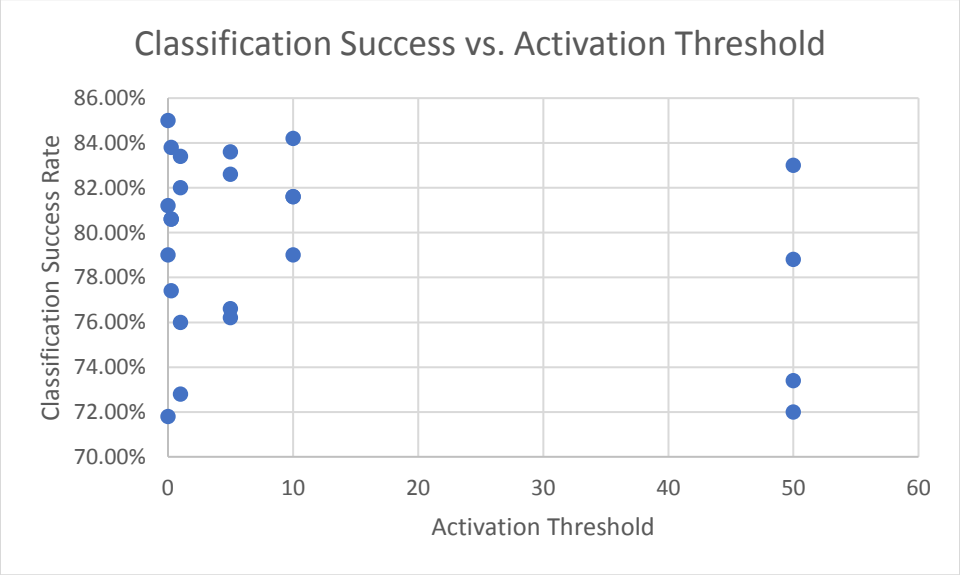


Figure 6: Perceptron ANN Classification Success vs. Activation Threshold, with weights initialized to 0

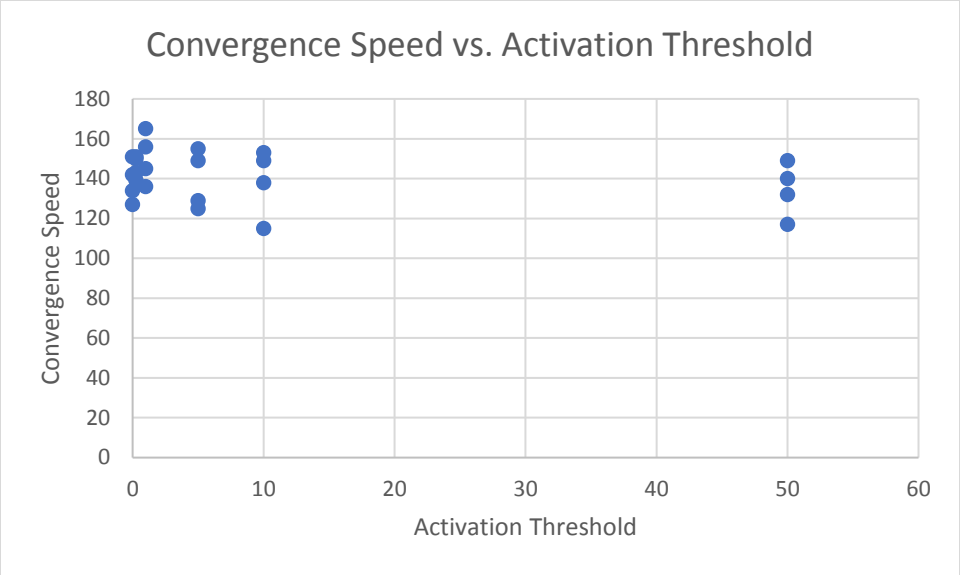


Figure 7: Perceptron ANN Convergence Speed vs. Activation Threshold, with weights initialized to 0

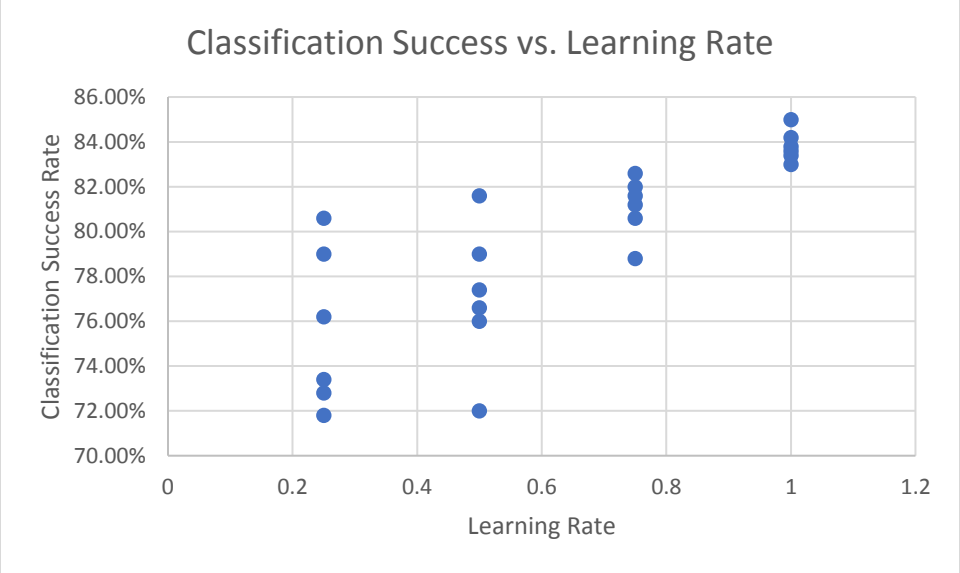


Figure 8: Perceptron ANN Classification Success vs. Learning Rate, with weights initialized to 0

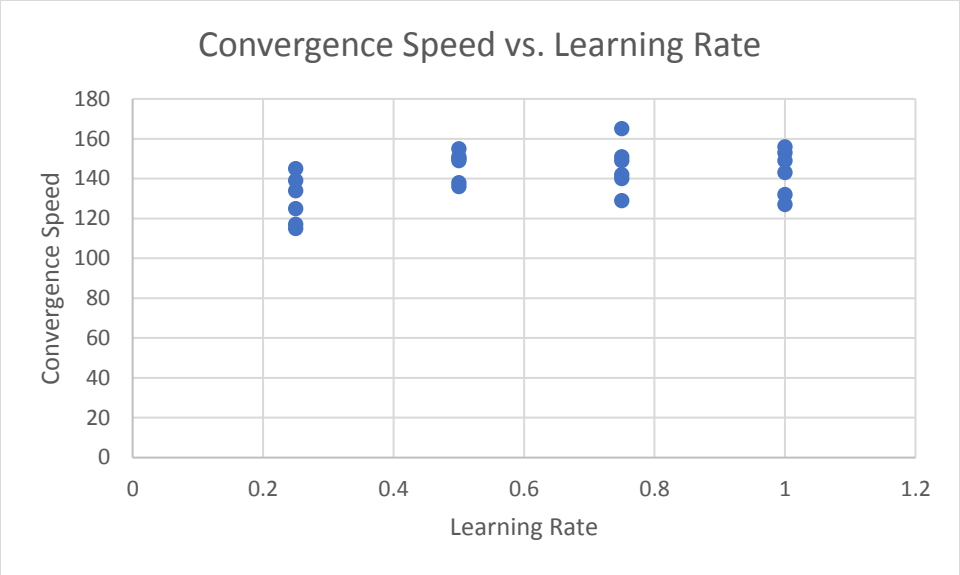


Figure 9: Perceptron ANN Convergence Speed vs. Learning Rate, with weights initialized to 0

As shown in Fig. 6 and Fig. 7, increasing the activation threshold of the ANN did not have a dramatic effect on either Classification Success or Convergence Speed. However, as shown in Fig. 8 and Fig. 9, we found that increasing the Learning Rate did have a significant effect on Classification Success and Convergence Speed. As we increased the Learning Rate, the Classification Success of the ANN increased, while the Convergence Speed of the ANN decreased.

We found a similar result when we initialized the weights of the ANN to a random value $\in [-0.5, 0.5]$. As shown in Fig. 10 and Fig. 11, increasing the activation Threshold of the ANN did not have a significant effect on either Classification Success or Convergence Speed. However, as shown in Fig. 12 and Fig. 13, we found that increasing the Learning Rate did have a significant effect on Classification Success, but not Convergence Speed. As the Learning Rate increased, the Classification Success dramatically increased, with a Learning Rate of 0.25 causing Classification Success to range from around 71% to 82%, while a Learning Rate of 1.0 caused Classification Success to range from around 81% to 85%. As Learning Rate increased, we saw no significant change in Convergence Speed.

