NEURAL NETWORKS

Siddhesh Kuvalekar (001238765)

Sneha Pai (001884109)

08-09-2018

TABLE OF CONTENTS

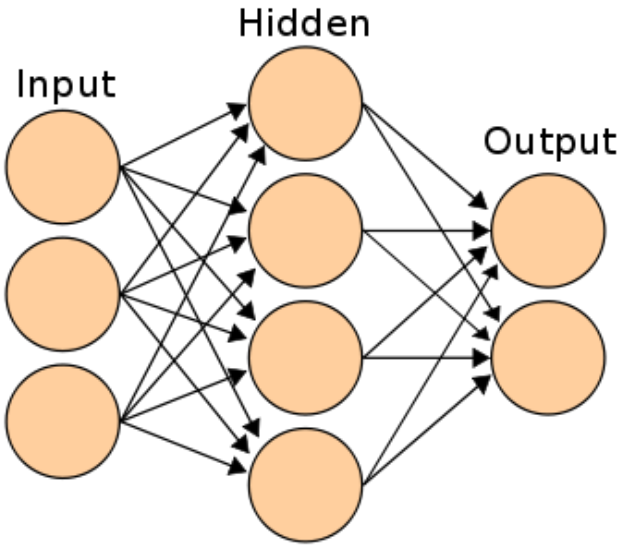
INTRODUCTION

Neural Network is a computational model based on the structure and functions of biological neural networks. Information that flows through the network affects the structure of the neural network because a neural network changes - or learns, in a sense - based on that input and output.

The most important advantage of neural networks is that it can actually learn from observing datasets. Neural Network can be used as a random function approximation tool which help estimate methods for arriving at solutions while defining computing functions or distributions.

ANNs have three layers that are interconnected. The first layer consists of input neurons. Those neurons send data on to the second layer, which in turn sends the output neurons to the third layer.

Training an artificial neural network involves choosing from allowed models for which there are several associated algorithms.



OBJECT DESCRIPTION

Perceptron

Perceptron is the basic building block of the neural network. It is modelled after the neuron present in a human brain. Like biological neuron it takes in many inputs and provides output after passing the weighted sum of the input through the activation function.

The perceptron object in the project consist of following variables –

1. List of input synapses: List used to store the synapses i.e a list of connections that the other perceptron is connected to
2. activation function object: Stores an instance of the activation function type object (sigmoid or linear) that the perceptron uses to generate the output from the weighted sum.
3. Output: Used to store the output of program.
4. derivative of the output: Output of the program after the weighted sum is passed through the derivative of the activation function. Used during back propagation.
5. weighted sum value: The value found after multiplying the inputs with their corresponding weight and summing the individual product.
6. error value: the difference in the expected output and the actual output of neuron

The object also consists of following object that operate on data function –

1. Calculateweightedsum
2. Activate
3. Getweights
4. Addinput

Synapse

The synapse object is used to store the connection between two perceptron. The object consist of following variable:

- Source Perceptron: used to store the source perceptron object that the synapse is used to connect.
- Weight: Each connection is associated with a weight that is used to multiply to the input which passes through it

Layer

The layer class is used to build various layer of the neural network. This object instance is then used as necessary as an input layer, one or many hidden layer and output layer. The layer class performs the important operation of calculating the feed forward output of the entire later by traversing all the perceptron in the layer and calling them activate function

The layer class consist of the following variables:

1. List<Perceptron> perceptrons
2. Layer previousLayer
3. Layer nextLayer
4. private Perceptron bias

Following functions as included in the class:

1. Add perceptron
2. feedforward

Neural Network

The neural network class holds the complete neural network comprised of various layers and perceptron. The neural network class stores the input layer, output layer and list of hidden layers. Provided with input the class is used to get the predicted output of the neural network.

