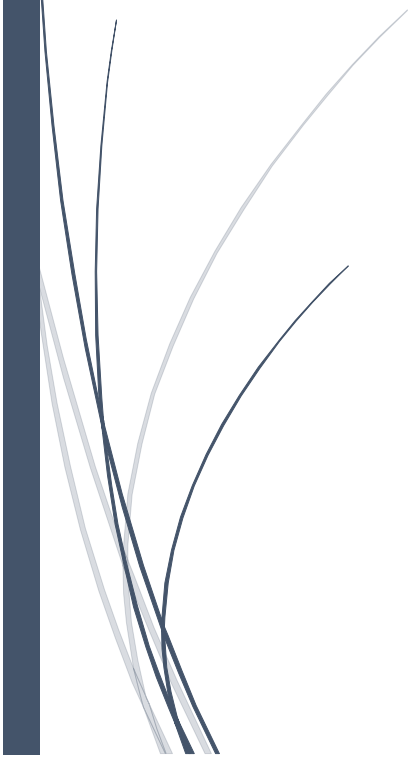
Neural Networks

3/13/2017

Project 3 Report



Braeden Brettin
and Pierre Balinda

Table of Contents

Contents

Table of Figures

Table of Tables

Table of Equations

1. Introduction to Neural Networks

Neural networks, along with genetic algorithms and fuzzy logic, are a critical area of advanced computational techniques. Using knowledge gained from a study of neurology, programmers can mimic the way neurons communicate with one another to perform advanced computational tasks otherwise unachievable through normal object-oriented programming methods. In the past, neural networks have primarily been used in the field of character recognition, although advances in the field of software engineering and neurology have enabled programmers to attempt far more difficult tasks.

In our brains, neurons relate to thousands of other neurons, sending electrochemical signals to communicate with each other. Inputs are received in the neurons through synapses, and the neuron itself sums up the various inputs and fires if the output is greater than a threshold value. If the neuron fires, it will output a signal along an axon, which will then input to other neurons (*Neural Networks in Plain English*, p. 2). The outline of this neuron is as shown in Figure 1 (, below. Artificial neural networks model this behavior on a much smaller scale.

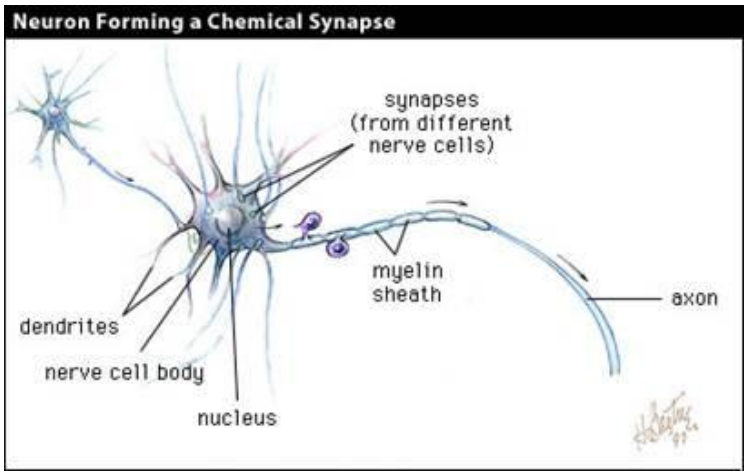


Figure 1: Neuron Anatomy

To model the hierarchy of a neural network, a layer of neurons is created which will take in the inputs to the program. These inputs are the various values of the defined parameters that will determine the final output result. The outputs of these neurons are then combined as inputs to another layer of neurons called the hidden layer. By defining weights for each of these inputs/outputs, an output can be derived for each of the neurons in the hidden layer using one of several types of functions, the most common of which being the sigmoid function, a more realistic deviation of a step function. The outputs from the hidden layer are then combined into the final neuron, which calculates the final output of the neural networks.