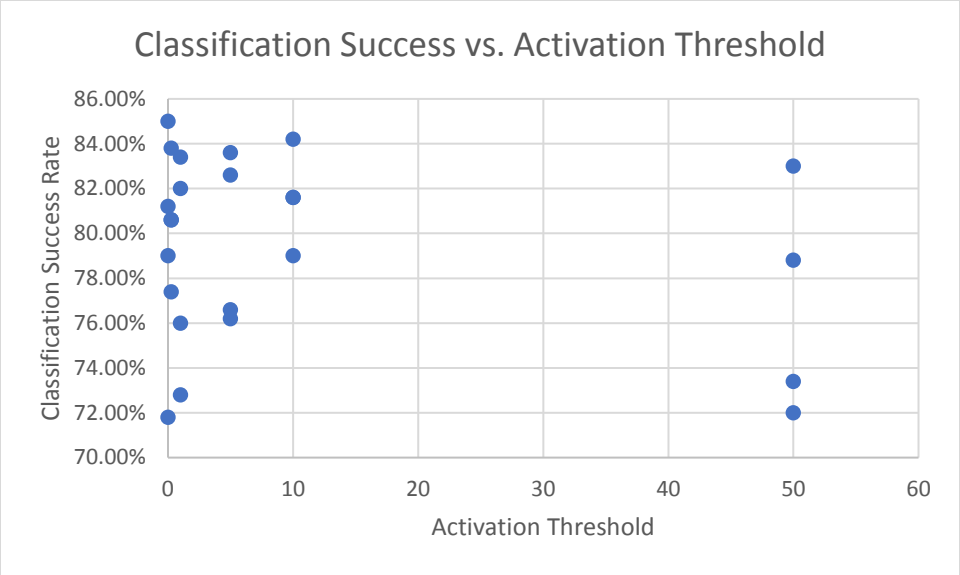


Figure 10: Perceptron ANN Classification Success vs. Activation Threshold, with weights initialized to a random value $\in [-0.5, 0.5]$

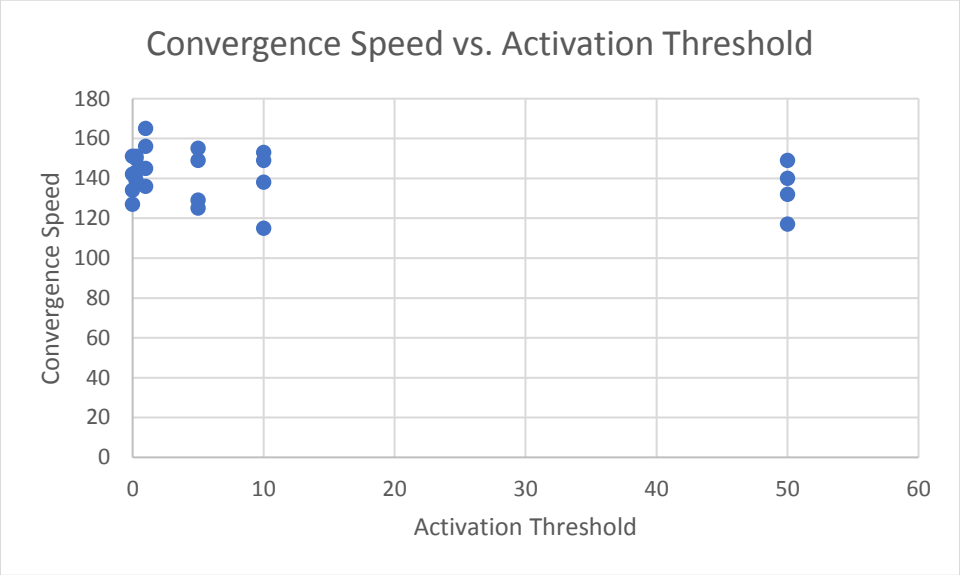


Figure 11: Perceptron ANN Convergence Speed vs. Activation Threshold, with weights initialized to a random value $\in [-0.5, 0.5]$

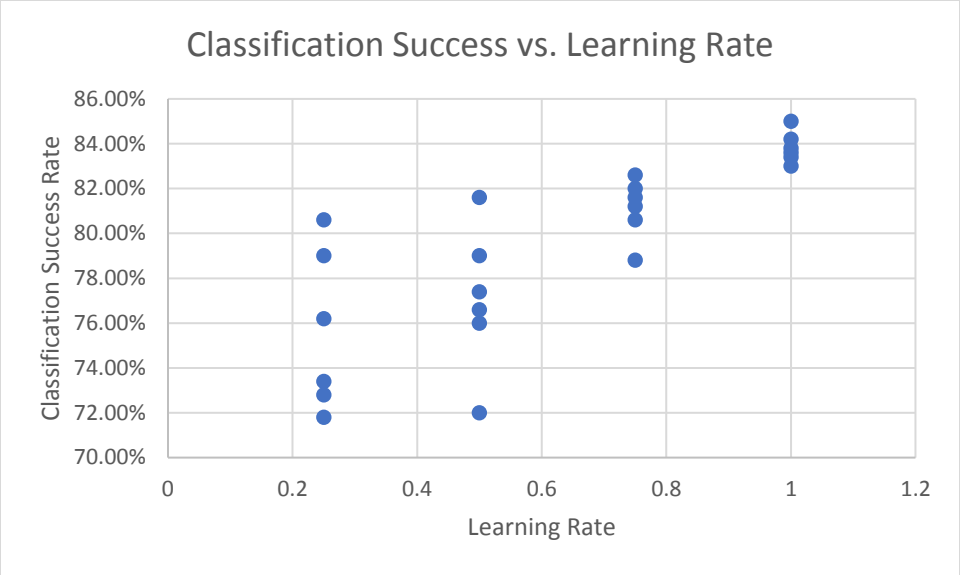


Figure 12: Perceptron ANN Classification Success vs. Learning Rate, with weights initialized to a random value $\in [-0.5, 0.5]$

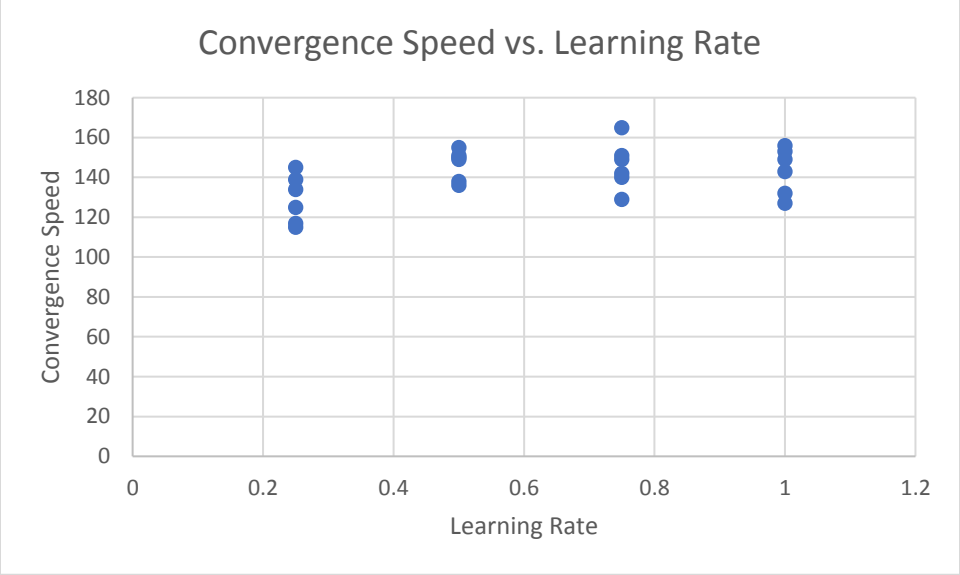


Figure 13: Perceptron ANN Convergence Speed vs. Learning Rate, with weights initialized to a random value $\in [-0.5, 0.5]$

General Conclusions

The Hopfield ANN performs worse than the Perceptron ANN in all our tests. After careful consideration, we have developed three reasons for why we believe this is the case.

Reason 1: Hopfield ANNs to not perform well with noise

We are working with images represented is pixel arrays. Due to these images being of handwritten digits, there is no way to have a *control* or *target* image, and thus every single image has some amount of noise. Hopfield ANNs work the best when there is a defined pattern to which each testing image will converge. However, even though we only have ten targets, the Hopfield ANN treats each new input pattern (60,000 of them) as separate patterns due to the noise.

Reason 2: Hopfield ANNs require more image training

To accurately represent all of the noise in our data, we would have needed to train with all 60,000 training samples. Given that each training image is a 28x28 pixel array, this is simply unfeasible. Each pixel is represented by 1 byte, and each image is 784 pixels (28x28), this means that we need 47.04MB just to represent all the training data. This exceeds the default allocated heap space by Java processes, but even if we increase the maximum allocated heap space, the time training and testing takes to complete is unfathomable long. If we had access to more computing power, perhaps a super computer with 128 or 256GB of RAM and a processor designed for number crunching or perhaps an ASIC machine, then we could have trained on all 60,000 training samples. However, given the restraints of our resources, we were required to sample the full training set into 500 images, which was not enough for the Hopfield ANN to handle with noise.

