**1. Introduction**

Machine Learning drives much of the technology we interact with nowadays, with applications in everything from search results on Google to ETA prediction on the road to tumor diagnosis.

Handwriting recognition is the ability of a computer or device to take as input handwriting from source such as printed physical documents, pictures and other devices. It also can use handwriting as a direct input to a touch screen and then interpret it as text. There are many devices now can take handwriting as an input such as smartphones, tablets and PDA to a touch screen through a stylus or finger. This is useful as it allows the user to quickly write down number and text to the devices.

Building an effective methodology to detect hand-written characters from images with less error rate is the great task. Our aim is to make such an algorithm that will be able to generate error free recognition of hand written text from the given input image which will be a hand written character, and will help in document digitizing. In this paper we will use Neural Network classification algorithm to recognize the handwriting.

The dataset we shall use for training and testing the algorithm is from University of California at Irvine's Machine Learning Repository.

**2. Problem Definition and Algorithm**

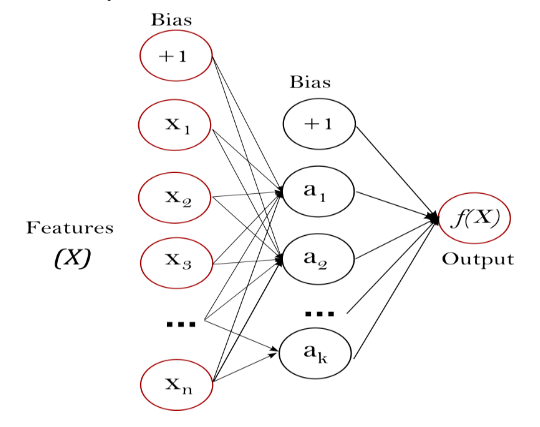
**2.1 Task Definition**

Our task is to design and implement a neural network algorithm to analyze a training and test character recognition dataset downloaded from University of California at Irvine's Machine Learning Repository. We shall then evaluate the accuracy and estimated error of the output results.

**2.2 Algorithm Definition**

There are several neural network algorithms that can be used to train this dataset with each having it unique strength and weakness over others, but we have chosen Multi-layer Perceptron (MLP) Neural Network Algorithm, MLP is a supervised algorithm that learns a function.

The function will learn by training on a dataset, with several number of dimensions for input and output. i.e., Given a set of features X = x1, x2, ...., Xm and a target y, it can learn a non-linear function approximator for either classification or regression. It is different from logistic regression, in that between the input and the output layer, there can be one or more non-linear layers, call hidden layers.