As it exists right now, the network is far from ideal. With weights having randomly been assigned to each input, the programmer has little to no idea of the importance of each parameter in regards to the final output. As such, the programmer must 'train' the network. This training occurs in the form of both unsupervised and supervised training. For the purposes of this project, the team will address only supervised training, as this process will be carried out to train the team's network. This training is achieved by inputting values for an example for which the team knows the output. By finding the difference between the network's calculated output and the expected output, the team can adjust the weights of the various inputs. The full calculations will be derived from Miller's work on neural networks (p. 1). This process is repeated until the network gives reliable results.

With the theory behind neural networks discussed, the team can proceed into the creation of the actual network. The creation of an artificial neural network involves three major steps:

1. Define the parameters for the system. These parameters are a set of variables which will be used to calculate the final output.
2. Create the hierarchy of the network. This step involves creating classes for the neuron, layer, and network.
3. Train the network using unsupervised or supervised learning. The team's project will use supervised learning, as it is far easier to explain and is more useful in displaying the modeling of an artificial network.

With these steps completed, the team can now input any set of values into the network and achieve a desirable, reliable, and realistic outcome.

2. **Brainstorming**

As previously mentioned, neural networks are most well-known for their use in character recognition. However, the team wanted to approach this technique from a different perspective. As such, the team desired to model a program with significant real-world applications. Having recently watched 'The Big Short,' a film dealing with the housing market collapse of 2008, the team had an interest in housing bonds. After researching these bonds further, the team found the topic to be an extremely pertinent one. No set formula exists to determine the credit grade of a bond; however, factors such as liquidity, credit score, and earnings play an important role in determining the grade of the bond (*Standard & Poor's Definitions*, p. 1). The team ultimately decided to create a neural network that would specialize in determining the grade of a bond given several parameters.

To begin, the team created an input layer of neurons, each neuron taking in one input that represented the value of a parameter. Each input would then be passed through a positive or negative linear function that would return a value between 0 and 1. This value would then be passed to the hidden layer. The hidden layer consists of three neurons, each taking in every output from the input layer. However, those parameters which are more similar, such as earnings and credit score, would be given a greater weight at one of the hidden layer neurons. The sum of the inputs multiplied by the weights would then be passed through a sigmoid function to return a value between 0 and 1. The average of these outputs is the result of the output layer, with each score corresponding to a credit grade, as shown in Figure 2, below.

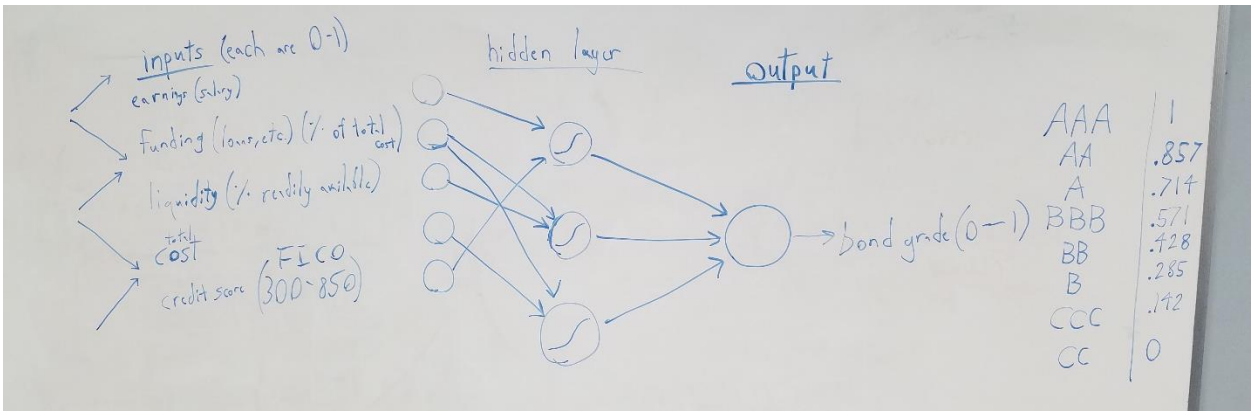


Figure 2: Neural Network Layers

After determining the hierarchy of the neural network, the team set about brainstorming the programming side of the network. To code this network, three classes must be created: the neuron class, the layer class, which holds a group of neurons, and the network class, which holds a group of layers. The variables and methods found in each class are shown in Figure 3, below.