Reason 3: Architectural Differences between Perceptron ANNs and Hopfield ANNs

The biggest architectural difference is that Perceptron ANNs are largely designed for and work well with *Pattern Classification (PC)*, while Hopfield ANNs are largely designed and work well with *Pattern Association (PA)*. PC is when a computed classification result is compared with a specific expected output (target), while PA is used when comparing how similar or different testing samples are from each other. Give than the problem of this project is PC, because we are given ten distinct targets and are asking the network to classify inputs as one of said targets, the Perceptron ANN has the natural advantage, given that it is optimized for PC.

Appendix

Project Proposal

Handwritten Number Classification

Background Information

Modern communication is founded in handwriting; humans have used handwriting for thousands of years to illustrate their thoughts. Even though there is a general form for each character or number in an alphabet, each human has a seemingly unique form of writing each. Despite this vast variety of handwritings, most humans can determine the difference between an '8' and a '2' without much thought.

In order for computers and machines to continue their advancements in intelligence, they will need to be able to communicate with humans. One of these key forms of communication is written characters, and thus it is important for computers to be able to read human handwriting. This is a difficult and not-straight-forward task, as there are many different styles and forms of handwriting, yet it is accomplished by most children during elementary school.

We will develop an artificial neural network to recognize numbers written in different handwritings, just as a human brain does.

Project Description

Our project seeks to implement an application of image recognition utilizing deep learning. We desire to create a neural network that successfully classifies handwritten images- particularly handwritten digits zero through nine (0-9). Using the MNIST database of handwritten digits (see description in the *Data* section), our project will feed the neural network a series of images. These images - which are a grid of numbers representing the shade of each pixel - form an eighteen-by-eighteen matrix of input pixel values and are processed as an array of three-hundred and twenty-four numbers. Once processed, the neural network will produce ten different outputs, with each output corresponding to a likelihood that the image is a specific digit. By having a separate output for each type of object we want to recognize, we can use a neural network to classify objects into groups. Thus, our training of the neural network will involve reading in the pixelated images and classifying them as "0's" and "not 0's", "1's" and "not 1's", etc. so it learns to tell them apart. Ultimately, the neural network should be able to accept an image input vector, classify it as a digit ranging from zero to nine, and then return a probability of the image being classified within the ten image groups - e.g. the image of "8" is classified as 80% "8" and also 20% "5". When the project is complete, we will have a neural network that can recognize digits with a very high degree of accuracy.

Data

We will be using data from the MNIST data set provided by Yann LeCun, a professor at New York University, and Corinna Cortes, a research scientist at Google Labs, as well as Christopher J.C. Burges, a Microsoft Research Analyst. The MNIST data set contains 60,000 training samples and 10,000 testing samples of handwritten numbers. The numbers are in the range [0,9], and have been collected from 500

different writers. In addition, all of the numbers in the data set have been self-normalized and centered in a fixed-size image. The data set can be accessed at yann.lecun.com/exdb/mnist/ (see *References* section).

Deliverables

By the end of this project, we will deliver a program that can read in a set of correctly formatted images of hand-written numbers to train an artificial neural network. The trained artificial neural network will then be able to classify previously unknown images of hand-written numbers as the correct number, in the range of [0,9].

Deadlines

This project will adhere to the following deadlines:

- 12:00pm on Monday, April 24th, 2017: written project proposal
- 11:59pm on Sunday, May 7th, 2017: final source code
- 2:30pm on Monday, May 8th, 2017: project presentation slides
- 2:30pm on Tuesday, May 9th, 2017: project presentations
- 12:00pm on Wednesday, May 17th, 2017: final written project proposal

References

Geitgey, Adam. *"Machine Learning Is Fun! Part 3: Deep Learning and Convolutional Neural Networks."* Medium. A Medium Corporation, 13 June 2016. Web. 24 Apr. 2017. <https://medium.com/@ageitgey/machine-learning-is-fun-part-3-deep-learning-and-convolutional-neural-networks-f40359318721>.

LeCun, Yann, Corinna Cortes, and Christopher J.C. Burges. *"THE MNIST DATABASE"* MNIST Handwritten Digit Database, Yann LeCun, Corinna Cortes and Chris Burges. N.p., Nov. 1998. Web. 24 Apr. 2017. <http://yann.lecun.com/exdb/mnist/>.

Sample I/O Files

The input (*mnist_train.csv* and *mnist_test.csv*) and output (*results.txt*) files are too large to include in full in this writeup, so we have provided samples of each.

***mnist_train.csv* Sample**

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,51,159,253,159,50,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,48,238,252,25
2,252,237,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,54,227,253,252,239,233,252,57,6,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,10,60,224,252,253,252,202,84,252,253,122,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,163,252,252,252,
253,252,252,96,189,253,167,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,51,238,253,253,190,114,253,228,47,79,255,1
68,0,0,0,0,0,0,0,0,0,0,0,0,0,0,48,238,252,252,179,12,75,121,21,0,0,253,243,50,0,0,0,0,0,0,0,0,0,0,0,0,0
,38,165,253,233,208,84,0,0,0,0,0,0,253,252,165,0,0,0,0,0,0,0,0,0,0,0,0,7,178,252,240,71,19,28,0,0,0,0,0,
0,253,252,195,0,0,0,0,0,0,0,0,0,0,0,57,252,252,63,0,0,0,0,0,0,0,0,0,0,253,252,195,0,0,0,0,0,0,0,0,0,0,
198,253,190,0,0,0,0,0,0,0,0,0,0,255,253,196,0,0,0,0,0,0,0,0,0,0,0,76,246,252,112,0,0,0,0,0,0,0,0,0,0,253,
252,148,0,0,0,0,0,0,0,0,0,0,0,85,252,230,25,0,0,0,0,0,0,0,0,7,135,253,186,12,0,0,0,0,0,0,0,0,0,0,0,85,252
,223,0,0,0,0,0,0,0,0,7,131,252,225,71,0,0,0,0,0,0,0,0,0,0,0,0,85,252,145,0,0,0,0,0,0,48,165,252,173,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,86,253,225,0,0,0,0,0,0,114,238,253,162,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,85,252,249,1
46,48,29,85,178,225,253,223,167,56,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,85,252,252,252,229,215,252,252,252,1
96,130,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,28,199,252,252,253,252,252,233,145,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,25,128,252,253,252,141,37,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0

***mnist_test.csv* Sample**

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,11,150,253,202,31,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,37,251,251,253,1
07,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,21,197,251,251,253,107,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,110,190,251,251,251,253,169,109,62,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,253,251,251,251,251,253,2
51,251,220,51,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,182,255,253,253,253,253,234,222,253,253,253,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,63,221,253,251,251,251,147,77,62,128,251,251,105,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,32,2
31,251,253,251,220,137,10,0,0,31,230,251,243,113,5,0,0,0,0,0,0,0,0,0,0,0,0,37,251,251,253,188,20,0,0,
0,0,0,109,251,253,251,35,0,0,0,0,0,0,0,0,0,0,0,0,37,251,251,201,30,0,0,0,0,0,0,0,31,200,253,251,35,0,0,0,
0,0,0,0,0,0,0,0,0,37,253,253,0,0,0,0,0,0,0,0,32,202,255,253,164,0,0,0,0,0,0,0,0,0,0,0,0,0,140,251,251,0,0,0
,0,0,0,0,0,109,251,253,251,35,0,0,0,0,0,0,0,0,0,0,0,0,217,251,251,0,0,0,0,0,0,21,63,231,251,253,230,30,
0,0,0,0,0,0,0,0,0,0,0,217,251,251,0,0,0,0,0,0,144,251,251,251,221,61,0,0,0,0,0,0,0,0,0,0,0,0,0,217,251,
251,0,0,0,0,0,182,221,251,251,251,180,0,0,0,0,0,0,0,0,0,0,0,0,218,253,253,73,73,228,253,253,255,2
53,253,253,253,0,0,0,0,0,0,0,0,0,0,0,0,0,0,113,251,251,253,251,251,251,251,253,251,251,251,147,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,31,230,251,253,251,251,251,251,253,230,189,35,10,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,62,142,253,251,251,251,251,253,107,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,72,174,251,173,71,72,30
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0