Neural Network class consist of the variable:

1. name;
2. List<Layer> layers
3. Layer input
4. Layer output

Functions included in the class are :

1. addLayer
2. getWeights
3. setInputs
4. getOutput
5. reset

ACTIVATION FUNCTION

Theory

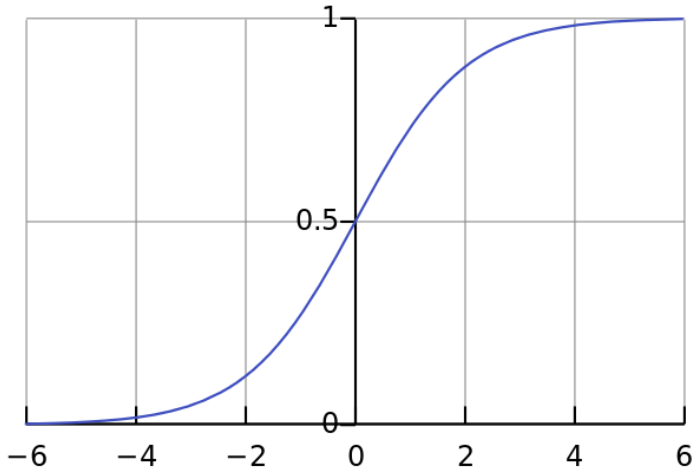
In a biologically inspired neural network, the output of a neuron is usually an abstraction representing the rate of action potential firing in the cell. In its simplest form, this is binary value, i.e., either the neuron is firing or not. Hence, the need for normalization of this output value.

To achieve this normalization we apply what is known as an activation function to the output of the neuron. If we take the example of a really simple Heaviside step function which assigns a 0 to any negative value and a 1 to any positive value, then a large number of neurons would be required to achieve the required granularity of slowly adjusting the weights to reach an acceptable consensus of the training set.

this concept of slowly adjusting the weights can be represented mathematically as the slope of the activation function. In biological terms, it can be thought of as the increase in firing rate that occurs as input current increases. If we were to use a linear function instead of the Heaviside function, then we would find that the resulting network would have an unstable convergence because neuron inputs along favored paths would tend to increase without bound, as a linear function is not normalizable.

All problems mentioned above can be handled by using a normalizable sigmoid activation function. One realistic model stays at zero until input current is received, at which point the firing frequency increases quickly at first, but gradually approaches an asymptote at 100% firing rate.

If plotted on a graph, the Sigmoid function draws an S shaped curve:



Thus, the final formula for the output of a neuron now becomesOutput=11+e−(∑weighti.inputi)Output=11+e−(∑weighti.inputi)

There are other normalization functions that we can use but the sigmoid has the advantage of being simple and having a simple derivative which will be useful when we look at the back propagation below.

Implementation

The Activation function is implemented in the project as an interface consisting of two methods i.e activate and derivative. This interface is then implemented by two classes SigmoidFUnction and LinerFunction. Each of the class contain their activation formula in the activation formula and the derivative formula in the derivative function.

Back Propagation

Theory

During the training cycle, we adjusted the weights depending on the error. To do this, we can use the "Error weighted derivative" formula

Adjustment=error.input. SigmoidCurveGradient(output)

The reason we use this formula is that firstly, we want to make the adjustment proportional to the size of the error. Secondly, we multiply by the input, which is either a 0 or a 1. If the input is 0, the weight isn't adjusted. Finally, we multiply by the gradient of the Sigmoid curve (or the derivative).

The reason that we use the gradient is because we are trying to minimize the loss. Specifically, we do this by a gradient descent method. It basically means that from our current point in the parameter space (determined by the complete set of current weights), we want to go in a direction which will decrease the loss function. Visualize standing on a hillside and walking down the direction where the slope is steepest. The gradient descent method as applied to our neural net is illustrated as follows:

1. If the output of the neuron is a large positive or negative number, it signifies the neuron was quite confident one way or another.
2. From the sigmoid plot, we can see that at large numbers the Sigmoid curve has a shallow gradient.
3. Thus, if the neuron is confident that the existing weight is correct, it doesn't want to adjust it very much and multiplying by the gradient of the sigmoid curve achieves this.

The derivative of the sigmoid function is given by the following formula

$$\text{SigmoidCurveGradient(output)} = \text{output}.(1-\text{output})$$

Substituting this back into the adjustment formula gives us

$$\text{Adjustment} = \text{error}.\text{input}.\text{output}.(1-\text{output})$$

Implementation

The BackPropagation class is used in the training phase of the neural network. It takes in the neural network object, input array and output array to adjust the weights and errors of each neuron in the neural network. It does the adjustment based on the logic as explained above in the concept.

The variables contained in the class are :

1. NeuralNetwork neuralNetwork;
2. double learningRate : the rate determines the how fast or slow the the weights of the network are adjusted. A high value would oscillate the values of weights a lot whereas a low value would increase the time taken to arrive at the optimum value.
   it is initialized to 0.3.