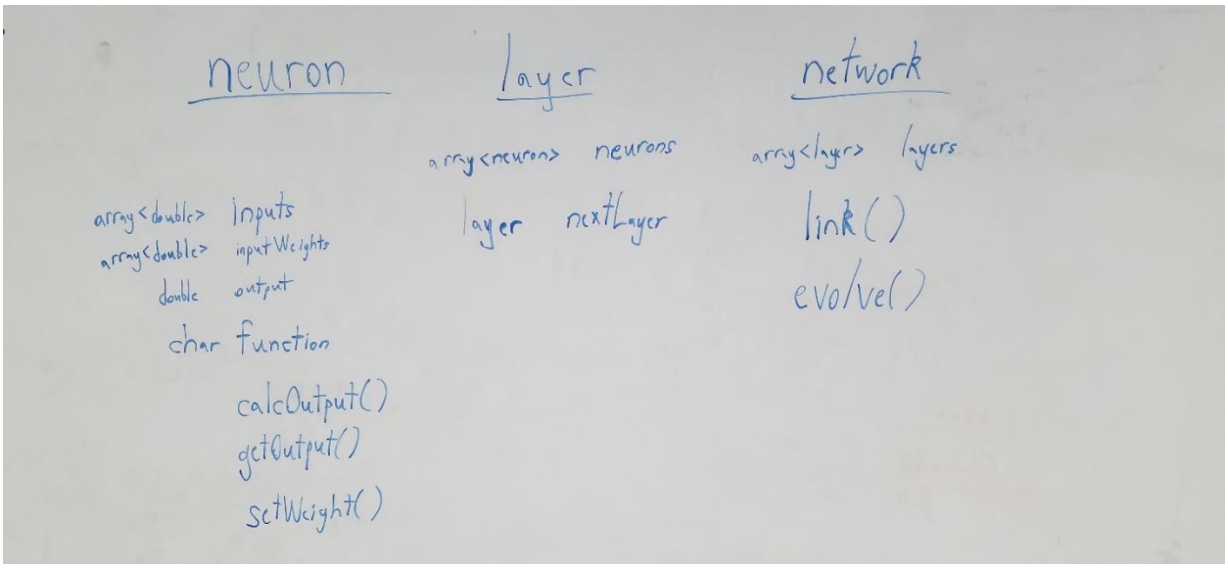


Figure 3: Neural Network Classes

With the hierarchy and programming structure of the neural network defined, the team can now proceed to creating the various classes needed for this network. The team will walk through each of the three major steps required to create a neural network in greater detail as this report progresses: defining the parameters, creating the network, and training the network. The team will begin by defining the parameters required for this network.

3. Defining the Parameters

As previously mentioned, the parameters used to identify the grade of bond are credit score, liquidity, and earnings. To accurately determine the viability of the bonds, the team added two parameters, funding and total cost. The credit score parameter ranges from 300 to 850, 300 means that the person or entity does not have enough credit history to gage their credit worthiness and 700-850 that the person or entity is credit worthy. The liquidity refers the cash available, earnings refers to salary and/or any other income, funding refers to the amount of loans that the person already has, and the total cost refers to the cost of the investment.

To establish linear functions that will translate the values of parameters, the team set a minimum and maximum value for each parameter as shown in table 1.

Table 1: Maximum and Minimum Values of Parameters

Parameter	Minimum value	Maximum Value
Credit score	300	850
Earnings (Salary)	30,000	1,000,000
Funding (Loans)	40,000	900,000
Liquidity	45,000	900,000
Cost	40,000	1,000,000

To translate the values of the above parameter from a 0 to 1 scale the team used equations 1, shown below. This equation is used by the input layer.

$$Output = \frac{input\ value - min}{\max - min}$$

Equation 1: Linear function equation

As previously mentioned above, the neurons in our network mimic the human neuron. For a neuron to transmit an incoming signal/output, the output value must be greater than its activation value which is computed using equation 2. In equation 2, x represents the incoming signal value and w its weight.