,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0

Hopfield ANN *results.txt* Sample

```
Results from test sample #1
      Confidence Interval of 0: 94.01%
      Confidence Interval of 1: 81.38%
      Confidence Interval of 2: 86.1%
      Confidence Interval of 3: 85.84%
      Confidence Interval of 4: 82.53%
      Confidence Interval of 5: 87.24%
      Confidence Interval of 6: 88.52%
      Confidence Interval of 7: 86.73%
      Confidence Interval of 8: 83.16%
      Confidence Interval of 9: 84.44%
I am 94.01% sure this number is a 0

.
.
.

Results from test sample #74
      Confidence Interval of 0: 82.53%
      Confidence Interval of 1: 97.07%
      Confidence Interval of 2: 89.92%
      Confidence Interval of 3: 89.67%
      Confidence Interval of 4: 91.07%
      Confidence Interval of 5: 92.86%
      Confidence Interval of 6: 88.01%
      Confidence Interval of 7: 91.45%
      Confidence Interval of 8: 91.71%
      Confidence Interval of 9: 91.58%
I am 97.07% sure this number is a 1

.
.
.

Results from test sample #134
      Confidence Interval of 0: 82.91%
      Confidence Interval of 1: 91.96%
      Confidence Interval of 2: 92.86%
      Confidence Interval of 3: 90.56%
      Confidence Interval of 4: 86.35%
      Confidence Interval of 5: 90.82%
      Confidence Interval of 6: 85.33%
      Confidence Interval of 7: 85.84%
      Confidence Interval of 8: 91.33%
      Confidence Interval of 9: 86.73%
I am 92.86% sure this number is a 2

.
.
.
```

```
Results from test sample #310
      Confidence Interval of 0: 86.35%
      Confidence Interval of 1: 86.99%
      Confidence Interval of 2: 90.18%
      Confidence Interval of 3: 84.95%
      Confidence Interval of 4: 87.63%
      Confidence Interval of 5: 90.82%
      Confidence Interval of 6: 91.96%
      Confidence Interval of 7: 86.35%
      Confidence Interval of 8: 85.71%
      Confidence Interval of 9: 86.61%
I am 91.96% sure this number is a 6
```

Perceptron ANN *results.txt* Sample

```
Actual Output: 0
1 -1 -1 -1 -1 -1 -1 -1 -1 -1
Classified Output: 0
1 -1 -1 -1 -1 -1 -1 -1 -1 -1

Actual Output: 0
1 -1 -1 -1 -1 -1 -1 -1 -1 -1
Classified Output: 0
1 -1 -1 -1 -1 -1 -1 -1 -1 -1

Actual Output: 1
-1 1 -1 -1 -1 -1 -1 -1 -1 -1
Classified Output: 1
-1 1 -1 -1 -1 -1 -1 -1 -1 -1

Actual Output: 1
-1 1 -1 -1 -1 -1 -1 -1 -1 -1
Classified Output: 1
-1 1 -1 -1 -1 -1 -1 -1 -1 -1

Actual Output: 2
-1 -1 1 -1 -1 -1 -1 -1 -1 -1
Classified Output: 2
-1 -1 1 -1 -1 -1 -1 -1 -1 -1

Actual Output: 2
-1 -1 1 -1 -1 -1 -1 -1 -1 -1
Classified Output: 2
-1 -1 1 -1 -1 -1 -1 -1 -1 -1

Actual Output: 3
-1 -1 -1 1 -1 -1 -1 -1 -1 -1
Classified Output: 3
-1 -1 -1 1 -1 -1 -1 -1 -1 -1

Actual Output: 3
-1 -1 -1 1 -1 -1 -1 -1 -1 -1
Classified Output: 3
-1 -1 -1 1 -1 -1 -1 -1 -1 -1
```

Sample Runs

Hopfield Artificial Neural Network

Sample Run #1:

```
Welcome to my Hopfield Neural Network!


==========================================
|                 USER MENU                |
==========================================
|       Operation:            Command:     |
|                                          |
| 1. Train the neural net.       1         |
| 2. Test the neural net.        2         |
| 3. Exit the program.           3         |
==========================================
Please enter a command from the user menu above:
1
Please enter the image training file name (with extension):
mnist_train.csv
Please enter the file name where the weights will be saved:
weights.txt
Now Training...
Successfully trained.
Now saving weights...
Weights successfully saved.
```

Sample Run #2:

```
==========================================
|                 USER MENU                |
==========================================
|       Operation:            Command:     |
|                                          |
| 1. Train the neural net.       1         |
| 2. Test the neural net.        2         |
| 3. Exit the program.           3         |
==========================================
Please enter a command from the user menu above:
2
Please enter the image testing file name (with extension):
mnist_test.csv
Please enter the trained weights settings file name (with extension):
weights.txt
Please enter the file name where the testing results will be saved:
results.txt
Please enter the steepness parameter.
1
Please enter the error threshold.
0.1
Now Testing...
Testing Pattern 1 converged after 3 epochs.
Testing Pattern 2 converged after 3 epochs.
Testing Pattern 3 converged after 3 epochs.
```

```
Testing Pattern 4 converged after 2 epochs.
Testing Pattern 5 converged after 3 epochs.
Testing Pattern 6 converged after 3 epochs.
Testing Pattern 7 converged after 3 epochs.
.
.
.
.
.
Testing Pattern 499 converged after 3 epochs.
Testing Pattern 500 converged after 3 epochs.
Successfully tested. Results have been saved.
Computing statistics...
Target: 0
The Hopfield net classified 64.0% of the target patterns correctly.
------------------------
Target: 1
The Hopfield net classified 0.0% of the target patterns correctly.
------------------------
Target: 2
The Hopfield net classified 70.0% of the target patterns correctly.
------------------------
Target: 3
The Hopfield net classified 36.0% of the target patterns correctly.
------------------------
Target: 4
The Hopfield net classified 32.0% of the target patterns correctly.
------------------------
Target: 5
The Hopfield net classified 36.0% of the target patterns correctly.
------------------------
Target: 6
The Hopfield net classified 6.0% of the target patterns correctly.
------------------------
Target: 7
The Hopfield net classified 2.0% of the target patterns correctly.
------------------------
Target: 8
The Hopfield net classified 6.0% of the target patterns correctly.
------------------------
Target: 9
The Hopfield net classified 0.0% of the target patterns correctly.
------------------------
```

Perceptron Artificial Neural Network

Sample Run #1:

```
Enter 1 to train using a training data file, enter 2 to train using a trained weight
settings data file:
1
Enter the training data file name:
mnist_train.csv
Beginning training...
Enter 0 to initialize weights to 0, enter 1 to initialize weights to random values
between -0.5 and 0.5.
0
Enter the maximum number of training epochs:
300
Enter a file name to save the trained weight settings:
weights.txt
Enter the learning rate alpha from 0 to 1 but not including 0:
1
Enter the threshold theta:
0.25
Training converged after 131 epochs.
Saving weights to file...
Weights saved to file!
Enter the trained weight settings input data file name:
weights.txt
Enter 1 to test using a testing file, enter 2 to quit.
1
Enter the testing data file name.
mnist_test.csv
Enter a file name to save the testing results
results.txt
Please enter threshold theta:
0.25
Computing statistics...
Total Correct Classifications: 370.0
Total Classifications: 500.0
The perceptron classified 74.00% of the testing patterns correctly.
```

Sample Run #2:

```
Enter 1 to train using a training data file, enter 2 to train using a trained weight
settings data file:
1
Enter the training data file name:
mnist_train.csv
Beginning training...
Enter 0 to initialize weights to 0, enter 1 to initialize weights to random values
between -0.5 and 0.5.
0
Enter the maximum number of training epochs:
300
Enter a file name to save the trained weight settings:
weights.txt
Enter the learning rate alpha from 0 to 1 but not including 0:
```

```
1
Enter the threshold theta:
0
Training converged after 120 epochs.
Saving weights to file...
Weights saved to file!
Enter the trained weight settings input data file name:
weights.txt
Enter 1 to test using a testing file, enter 2 to quit.
1
Enter the testing data file name.
mnist_test.csv
Enter a file name to save the testing results
results.txt
Please enter threshold theta:
0
Computing statistics...
Total Correct Classifications: 400.0
Total Classifications: 500.0
The perceptron classified 80.00% of the testing patterns correctly.
```

Source Code

IDX-to-CSV Converter

```
def convert(imgf, labelf, outf, n):
    f = open(imgf, "rb")
    o = open(outf, "w")
    l = open(labelf, "rb")

    f.read(16)
    l.read(8)
    images = []

    for i in range(n):
        image = [ord(l.read(1))]
        for j in range(28*28):
            image.append(ord(f.read(1)))
        images.append(image)

    for image in images:
        o.write(",".join(str(pix) for pix in image)+"\n")
    f.close()
    o.close()
    l.close()

convert("train-images-idx3-ubyte", "train-labels-idx1-ubyte",
        "mnist_train.csv", 60000)
convert("t10k-images-idx3-ubyte", "t10k-labels-idx1-ubyte",
        "mnist_test.csv", 10000)
```