The function in the class are:

1. backpropagate: it takes in the entire input and output values in the form of 2D arrays and uses it to adjust the weights of the neural network.
2. Error: the function is used of the backpropagate method to calculate MSE error between actual and predicted output.

TESTING

The Neural network is tested extensively using the openly found MNIST database from Kaggle.com

The MNIST dataset consist of handwritten digits image data in terms of greyscale pixel values ranging from 0 to 255. Each row consists of 784 columns representing the pixel data of the image. The data is available as comma separated values file. The training dataset used in the project consist of 10000 rows and the test dataset consist of 100 rows.

Neural network: The neural network for the dataset is consist of the 3 layers – one input layer consisting of 784 neurons (each for a pixel value), one hidden layer, and an output layer consisting of 10 neurons(one of each of the digits to be predicted). The number of neurons in the hidden layer is calculated as 2/3 * 784 + 10.

The input layer as well as the hidden layer consist of a bias neuron to adjust the zero bias of the layer. The bias neurons are initialized with output values of 1 and liner activation function.

Each of the neuron in the input layer is traversed and initialized to the output values corresponding to the pixel value input.

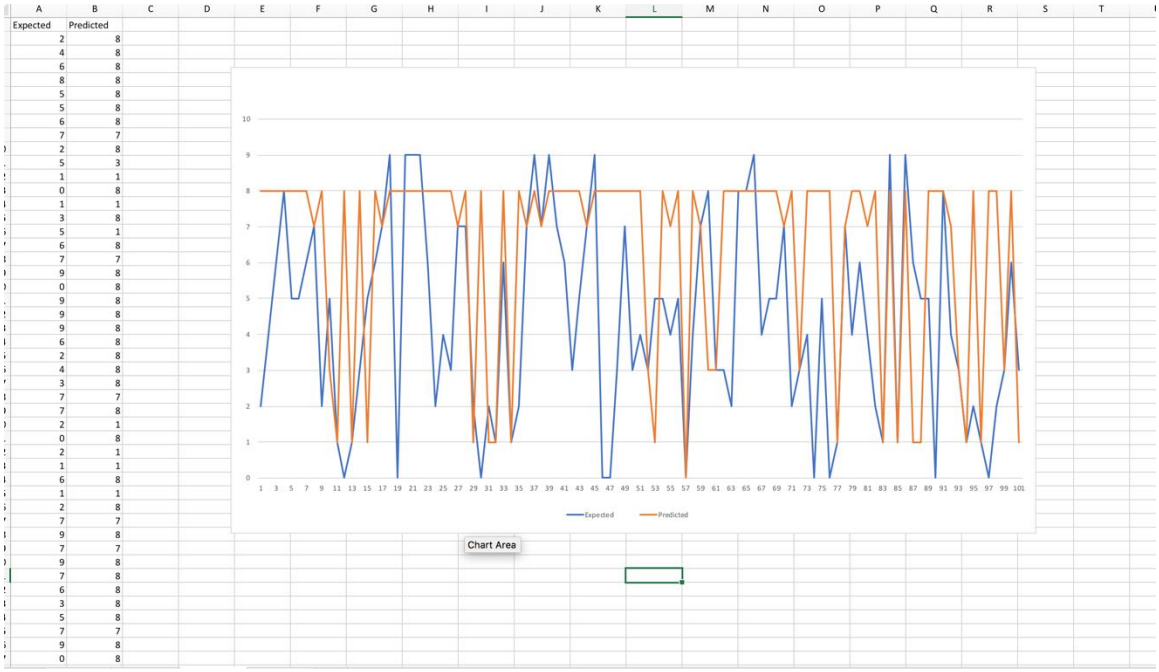
OTSU Algorithm: The input image values consisting of the greyscale range were difficult for the network to identify. The greyscale values were hence converted to binary values using the OTSU algorithm.

The neural network is then trained on 10,000 image values for adjusting the weights of the neural network. The training cycle is then repeated three times to increase the accuracy of the neural networks.

Once the network is trained the testing dataset is read and passed to the neural network for prediction. The output of the program thus shows the predicted values and the actual values for the image values in the testing dataset.