$$Activation = \sum_{k=0}^{n} \mathrm{x} * \mathrm{w}$$

Equation 2: Neuron activation value calculation

The activation value is then used by the sigmoid function to compute the output values of the hidden and output layer, as shown below.

$$Output = \frac{1}{1 - e^{-activation}}$$

Equation 3: Sigmoid function

4. Creating the Network

4.1. Neuron Class

To create the network, the team started by its basic unit, the neuron. The neuron class has seven variables; max, min, an input array, a weights array, output, and activation are of double type. The seventh class variable is func, which represents the function of the neuron.

```
1 public class Neuron
2 {
3     double max;
4     double min;
5     double[] inputs  = new double[5];
6     double[] weights = new double[5];
7     double output;
8     double activation = 0;
9     char func;
```

Figure 4: Neuron class variables

The neuron class constructor takes in four variables, an array of double for the inputs, a character that represent the neuron function, and two double variables that represent the minimum and maximum values of the inputs. The team decided to use an array of double because the hidden and output layers take in more than one input, unlike the input layer. After assigning the passed

parameters to their respective class variables, the weight of each input was assigned a random number, as shown in figure 5.

```
10    public Neuron(double[] inputsTemp, char function,double minT, double maxT)
11    {
12        inputs = inputsTemp;
13        output = 0;
14        func= function;
15        min = minT;
16        max = maxT;
17        for(int i=0; i<inputs.length; i++)
18                weights[i] = Math.random();
19
20
```

Figure 5: Neuron constructor

The team proceed to writing the method that computes the output of the neuron, as shown in figure 6. The activation value is computed using equation 2, which is the summation of the product of every input and its weight. The function method, which computes the output based on the neuron's function type, is then called.

```
37    protected void calcOutput()
38    {
39        for(int i=0; i<inputs.length; i++)
40            activation +=inputs[i]*weights[i];
41        function();
42    }
```

Figure 6: Calculate output method

The function method checks the character class variable func to determine the appropriate function. These functions, as previously mentioned, are positive linear, negative linear, and sigmoid. They all return a double value which is assigned to the output.

```
26    protected void function()
27    {
28        if(func =='p')
29            output = linearPos();
30
31        else if (func =='n')
32            output = linearNeg();
33
34        else if(func =='s')
35            output = sigmoid();
36    }
```

Figure 7: Function method

The neuron function return a double value which is computed using equation 2 and 3, as shown in figure 8. The linear equations use the first value in the input array because they used by neurons that has only one input.

```
60    protected double sigmoid()
61    {
62        return 1/(1 + Math.pow(Math.E, -activation));
63    }
64    protected double linearNeg()
65    {
66        return  -(inputs[0]-min)/(max-min);
67    }
68    protected double linearPos()
69    {
70        return  (inputs[0]-min)/(max-min);
71    }
```

Figure 8: Neuron equation methods

After writing the methods that computes the output, the team proceed to writing the getter methods, as shown in figure 9. The output getter method calls the calculation method to update the output class variable. The output is then compared to the activation value, if the output is

greater than the neuron's activation, the output is returned, otherwise 0 is returned meaning the neuron was not activated. The weight getter methods simply return the array of double class variable, weights.

```
43      protected double getOutput()
44      {
45          calcOutput();
46          if(output > activation)
47                  return output;
48          else
49              return 0;
50      }
51      protected double[] getWeight()
52      {
53          return weights;
54      }
```

Figure 9: Getter methods

4.2. Network Class (client class)

The team then proceeded to creating the network class that encompasses the neurons and layers. The network class has three arrays of neurons that represents the three different layers of the network and an array of doubles that will hold entries. The array datatype was chosen over the arrayList datatype because the arrayList was causing memory issues.

```
1 import java.util.Scanner;
2
3
4 public class Network
5 {
6     static  Neuron[] inputLayer = new  Neuron[5];
7     static  Neuron[] hiddenLayer = new  Neuron[3];
8     static  Neuron[] outputLayer = new  Neuron[1];
9     static  double[] entries = new  double[5];
```