Hopfield Artificial Neural Network

```
/* Authors: Chaise Brown & Will Carhart
 * Due Date: 04/20/17
 * Summary: Implements a Hopfield Neural Network.
 */

import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.text.DecimalFormat;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Scanner;
import java.util.StringTokenizer;

public class fpHopfield {
	public static void main(String[] args) {
		// File I/O Variables
		Scanner input = new Scanner(System.in);
		BufferedReader br = null;
		String imageTrainingFile = null;
```

```
            String imageTestingFile = null;
            String savedWeightsFile = null;
            String resultsFile = null;
            double steepness, errorThreshold;

            System.out.println("Welcome to my Hopfield Neural Network!");
            while (true) {
                  displayUserMenu();
                  System.out
                              .println("Please enter a command from the user menu
above: ");
                  int command = input.nextInt();
                  String dummy = input.nextLine();
                  switch (command) {
                  case 1:
                        boolean validTrainingFile = false;
                        while (!validTrainingFile) {
                              System.out
                                          .println("Please enter the image
training file name (with extension): ");
                              imageTrainingFile = input.nextLine();
                              try {
                                    br = new BufferedReader(new FileReader(
                                                imageTrainingFile));
                                    validTrainingFile = true;
                              } catch (FileNotFoundException e) {
                                    System.err.print(imageTrainingFile + " not
found.\n");
                              }
                        }
                        createKeyFile(imageTrainingFile);
                        System.out
                                    .println("Please enter the file name where
the weights will be saved: ");
                        savedWeightsFile = input.nextLine();
                        train(imageTrainingFile, savedWeightsFile);
                        break;
                  case 2:
                        boolean notTrained = false;
                        try {
                              br = new BufferedReader(new FileReader("key.txt"));
                              if (br.readLine() == null) {
                                    System.err
                                                .println("The neural network has
not been trained yet. Please train the network before testing");
                                    notTrained = true;
                              }
                        } catch (IOException e) {
                              System.err
                                          .println("The neural network has not
been trained yet. Please train the network before testing");
                              notTrained = true;
                        }
                        if (notTrained) {
                              break;
```

```
                        }

                        boolean validTestingFile = false;
                        while (!validTestingFile) {
                              System.out
                                          .println("Please enter the image
testing file name (with extension): ");
                              imageTestingFile = input.nextLine();
                              try {
                                    br = new BufferedReader(
                                                new
FileReader(imageTestingFile));
                                    validTestingFile = true;
                              } catch (FileNotFoundException e) {
                                    System.err.print(imageTestingFile + " not
found.\n");
                              }
                        }
                        boolean validWeightFile = false;
                        while (!validWeightFile) {
                              System.out
                                          .println("Please enter the trained
weights settings file name (with extension): ");
                              savedWeightsFile = input.nextLine();
                              try {
                                    br = new BufferedReader(
                                                new
FileReader(savedWeightsFile));
                                    validWeightFile = true;
                              } catch (FileNotFoundException e) {
                                    System.err.print(savedWeightsFile + " not
found.\n");
                              }
                        }
                        System.out
                                    .println("Please enter the file name where
the testing results will be saved: ");
                        resultsFile = input.nextLine();
                        System.out.println("Please enter the steepness
parameter.");
                        steepness = input.nextDouble();
                        System.out.println("Please enter the error threshold.");
                        errorThreshold = input.nextDouble();
                        test(imageTestingFile, savedWeightsFile, resultsFile,
                                    steepness, errorThreshold);
                        statistics(resultsFile);
                        break;
                  case 3:
                        try {
                              br.close();
                              input.close();
                        } catch (IOException e) {
                              System.err.print("Error in I/O");
                              System.exit(1);
                        }
```

```
                    System.exit(1);
                    break;
              }
        }
}

/*
 * Creates a key file (reference to the training patterns)
 *
 * @param imageTrainingFile the name of the image training file
 */
public static void createKeyFile(String imageTrainingFile) {
        // I/O variables
        PrintWriter pw = null;
        BufferedReader br = null;
        String temp;

        try {
              br = new BufferedReader(new FileReader(imageTrainingFile));
              pw = new PrintWriter("key.txt", "UTF-8");

              // transfer from training file to key file
              while ((temp = br.readLine()) != null) {
                    pw.write(temp + "\n");
              }

              br.close();

        } catch (FileNotFoundException e) {
              System.err.print("File not found");
              System.exit(1);
        } catch (IOException e) {
              e.printStackTrace();
        }
        pw.close();
}

/*
 * Implements the training of the Hopfield NN
 *
 * @param imageTrainingFile the name of the image training file
 *
 * @param savedWeightsFile the name of the saved weights file
 */
public static String[] train(String imageTrainingFile,
              String savedWeightsFile) {

        // File I/O Variables
        BufferedReader br = null;
        String toBeTokenized = null;
        String[] lineElements = null;
        int image_dim = 0;

        // training variables
        ArrayList<Double> trainingData = null;
```

```
            String[] fileNames = { "matrix0.txt", "matrix1.txt", "matrix2.txt",
                        "matrix3.txt", "matrix4.txt", "matrix5.txt",
"matrix6.txt",
                        "matrix7.txt", "matrix8.txt", "matrix9.txt" };

            System.out.println("Now Training...");

            // read in information about training data
            try {

                  br = new BufferedReader(new FileReader(imageTrainingFile));
                  toBeTokenized = br.readLine();
                  lineElements = toBeTokenized.split(",");
                  StringBuilder sb = new StringBuilder(lineElements[0]);
                  sb.deleteCharAt(0);
                  sb.deleteCharAt(0);
                  sb.deleteCharAt(0);
                  String newStr = sb.toString();
                  image_dim = Integer.parseInt(newStr);
                  br.readLine();
                  br.readLine();

                  double[][] weightMatrix = new double[image_dim][image_dim];
                  double[][] averageTargetMatrix = new double[10][image_dim];
                  int[] targets = new int[10];
                  int target;

                  // Training
                  while ((toBeTokenized = br.readLine()) != null) {
                        lineElements = toBeTokenized.split(",");
                        target = Integer.parseInt(lineElements[0]);
                        switch (target) {
                        case 0:
                              targets[0]++;
                              break;
                        case 1:
                              targets[1]++;
                              break;
                        case 2:
                              targets[2]++;
                              break;
                        case 3:
                              targets[3]++;
                              break;
                        case 4:
                              targets[4]++;
                              break;
                        case 5:
                              targets[5]++;
                              break;
                        case 6:
                              targets[6]++;
                              break;
                        case 7:
```

```
                                    targets[7]++;
                                    break;
                              case 8:
                                    targets[8]++;
                                    break;
                              case 9:
                                    targets[9]++;
                                    break;
                              }
                              trainingData = new ArrayList<Double>();
                              for (int i = 1; i < lineElements.length; i++) {
                                    // normalize to bipolar vector
                                    double proportion =
(Double.parseDouble(lineElements[i]) / 255);
                                    double toAdd = ((proportion * 2) - 1);
                                    trainingData.add(toAdd);
                                    averageTargetMatrix[target][i - 1] += toAdd;
                              }
                              // Creating weight matrix
                              for (int i = 0; i < image_dim; i++) {
                                    for (int j = 0; j < image_dim; j++) {
                                          weightMatrix[i][j] += (trainingData.get(i) *
trainingData.get(j));
                                    }
                              }
                        }

                        // Updating weights on main diagonal once training completes
                        int i = 0;
                        int j = 0;
                        while (i < image_dim && j < image_dim) {
                              weightMatrix[i][j] = 0;
                              i++;
                              j++;
                        }

                        for (int x = 0; x < image_dim; x++) {
                              for (int y = 0; y < image_dim; y++) {
                                    weightMatrix[x][y] /= 500;
                              }
                        }

                        for (int k = 0; k < averageTargetMatrix.length; k++) {
                              for (int l = 0; l < averageTargetMatrix[0].length; l++) {
                                    averageTargetMatrix[k][l] /= targets[k];
                              }
                        }

                        for (int k = 0; k < fileNames.length; k++) {
                              PrintWriter pw = new PrintWriter(new
FileWriter(fileNames[k]));
                              for (int l = 0; l < averageTargetMatrix[0].length; l++) {
                                    pw.print(averageTargetMatrix[k][l] + " ");
                              }
                              pw.close();
```

```
			}

			System.out.println("Successfully trained.");
			System.out.println("Now saving weights...");

			saveWeights(weightMatrix, savedWeightsFile);

			System.out.println("Weights successfully saved.");

			br.close();
		} catch (IOException e) {
			System.err.print("Error in I/O");
			System.exit(1);
		}

		return fileNames;
	}

	/*
	 * Implements the testing of the trained Hopfield NN
	 *
	 * @param imageTestingFile the name of the image testing file
	 *
	 * @param savedWeightsFile the name of the saved weights file
	 *
	 * @param resultsFile the name of the results file
	 */
	public static void test(String imageTestingFile, String savedWeightsFile,
			String resultsFile, double steepness, double errorThreshold) {
		// File I/O Variables
		BufferedReader br = null;
		String toBeTokenized = null;
		String[] lineElements = null;
		int image_dim = 0;

		System.out.println("Now Testing...");

		// testing variables
		ArrayList<Double> testingData = null;
		ArrayList<Integer> target = new ArrayList<Integer>();
		ArrayList<Double> y = null;