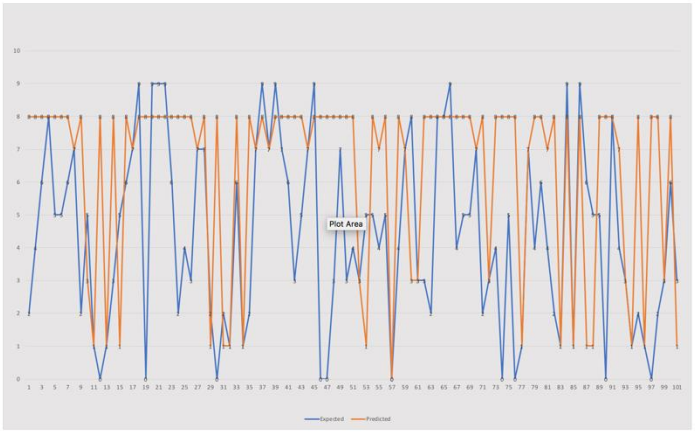
OBSERVATION

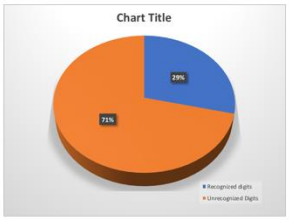
The following graph shows the expected output and the predicted output on running the program for 101 records. T.he accuracy achieved in predicting the correct output is 28%



Expected	Predicted
2	8
4	8
6	8
8	8
5	8
5	8
6	8
7	7
2	8
5	3
1	1
0	8
1	1
3	8
5	1
6	8
7	7
9	8
0	8
9	8
9	8
9	8
6	8
2	8
4	8
3	8
7	7
7	8
2	1
0	8
2	1
1	1
6	8
1	1
2	8
7	7
9	8
7	7
9	8
7	8
6	8
3	8
5	8
7	7

Total no.of digits tested	Recognized digits	Unrecognized Digits
101	29	72

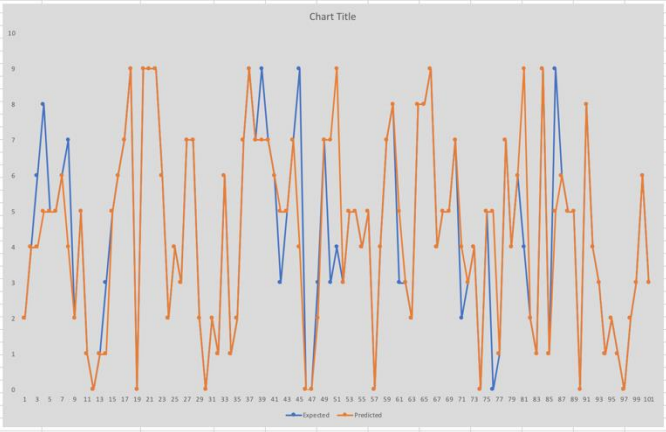


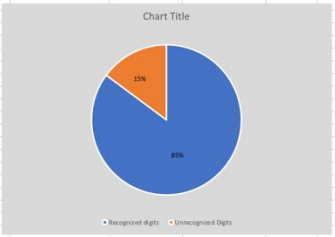


The following results are observed after the dataset is converted from grayscale to black and white using the otsu() method. It is observed that the accuracy of prediction to identify the correct digit is more than the accuracy derived for previous observations without the conversion of dataset from grayscale to binary.

Expected	Predicted
2	2
4	4
6	4
8	5
5	5
5	5
6	6
7	4
2	2
5	5
1	1
0	0
1	1
3	1
5	5
6	6
7	7
9	9
0	0
9	9
9	9
9	9
6	6
2	2
4	4
3	3
7	7
7	7
2	2
0	0
2	2
1	1
6	6
1	1
2	2
7	7
9	9
7	7
9	7
7	7
6	6
3	5
5	5
7	7
9	4

Total no.of digits tested	Recognized digits	Unrecognized Digits
101	86	15





BIBLIOGRAPHY

The project was developed using the following reference links:

- https://smalldata.tech/blog/2016/05/03/building-a-simple-neural-net-in-java
- http://stevenmiller888.github.io/mind-how-to-build-a-neural-network-part-2/
- https://natureofcode.com/book/chapter-10-neural-networks/
- https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0
- http://cs231n.github.io/neural-networks-1/
- https://www.youtube.com/watch?v=XJ7HLz9VYz0&list=PLRqwX-V7Uu6Y7MdSCaIfsxc561QI0U0Tb
- https://www.youtube.com/watch?v=2sevic5Vy4E