Figure 10: Network class variables

The first part of the network is to get values from the users, the values the program is looking for are earnings, funding, liquidity, cost, and credit score, all of which are of the double datatypes. These values are then put in the global array, entries, as shown in figure 11.

```
29    public static void inputSet()
30    {
31        Scanner in = new Scanner(System.in);
32        System.out.println("Enter earnings per year: ");
33        double earning = in.nextDouble();
34        System.out.println("Enter funding: ");
35        double funding = in.nextDouble();
36        System.out.println("Enter liquidity: ");
37        double liquidity = in.nextDouble();
38        System.out.println("Enter cost: ");
39        double cost = in.nextDouble();
40        System.out.println("Enter credit score: ");
41        double creditScore = in.nextDouble();
42        entries[0] = (earning);
43        entries[1] = (funding);
44        entries[2] = (liquidity);
45        entries[3] = (cost);
46        entries[4] = (creditScore);
47        in.close();
48    }
```

Figure 11: Get user inputs

The entries are then put in the input layer along with the values' respective maximum, minimum, and function type values. The positive linear, negative linear, and sigmoid functions are represented by p, n, and s. In this case, the input layer's neuron only use the positive and negative linear functions. The earning, liquidity, and credit score parameters have positive linear functions because the higher their values are, the more likely they are to pay back the loan. On the other hand, the funding and total cost parameters have negative linear functions because, the higher their values are, the more unlikely they are to pay back the loan.

The input values are entered in their respective neurons along with their minimum, maximum, and function type values, as shown in figure 12. The output individual neurons are then computed and put in the hidden layer.

```
49      public static void InputLayer()
50      {
51          double [] min={30000,40000,45000,40000,250};
52          double [] max={1000000, 9000000,90000,1000000,850};
53          char[] functions = {'p','n','p','n','p'};
54          for(int i=0; i<entries.length; i++)
55          {
56               double[] temp = new  double[1];
57              temp[0] = entries[i];
58              inputLayer[i] = new Neuron(temp, functions[i],min[i], max[i]);
59
60          }
61          for(int i=0; i>inputLayer.length;i++)
62              inputLayer[i].calcOutput();
63
64      }
```

Figure 12: Input layer

Once the outputs of the input layers are computed the hidden layer method is called and takes in the outputs as inputs. The team decided to have a single layer in the hidden layer as shown in figure 13. The hidden layer's output is computed using the sigmoid function, equation 3.

```
65      public static void HiddenLayer()
66      {
67          int size = inputLayer.length;
68          for(int i=0; i<3 ;i++)
69          {
70               double[] temp = new  double[2];
71              temp[0] = (inputLayer[i].getOutput());
72              temp[1] = (inputLayer[Math.abs(i-size+1)].getOutput());
73              hiddenLayer[i] = new Neuron(temp,'s',0,0);
74          }
75      }
```

Figure 13: Hidden layer

The output values of the hidden layer are then computed and input in the output layer. The output layer has one neuron, as shown in figure 14. The output layer method returns its output value as a double.

```
76      public static double OutputLayer()
77      {
78           double[] temp = new  double[3];
79          for(int i=0; i<hiddenLayer.length;i++)
80              temp[i] = (hiddenLayer[i].getOutput());
81          outputLayer[0] = new Neuron(temp,'s',0,0);
82          return outputLayer[0].getOutput();
83
84      }
```

Figure 14: Output layer

The output layer's returned value is then passed as a parameter in the print method, shown in figure 15. The print method outputs the corresponding bond string of the passed value. The output is broken in intervals of 1/7. We divide by 7 because we have 7 possible ratings.

```
85      public static void print(double result)
86      {
87          String out;
88          if(result <= 1 && result >= 0)
89              {
90              if(result > .857)
91                  out = "AAA";
92              else if(result > .714)
93                  out = "AA";
94              else if(result > .571)
95                  out = "A";
96              else if(result > .428)
97                  out = "BBB";
98              else if(result > .285)
99                  out = "B";
100             else if(result > .142)
101                 out = "CCC";
102             else
103                 out = "CC";
104             }
105         else
106             out = "Invalid output";
107         System.out.println(out + "\t" + result);
108     }
```