		double[][] weightMatrix = null;
		double yin, yval;
		int index = 0;
		boolean converged;
		int count = 0;

		// read in information about testing data
		try {

			br = new BufferedReader(new FileReader(imageTestingFile));
			toBeTokenized = br.readLine();
			lineElements = toBeTokenized.split(",");
			StringBuilder sb = new StringBuilder(lineElements[0]);
			sb.deleteCharAt(0);
```

```
                        sb.deleteCharAt(0);
                        sb.deleteCharAt(0);
                        String newStr = sb.toString();
                        image_dim = Integer.parseInt(newStr);
                        br.readLine();
                        br.readLine();

                        weightMatrix = weightMatrix(savedWeightsFile, image_dim);

                        // Later used for randomized pattern generator
                        ArrayList<Integer> list = new ArrayList<Integer>();
                        for (int i = 0; i < image_dim; i++) {
                              list.add(new Integer(i));
                        }

                        // begin testing
                        while ((toBeTokenized = br.readLine()) != null) {
                              // Read in testing data
                              testingData = new ArrayList<Double>();
                              lineElements = toBeTokenized.split(",");
                              target.add(Integer.parseInt(lineElements[0])); // reads
target

                        // value
                              for (int i = 1; i < lineElements.length; i++) {
                                    // normalize to bipolar vector
                                    double proportion =
(Double.parseDouble(lineElements[i]) / 255);
                                    double toAdd = ((proportion * 2) - 1);
                                    testingData.add(toAdd);
                              }

                              y = new ArrayList<Double>();

                              // Create copy of testing data
                              for (int i = 0; i < image_dim; i++) {
                                    y.add(testingData.get(i));
                              }

                              converged = false;
                              int epochs = 0;

                              while (!converged) {
                                    converged = true;
                                    // Create random neuron testing order
                                    Collections.shuffle(list);
                                    // Calculating yin
                                    // k increments the value in the randomized neuron
list
                                    // m increments the value in the matrix to calculate
                                    for (int i = 0; i < image_dim; i++) {
                                          double sum = 0;
                                          for (int m = 0; m < image_dim; m++) {
                                                sum += (y.get(m) *
weightMatrix[m][list.get(i)]);
```

```
                                        }
                                        sum /= image_dim;
                                        yin = testingData.get(list.get(i)) + sum;
                                        yval = activationFunction(yin, steepness);
                                        if (Math.abs(y.get(list.get(i)) - yval) >
errorThreshold) {
                                              y.set(list.get(i), yval);
                                              converged = false;
                                        }
                                  }
                                  epochs++;
                            }
                            // save results to file
                            count++;
                            saveResults(resultsFile, y, count, errorThreshold);
                            System.out.println("Testing Pattern " + (index + 1)
                                        + " converged after " + epochs + " epochs.");
                            index++;
                      }
                } catch (IOException e) {
                      System.err.print("Error in I/O");
                      System.exit(1);
                }

                System.out.println("Successfully tested. Results have been saved.");
          }

          /*
           * Saves the weights of the trained Hopfield NN to a given weights file
           *
           * @param weightMatrix the 2D array that represents weighted matrix
           *
           * @param savedWeightsFile the name of the saved weights file
           */
          public static void saveWeights(double[][] weightMatrix,
                      String savedWeightsFile) {
                PrintWriter pw = null;

                try {
                      pw = new PrintWriter(savedWeightsFile);
                      for (int i = 0; i < weightMatrix.length; i++) {
                            for (int j = 0; j < weightMatrix[i].length; j++) {
                                  pw.print(weightMatrix[i][j] + " ");
                            }
                            pw.println();
                      }
                } catch (IOException e) {
                      System.err.println("Error in I/O while saving weight matrix");
                      System.exit(1);
                }

                pw.close();
          }

          /*
```

```
        * Copies the weights from a saved file to a 2D array, representing the
        * weight matrix
        *
        * @param savedWeightsFile the name of the saved weights file
        *
        * @param image_dim the dimension of the input image
        *
        * @return int[][] a 2D array that represents the weight matrix
        */
       public static double[][] weightMatrix(String savedWeightsFile, int image_dim)
{
              // File I/O variables
              BufferedReader br = null;
              String toBeTokenized = null;
              StringTokenizer tokenizer = null;
              double[][] weightMatrix = new double[image_dim][image_dim];

              // read from saved weights file
              try {
                     br = new BufferedReader(new FileReader(savedWeightsFile));
                     toBeTokenized = br.readLine();
                     tokenizer = new StringTokenizer(toBeTokenized);
                     for (int i = 0; i < weightMatrix.length; i++) {
                            for (int j = 0; j < weightMatrix[i].length; j++) {
                                   if (tokenizer.hasMoreTokens()) {
                                          weightMatrix[i][j] =
Double.parseDouble(tokenizer
                                                        .nextToken());
                                   } else {
                                          toBeTokenized = br.readLine();
                                          if (toBeTokenized == null)
                                                 break;
                                          else {
                                                 tokenizer = new
StringTokenizer(toBeTokenized);
                                                 weightMatrix[i][j] =
Double.parseDouble(tokenizer
                                                              .nextToken());
                                          }
                                   }
                            }
                     }
                     br.close();
              } catch (IOException e) {
                     System.err.print("Error in I/O");
                     System.exit(1);
              }
              return weightMatrix;
       }

       /*
        * Computes the activation of a given neuron
        *
        * @param yin the computed dot product for the given neuron
        *
```

```
 * @return int the activation value for the neuron
 */
public static double activationFunction(double yin, double steepness) {
	double denominator = 1 + Math.exp(-1 * steepness * yin);

	return (2 * (1 / denominator) - 1);
}

/*
 * Displays the user menu
 */
public static void displayUserMenu() {
	System.out.println("\n");
	System.out.println("==========================================");
	System.out.println("|                 USER MENU                |");
	System.out.println("==========================================");
	System.out.println("|       Operation:           Command:     |");
	System.out.println("|                                         |");
	System.out.println("| 1. Train the neural net.       1        |");
	System.out.println("| 2. Test the neural net.        2        |");
	System.out.println("| 3. Exit the program.           3        |");
	System.out.println("==========================================");
}

/*
 * Saves the results of the testing operation to an output file
 *
 * @param resultsFile the name of the results file
 *
 * @param y the converged results matrix
 *
 * @param count the index of the testing set
 */
public static void saveResults(String resultsFile, ArrayList<Double> y,
		int count, double errorThreshold) {
	PrintWriter pw = null;
	BufferedReader br;
	String input;
	String[] lineElements = null;
	double[] results = new double[10];

	// clear file if first time opening it
	if (count == 1) {
		FileWriter fwOb;
		PrintWriter pwOb;
		try {
			fwOb = new FileWriter(resultsFile, false);
			pwOb = new PrintWriter(fwOb, false);
			pwOb.flush();
			pwOb.close();
			fwOb.close();
		} catch (FileNotFoundException e) {
			System.err.println(resultsFile + " not found");
			System.exit(1);
		} catch (IOException e) {
```

```
                        System.err.println("Error in I/O");
                        System.exit(1);
                  }
            }

            double testValue;
            for (int i = 0; i < 10; i++) {
                  try {
                        br = new BufferedReader(new FileReader("matrix" + i +
".txt"));
                        input = br.readLine();
                        lineElements = input.split(" ");
                        for (int j = 0; j < y.size(); j++) {
                              testValue = Double.parseDouble(lineElements[j]);
                              if (Math.abs(y.get(j) - testValue) < errorThreshold)
{
                                    results[i]++;
                              }
                        }
                        results[i] /= y.size();
                  } catch (IOException e) {
                        System.err.println("Error in I/O");
                        System.exit(1);
                  }
            }

            // write to output file
            int maxIndex = 0;
            double max = 0;
            DecimalFormat df;
            try {
                  pw = new PrintWriter(new FileWriter(resultsFile, true));
                  pw.write("Results from test sample #" + count + "\n");
                  df = new DecimalFormat();
                  df.setMaximumFractionDigits(2);
                  for (int i = 0; i < results.length; i++) {
                        pw.write("\tConfidence Interval of " + i + ": "
                                    + df.format(results[i] * 100) + "%\n");
                        if (results[i] > max) {
                              max = results[i];
                              maxIndex = i;
                        }
                  }
                  pw.write("I am " + df.format(max * 100)
                              + "% sure this number is a " + maxIndex);
                  pw.write("\n\n");
                  pw.close();
            } catch (IOException e) {
                  System.err.println("Error in I/O");
                  System.exit(1);
            }
      }

      public static void statistics(String resultsFile) {
            BufferedReader br = null;
```

```
		String currentLine;
		String [] lineElements;

		int [] targets = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
		int count = 0;
		int target_index = 0;
		double correctlyClassified = 0;
		double totalClassified = 0;
		double [] percentageClassified = new double [targets.length];

		System.out.println("Computing statistics...");

		try{
			br = new BufferedReader(new FileReader(resultsFile));
			while((currentLine = br.readLine()) != null) {
				br.readLine();
				br.readLine();
				br.readLine();
				br.readLine();
				br.readLine();
				br.readLine();
				br.readLine();
				br.readLine();
				br.readLine();
				br.readLine();
				currentLine = br.readLine();
				lineElements = currentLine.split(" ");
				int classifiedNum = Integer.parseInt(lineElements[8]);
				if (classifiedNum == targets[target_index]) {
					correctlyClassified++;
				}
				totalClassified++;
				br.readLine();
				if(totalClassified == 50){
					percentageClassified[target_index] =
(correctlyClassified / totalClassified);
					totalClassified = 0;
					correctlyClassified = 0;
					target_index++;
				}
			}

			for(int i = 0; i < percentageClassified.length; i++){
				System.out.println("Target: " + i);
				System.out.println("The Hopfield net classified "
						+ (percentageClassified[i] * 100)
						+ "% of the target patterns correctly.");
				System.out.println("------------------------");
			}

		} catch (IOException e) {
			System.err.println("Error in I/O");
			System.exit(1);
		}
	}
```

```
}
```

Perceptron Artificial Neural Network

```
/*
 * COMP 380: Neural Networks
 * Project 1:Perceptron Neural Network
 * Authors:         Will Carhart, Chaise Brown
 * Date:            February 28th, 2017
 * Summary:         This program implements a perceptron learning algorithm to
 *                        classify specific letters in multiple fonts.
 */

import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.Random;
import java.util.Scanner;

public class fpPerceptron {
	public static void main(String[] args) {
		// file IO variables
		Scanner kb = new Scanner(System.in);
		String trainingfile = null, trainedweights = null;
		String dummy;
		BufferedReader br = null;
		PrintWriter pw = null;

		while (true) {
			System.out
					.println("Welcome to my first neural network - A
Perceptron Net!");
			System.out
					.println("Enter 1 to train using a training data
file, enter 2 to train using a trained weight settings data file:");
			int trainingMethod = kb.nextInt();
			dummy = kb.nextLine();
			switch (trainingMethod) {
			case 1:
				boolean validFile = false;
				while (!validFile) {
					System.out.println("Enter the training data file
name: ");
					trainingfile = kb.nextLine();
					try {
						br = new BufferedReader(new
FileReader(trainingfile));
						validFile = true;
						br.close();
					} catch (IOException e) {
```

```
                                        System.err.print(trainingfile + " not
found\n");
                                }
                        }
                        train(trainingfile);
                        System.out
                                        .println("Enter the trained weight settings
input data file name: ");
                        trainedweights = kb.nextLine();
                        test(trainedweights);
                        break;
                case 2:
                        System.out
                                        .println("Enter the trained weight settings
input data file name: ");
                        trainedweights = kb.nextLine();
                        test(trainedweights);
                        break;
                }
        }
    }

    /*
     * Trains the neural network using a specific input file
     *
     * @param trainingfile the name of the input training file
     */
    public static void train(String trainingfile) {
        System.out.println("Beginning training...");

        // IO Variables
        BufferedReader br = null;
        String currentLine, trainedweights;
        String[] lineElements = null;
        int input_dim = 0;
        int output_dim = 0;
        Scanner kb = new Scanner(System.in);

        // Training Variables
        boolean converged = false;
        double alpha;
        double theta;
        int MAX_EPOCHS;

        try {
            br = new BufferedReader(new FileReader(trainingfile));
            currentLine = br.readLine();
            lineElements = currentLine.split(",");
            StringBuilder sb = new StringBuilder(lineElements[0]);
            sb.deleteCharAt(0);
            sb.deleteCharAt(0);
            sb.deleteCharAt(0);
            String newStr = sb.toString();
            input_dim = Integer.parseInt(newStr);
            currentLine = br.readLine();
```

```
                lineElements = currentLine.split(",");
                output_dim = Integer.parseInt(lineElements[0]);
                currentLine = br.readLine();
                lineElements = currentLine.split(",");
                br.readLine();
            } catch (IOException e) {
                System.err.print("Error in I/O");
                System.exit(1);
            }

            /*
             * first element of each first dimension array is the bias first
             * dimension is the row + column data second dimension is the pattern we
             * are trying to recognize (output_dim)
             */
            System.out
                        .println("Enter 0 to initialize weights to 0, enter 1 to
intialize weights to random values between -0.5 and 0.5.");
            int input = kb.nextInt();
            Random rand = new Random();
            double n = 1.5;
            double num;
            double my_num;
            double[][] weights = new double[output_dim][input_dim + 1];
            if (input == 0) {
                for (int a = 0; a < output_dim; a++) {
                    for (int b = 0; b < input_dim + 1; b++) {
                        weights[a][b] = 0;
                    }
                }
            } else {
                for (int a = 0; a < output_dim; a++) {
                    for (int b = 0; b < input_dim + 1; b++) {
                        num = rand.nextDouble() % n;
                        my_num = -0.5 + num;
                        weights[a][b] = my_num;
                    }
                }
            }

            String dummy = kb.nextLine();
            System.out.println("Enter the maximum number of training epochs: ");
            MAX_EPOCHS = kb.nextInt();
            int epochs = 0; // the number of epochs that have occurred

            dummy = kb.nextLine();
            System.out
                        .println("Enter a file name to save the trained weight
settings: ");
            trainedweights = kb.nextLine();

            System.out
                        .println("Enter the learning rate alpha from 0 to 1 but
not including 0: ");
            alpha = kb.nextDouble();
```

```
		boolean run = true;
		while (run) {
			if (alpha == 0) {
				System.out
						.println("Alpha should not be zero. Please
try again.");
				System.out
						.println("Enter the learning rate alpha from
0 to 1 but not including 0: ");
				alpha = kb.nextDouble();
			} else
				run = false;
		}

		System.out.println("Enter the threshold theta: ");
		theta = kb.nextDouble();

		ArrayList<Integer> trainingData = null; // training data collected from
                                                                        //
training file
		int target; // target
		int yin; // inputs to the af
		int y; // outputs of the af
		double[][] w_old = new double[output_dim][input_dim + 1];

		boolean changed = false;

		while (!converged && epochs < MAX_EPOCHS) {

			// reading in training set
			try {
				while ((currentLine = br.readLine()) != null) {
					trainingData = new ArrayList<Integer>();
					int[] targets = new int[output_dim];
					lineElements = currentLine.split(",");
					target = Integer.parseInt(lineElements[0]);
					for (int i = 0; i < targets.length; i++) {
						if (i == target) {
							targets[i] = 1;
						} else {
							targets[i] = -1;
						}
					}
					for (int i = 1; i < lineElements.length; i++) {
						if (Integer.parseInt(lineElements[i]) > 0)
							trainingData.add(1);
						else
							trainingData.add(-1);
					}

					// Training
					yin = 0;
					for (int i = 0; i < output_dim; i++) {
						for (int j = 0; j < input_dim; j++) {
```

```
                                        yin += trainingData.get(j) *
weights[i][j + 1];
                                  }
                                  yin += weights[i][0];
                                  y = activationFunction(yin, theta);
                                  if (y != targets[i]) {
                                        changed = true;
                                        for (int j = 0; j < (input_dim + 1);
j++) {
                                              w_old[i][j] = weights[i][j];
                                        }
                                        for (int j = 1; j < (input_dim + 1);
j++) {
                                              weights[i][j] = w_old[i][j]
                                                          + (alpha *
targets[i] * trainingData
                                                                        .get(j
- 1));
                                        }
                                        weights[i][0] = w_old[i][0] + (alpha *
targets[i]);
                                  }
                            }
                      }

                      // record
                      epochs++;
                      if (!changed) {
                            converged = true;
                      }
                      changed = false;
                      try {
                            br = new BufferedReader(new
FileReader(trainingfile));
                            br.readLine();
                            br.readLine();
                            br.readLine();
                      } catch (FileNotFoundException e) {
                            e.printStackTrace();
                      } catch (IOException e) {
                            e.printStackTrace();
                      }
                } catch (IOException e) {
                      System.err.print("Error in I/O");
                      System.exit(1);
                }
          }

          System.out.println("Training converged after " + epochs + " epochs.");

          System.out.println("Saving weights to file...");

          saveWeights(input_dim, output_dim, weights, trainedweights);

          System.out.println("Weights saved to file!");
```

```
	}

	/*
	 * Implements the deployment of the trained neural network
	 *
	 * @param trainedweights the name of the file containing the trained weights
	 */
	public static void test(String trainedweights) {

		BufferedReader br;
		Scanner kb = new Scanner(System.in);
		String[] lineElements;
		String testingfile = null, testingresults = null, dummy, currentLine;
		int input_dim = 0, output_dim = 0;
		int num = 0;