Figure 15: Print method

The methods were then combined and run in the main method, as shown below. The first method gets the network's input from the user, the second runs the input layer of the network, the third runs the hidden layer of the network, the fourth runs the output layer, and the fifth translates the output into a letter bond rating.

```
10          double result=0, output = .6, deviation = 0;
11          inputSet();
12          System.out.println("Grade \tOutput");
13          do
14          {
15              InputLayer(deviation);
16              HiddenLayer(deviation);
17              result = OutputLayer(deviation);
18              print(result);
19              if(result<output)
20                  deviation+=.05;
21              else
22                  deviation-=.05;
23          }
24          while(result<output);
```

Figure 16: Network main methods

5. Training the Network

After completing the network's code, the team proceeded to train it. The team ran the network program and input the values shown below. The team was expecting an A bond rating, but got a CC rating instead. The team proceed to modify the network by adding a backpropagation i.e. the program will run until the desired output value is reached.

```
Grade   Output
Enter earnings per year: 70000
Enter funding: 200000
Enter liquidity: 120000
Enter cost: 200000
Enter credit score: 800
CC      0.0
```

Figure 17: First run of the program

To ensure that the program will not get stuck in an endless loop, the team added a set weights method which takes in a deviation parameter to adjust the current weights of the input values. The deviation will be increased or decreased with respect to the output.

```
55    protected void setWeight(double deviation)
56    {
57        for(int i=0; i<inputs.length; i++)
58            weights[i] +=  deviation;
59    }
```

Figure 18: Set weight method

To do so, the team started by adding a condition to the neuron constructor method that checks the value of the randomly generated number, as shown in figure 18. The condition insures that the weight of each neuron is greater than 0.05 which in turn gives a better neuron activation value.

```
10    public Neuron(double[] inputsTemp, char function,double minT, double maxT)
11    {
12        inputs = inputsTemp;
13        output = 0;
14        func= function;
15        min = minT;
16        max = maxT;
17        for(int i=0; i<inputs.length; i++)
18        {
19            double temp = Math.random();
20            if(temp > .05)
21                weights[i] = temp;
22            else
23                i--;
24        }
25    }
```

Figure 19: Modified neuron constructor

The input layer, hidden layer, output layer, and main methods were modified to consider the deviation. The deviation starts out at 0 and gets incremented or decremented depending on the if condition that is on line 19 in figure 20. A do-while loop was also added to keep the program running until the A rating is reached.

```
10        double result=0, output = .6, deviation = 0;
11        System.out.println("Grade \tOutput");
12        do
13        {
14            inputSet();
15            InputLayer(deviation);
16            HiddenLayer(deviation);
17            result = OutputLayer(deviation);
18            print(result);
19            if(result<output)
20                deviation+=.05;
21            else
22                deviation-=.05;
23        }
24        while(result<output);
25    }
```

Figure 20: Refactored main method

The input, hidden, and output layer methods were changed to having a double parameter passed and the set weight class method of the neuron was called after every output computation.

```
47    public static void InputLayer(double deviation)
48    {
49        double [] min={30000,40000,45000,40000,250};
50        double [] max={1000000, 9000000,90000,1000000,850};
51        char[] functions = {'p','n','p','n','p'};
52        for(int i=0; i<entries.length; i++)
53        {
54             double[] temp = new  double[1];
55            temp[0] = entries[i];
56            inputLayer[i] = new Neuron(temp, functions[i],min[i], max[i]);
57
58        }
59        for(int i=0; i>inputLayer.length;i++)
60        {
61            inputLayer[i].setWeight(deviation);
62            inputLayer[i].calcOutput();
63        }
64    }
```

Figure 21: Modified input layer method

```
65    public static void HiddenLayer(double deviation)
66    {
67        int size = inputLayer.length;
68        for(int i=0; i<3 ;i++)
69        {
70             double[] temp = new  double[2];
71            temp[0] = (inputLayer[i].getOutput());
72            temp[1] = (inputLayer[Math.abs(i-size+1)].getOutput());
73            hiddenLayer[i] = new Neuron(temp,'s',0,0);
74            hiddenLayer[i].setWeight(deviation);
75        }
76    }
```

Figure 22: Modified hidden layer method

```
77      public static double OutputLayer(double deviation)
78      {
79           double[] temp = new  double[3];
80          for(int i=0; i<hiddenLayer.length;i++)
81              temp[i] = (hiddenLayer[i].getOutput());
82          outputLayer[0] = new Neuron(temp,'s',0,0);
83          outputLayer[0].setWeight(deviation);
84          return outputLayer[0].getOutput();
```

Figure 23: Modified output layer method

6. The Final Product

After modifying the code the program was run and produced the A rating bond that the team was looking for. The team input the previous data in the modified program and received the A rating, as shown in figure 24.

```
Enter earnings per year: 70000
Enter funding: 200000
Enter liquidity: 120000
Enter cost: 200000
Enter credit score: 800
Grade   Output
CC      0.0
A       0.621477939857002
```

Figure 24: Program output

After training the network, the program was ready to receive and compute the bond ratings of loans.

7. References

Detweiler, G. (Dec. 8, 2016). What is a Good Credit Score? Retrieved Mar. 1, 2017, from https://www.credit.com/credit-scores/what-is-a-good-credit-score/

iamtrask. (July 12, 2015). A Neural Network in 11 Lines of Python (Part 1). Retrieved Feb. 28, 2017, from http://iamtrask.github.io/2015/07/12/basic-python-network/

Miller, S. (Aug. 10, 2015). Mind: How to Build a Neural Network (Part One). Retrieved March 1, 2017, from https://stevenmiller888.github.io/mind-how-to-build-a-neural-network/

Nielsen, M. (Jan. 2017). Using Neural Nets to Recognize Handwritten Digits. Retrieved March 1, 2017, from http://neuralnetworksanddeeplearning.com/chap1.html

Stergiou, C. & Siganos, D. Neural Networks. Retrieved March 2, 2017, from https://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol4/cs11/report.html

Example of a Neural Network. Retrieved Feb. 28, 2017, from http://www.jmp.com/support/help/Example_of_a_Neural_Network.shtml

Neural Networks in Plain English. Retrieved Feb. 26, 2017, from http://www.ai-junkie.com/ann/evolved/nnt1.html

(Sep. 20, 2010). Standard & Poor's Definitions. Retrieved Mar. 3, 2017, from http://www.bankersalmanac.com/addcon/infobank/credit_ratings/standardandpoors.aspx

8. Appendix

8.1. Neuron.java

```
9.  public class Neuron
10. {
11.         double max;
12.         double min;
13.         double[] inputs  = new double[5];
14.         double[] weights = new double[5];
15.         double output;
16.         double activation = 0;
17.         char func;
18.         public Neuron(double[] inputsTemp, char function,double minT, double maxT)
19.         {
20.                 inputs = inputsTemp;
21.                 output = 0;
22.                 func= function;
23.                 min = minT;
24.                 max = maxT;
25.                 for(int i=0; i<inputs.length; i++)
26.                 {
27.                         double temp = Math.random();
28.                         if(temp > .05)
29.                                 weights[i] = temp;
```

```
30.                         else
31.                                 i--;
32.                 }
33.         }
34.         protected void function()
35.         {
36.                 if(func =='p')
37.                         output = linearPos();
38.
39.                 else if (func =='n')
40.                         output = linearNeg();
41.
42.                 else if(func =='s')
43.                         output = sigmoid();
44.         }
45.         protected void calcOutput()
46.         {
47.                 for(int i=0; i<inputs.length; i++)
48.                         activation +=inputs[i]*weights[i];
49.                 function();
50.         }
51.         protected double getOutput()
52.         {
```

```
53.                 calcOutput();
54.                 if(output > activation)
55.                                 return output;
56.                 else
57.                                 return 0;
58.         }
59.         protected double[] getWeight()
60.         {
61.                 return weights;
62.         }
63.         protected void setWeight(double deviation)
64.         {
65.                 for(int i=0; i<inputs.length; i++)
66.                                 weights[i] +=  deviation;
67.         }
68.         protected double sigmoid()
69.         {
70.                 return 1/(1 + Math.pow(Math.E, -activation));
71.         }
72.         protected double linearNeg()
73.         {
74.                 return  -(inputs[0]-min)/(max-min);
75.         }
```

```
76.         protected double linearPos()
77.         {
78.                 return  (inputs[0]-min)/(max-min);
79.         }
80. }
```