		double theta;

		System.out
				.println("Enter 1 to test using a testing file, enter 2 to
quit.");
		int answer = kb.nextInt();
		switch (answer) {
		case 1:
			dummy = kb.nextLine();
			System.out.println("Enter the testing data file name.");
			testingfile = kb.nextLine();
			System.out.println("Enter a file name to save the testing
results");
			testingresults = kb.nextLine();
			break;
		case 2:
			System.exit(0);
			break;
		}

		// read in weights from file
		double weights[][] = null;
		try {
			br = new BufferedReader(new FileReader(trainedweights));
			currentLine = br.readLine();
			input_dim = Integer.parseInt(currentLine);
			currentLine = br.readLine();
			output_dim = Integer.parseInt(currentLine);
			br.readLine();

			weights = new double[output_dim][input_dim + 1];
			for (int i = 0; i < output_dim; i++) {
				for (int j = 0; j < input_dim + 1; j++) {
					weights[i][j] = Double.parseDouble(br.readLine());
				}
			}
		} catch (FileNotFoundException e) {
			System.err.print("File not found");
			System.exit(1);
		} catch (IOException e) {
```

```
                System.err.print("Error in I/O");
                System.exit(1);
        }

        System.out.println("Please enter threshold theta: ");
        theta = kb.nextDouble();

        // read in test cases
        int yin, y;
        ArrayList<Integer> testingData = null;
        int target;

        try {
                br = new BufferedReader(new FileReader(testingfile));
                currentLine = br.readLine();
                lineElements = currentLine.split(",");
                StringBuilder sb = new StringBuilder(lineElements[0]);
                sb.deleteCharAt(0);
                sb.deleteCharAt(0);
                sb.deleteCharAt(0);
                String newStr = sb.toString();
                input_dim = Integer.parseInt(newStr);
                currentLine = br.readLine();
                lineElements = currentLine.split(",");
                output_dim = Integer.parseInt(lineElements[0]);
                currentLine = br.readLine();
                lineElements = currentLine.split(",");
                br.readLine();

                while ((currentLine = br.readLine()) != null) {
                        testingData = new ArrayList<Integer>();
                        int[] targets = new int[output_dim];
                        lineElements = currentLine.split(",");
                        target = Integer.parseInt(lineElements[0]);
                        for (int i = 0; i < targets.length; i++) {
                                if (i == target) {
                                        targets[i] = 1;
                                } else {
                                        targets[i] = -1;
                                }
                        }
                        for (int i = 0; i < lineElements.length; i++) {
                                if (Integer.parseInt(lineElements[i]) > 0)
                                        testingData.add(1);
                                else
                                        testingData.add(-1);
                        }

                        int[] results = new int[output_dim];
                        yin = 0;

                        for (int i = 0; i < output_dim; i++) {
                                for (int j = 0; j < input_dim; j++) {
                                        yin += testingData.get(j) * weights[i][j +
1];
```

```
                        }
                        yin += weights[i][0];
                        y = activationFunction(yin, theta);
                        results[i] = y;
                  }
                  output(results, targets, output_dim, testingresults, num);
                  num++;
            }
      } catch (FileNotFoundException e) {
            System.err.print("File not found");
            System.exit(1);
      } catch (IOException e) {
            System.err.print("Error in I/O");
            System.exit(1);
      }

      System.out.println("Computing statistics...");
      statistics(testingresults);
}


/*
 * Outputs the classification of the perceptron to a file
 *
 * @param results the actual results of the perceptron
 *
 * @param target the expected results of the perceptron
 *
 * @param output_dim the number of output dimensions
 */
public static void output(int[] results, int[] targets, int output_dim,
            String testingresults, int num) {
      if (num == 0) {
            FileWriter fwOb;
            PrintWriter pwOb;
            try {
                  fwOb = new FileWriter(testingresults, false);
                  pwOb = new PrintWriter(fwOb, false);
                  pwOb.flush();
                  pwOb.close();
                  fwOb.close();
            } catch (FileNotFoundException e) {

            } catch (IOException e) {

            }
      }

      int index = 0;
      int r, t = 0;
      char goal = 0;
      boolean found = false;

      while (!found && index < output_dim) {
            if (results[index] == 1) {
```

```
                found = true;
            }
            if (!(index + 1 > output_dim)) {
                if (!found) {
                    index++;
                }
            } else {
                break;
            }
        }
        r = index;
        if (index == output_dim) {
            r = -1;
        }

        found = false;
        index = 0;
        while (!found) {
            if (targets[index] == 1) {
                found = true;
            }
            index++;
        }
        t = index - 1;

        if(r != -1){
            if(results[r] == targets[t]) {
                r = t;
            }
        }

        FileWriter pw = null;
        try {
            pw = new FileWriter(testingresults, true);
            pw.write("Actual Output: " + t + "\n");
            for (int i = 0; i < output_dim; i++) {
                pw.write(targets[i] + " ");
            }

            pw.write("\nClassified Output: " + r + "\n");
            for (int i = 0; i < output_dim; i++) {
                pw.write(results[i] + " ");
            }
            pw.write("\n\n");
            pw.close();

        } catch (FileNotFoundException e) {
            System.err.print("File not found");
            System.exit(1);
        } catch (IOException e) {
            System.err.print("Error in I/O");
            System.exit(1);
        }
}
```

```
	public static void statistics(String testingresults) {
		BufferedReader br;
		String currentLine;
		String[] lineElements;
		double correctClassifications = 0;
		double totalClassifications = 0;

		try {
			br = new BufferedReader(new FileReader(testingresults));
			while ((currentLine = br.readLine()) != null) {
				lineElements = currentLine.split(" ");
				int actualNum = Integer.parseInt(lineElements[2]);
				br.readLine();
				currentLine = br.readLine();
				lineElements = currentLine.split(" ");
				int classifiedNum = Integer.parseInt(lineElements[2]);
				if (actualNum == classifiedNum) {
					correctClassifications++;
				}
				totalClassifications++;
				br.readLine();
				br.readLine();
			}
		} catch (IOException e) {
			System.out.println("Error in I/O");
			System.exit(1);
		}

		double accurateClassifications = correctClassifications
				/ totalClassifications;

		System.out.println("Total Correct Classifications: " +
correctClassifications);
		System.out.println("Total Classifications: " + totalClassifications);
		System.out.println("The perceptron classified "
				+ (accurateClassifications * 100)
				+ "% of the testing patterns correctly.");

	}

	/*
	 * Uses the input to a neuron and a threshold to determine the output of a
	 * neuron
	 *
	 * @param yin the input to the neuron
	 *
	 * @param threshold the threshold value that when achieved will cause the
	 * neuron to fire
	 *
	 * @return the output of the activation function (bipolar)
	 */
	public static int activationFunction(int yin, double threshold) {
		int toReturn = 0;
		if (yin > threshold) {
			toReturn = 1;
```

```
			} else if (yin < threshold) {
				toReturn = -1;
			}
			return toReturn;
		}

		/*
		 * Saves the weights to an output file
		 *
		 * @param input_dim the number of inputs to each output
		 *
		 * @param output_dim the number of output dimensions
		 *
		 * @param weights the list of weights to save
		 *
		 * @param destination the name of the output file
		 */
		public static void saveWeights(int input_dim, int output_dim,
				double[][] weights, String destination) {
			PrintWriter pw = null;
			try {
				pw = new PrintWriter(destination, "UTF-8");
				pw.println(input_dim);
				pw.println(output_dim + "\n");
				for (int i = 0; i < output_dim; i++) {
					for (int j = 0; j < input_dim + 1; j++) {
						pw.println(weights[i][j]);
					}
				}
			} catch (FileNotFoundException e) {
				System.err.print("File not found");
				System.exit(1);
			} catch (IOException e) {
				e.printStackTrace();
			}
			pw.close();
		}
}
```

References

Geitgey, Adam. *"Machine Learning Is Fun! Part 3: Deep Learning and Convolutional Neural Networks."* Medium. A Medium Corporation, 13 June 2016. Web. 24 Apr. 2017. <https://medium.com/@ageitgey/machine-learning-is-fun-part-3-deep-learning-and-convolutional-neural-networks-f40359318721>.

LeCun, Yann, Corinna Cortes, and Christopher J.C. Burges. *"THE MNIST DATABASE"* MNIST Handwritten Digit Database, Yann LeCun, Corinna Cortes and Chris Burges. N.p., Nov. 1998. Web. 24 Apr. 2017. <http://yann.lecun.com/exdb/mnist/>.

Redmon, Joseph. “MNIST in CSV.” Pjreddie. Joseph Chet Redmon, n.d. Web. 18 May 2017.