8.2. Network.java

```
import java.util.Scanner;
public class Network
{
	static  Neuron[] inputLayer = new  Neuron[5];
	static  Neuron[] hiddenLayer = new  Neuron[3];
	static  Neuron[] outputLayer = new  Neuron[1];
	static  double[] entries = new  double[5];
	public static void main(String[] args)
	{
		double result=0, output = .6, deviation = 0;
		inputSet();
		System.out.println("Grade \tOutput");
		do
		{
			InputLayer(deviation);
			HiddenLayer(deviation);
			result = OutputLayer(deviation);
			print(result);
			if(result<output)
				deviation+=.05;
			else
				deviation-=.05;
```

```
        }
        while(result<output);
}
public static void inputSet()
{
        Scanner in = new Scanner(System.in);
        System.out.print("Enter earnings per year: ");
        double earning;
        earning = in.nextDouble();
        System.out.print("Enter funding: ");
        double funding = in.nextDouble();
        System.out.print("Enter liquidity: ");
        double liquidity = in.nextDouble();
        System.out.print("Enter cost: ");
        double cost = in.nextDouble();
        System.out.print("Enter credit score: ");
        double creditScore = in.nextDouble();
        entries[0] = (earning);
        entries[1] = (funding);
        entries[2] = (liquidity);
        entries[3] = (cost);
        entries[4] = (creditScore);
        in.close();
```

```
}
public static void InputLayer(double deviation)
{
        double [] min={30000,40000,45000,40000,250};
        double [] max={1000000, 9000000,90000,1000000,850};
        char[] functions = {'p','n','p','n','p'};
        for(int i=0; i<entries.length; i++)
        {
             double[] temp = new  double[1];
            temp[0] = entries[i];
            inputLayer[i] = new Neuron(temp, functions[i],min[i], max[i]);

        }
        for(int i=0; i>inputLayer.length;i++)
        {
            inputLayer[i].setWeight(deviation);
            inputLayer[i].calcOutput();
        }
}
public static void HiddenLayer(double deviation)
{
        int size = inputLayer.length;
        for(int i=0; i<3 ;i++)
```

```
        {
                 double[] temp = new  double[2];
                temp[0] = (inputLayer[i].getOutput());
                temp[1] = (inputLayer[Math.abs(i-size+1)].getOutput());
                hiddenLayer[i] = new Neuron(temp,'s',0,0);
                hiddenLayer[i].setWeight(deviation);
        }
}
public static double OutputLayer(double deviation)
{
         double[] temp = new  double[3];
        for(int i=0; i<hiddenLayer.length;i++)
                temp[i] = (hiddenLayer[i].getOutput());
        outputLayer[0] = new Neuron(temp,'s',0,0);
        outputLayer[0].setWeight(deviation);
        return outputLayer[0].getOutput();

}
public static void print(double result)
{
        String out;
        if(result <= 1 && result >= 0)
                {
```

```
                    if(result > .857)
                            out = "AAA";
                    else if(result > .714)
                            out = "AA";
                    else if(result > .571)
                            out = "A";
                    else if(result > .428)
                            out = "BBB";
                    else if(result > .285)
                            out = "B";
                    else if(result > .142)
                            out = "CCC";
                    else
                            out = "CC";
                    }
            else
                    out = "Invalid output";
            System.out.println(out + "\t" + result);
    }


}
```