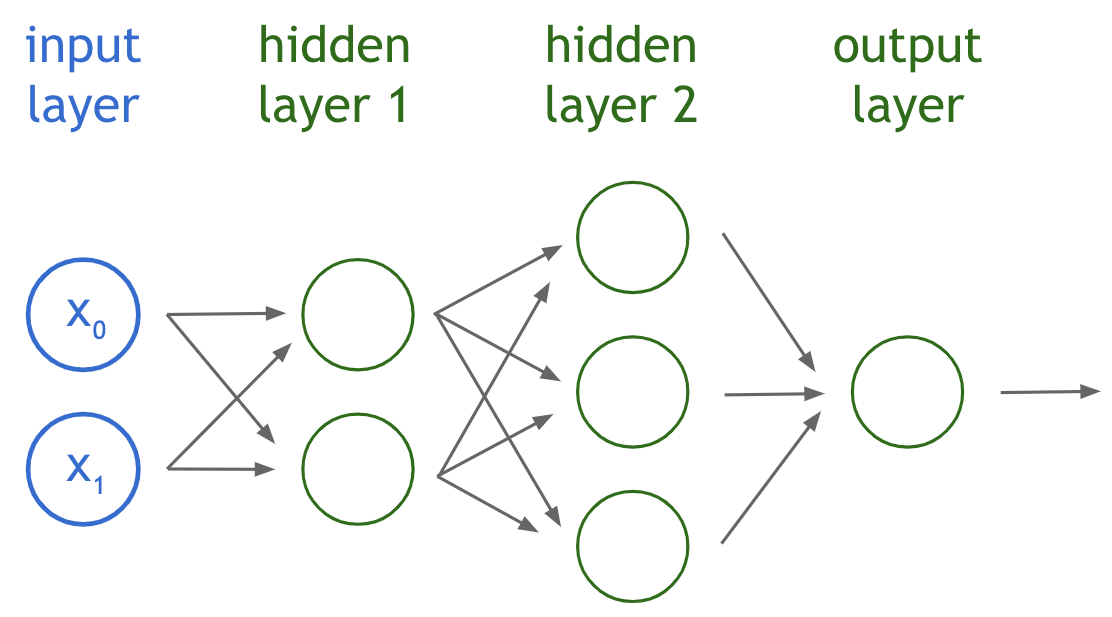


Diagram Illustrating the MLP Algorithm Toplology.

**MLP Algorithm Topology and Description.**

**Generating the test and train data**

For any supervised machine learning task, we need some data as training data to teach our program to identify the correct outputs and some data as test data to check how our program performs on inputs that it hasn’t seen before.

Both training and testing dataset is download from the University of California at Irvine's Machine Learning Repository. It's the Optical Recognition of Handwritten Digits Data Set. The training set is named cw2DataSet1 and the test set is named cw2DataSet2 This gives you two data sets, training set and a test set.

The size of learning rate, input layer, hidden layers, size of weight and the batch size of neurons use in implementing the algorithms

LEARNING\_RATE = 0.1;

BIAS\_RANGE\_SMALLEST = -0.5; BIAS\_RANGE\_BIGGEST = 0.7;

WEIGHTS\_RANGE\_SMALLEST = -1; WEIGHTS\_RANGE\_BIGGEST = 1;

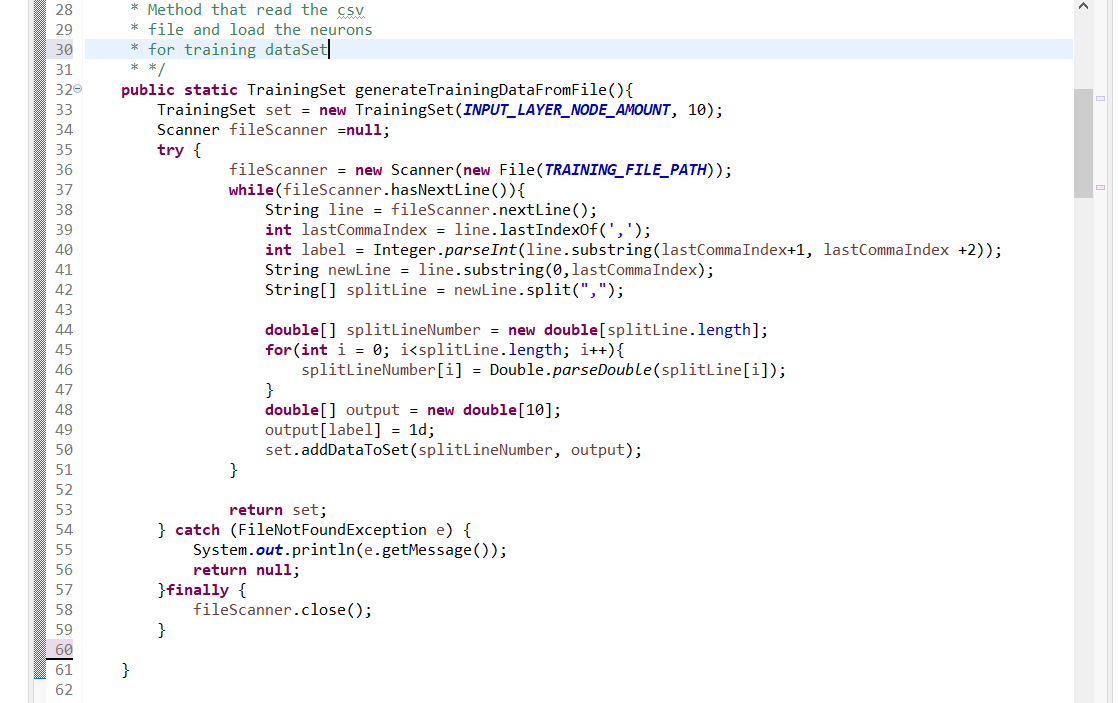
TRAINING\_EPOCHS\_VALUE = 500; TRAINING\_LOOPS\_VALUE = 500;

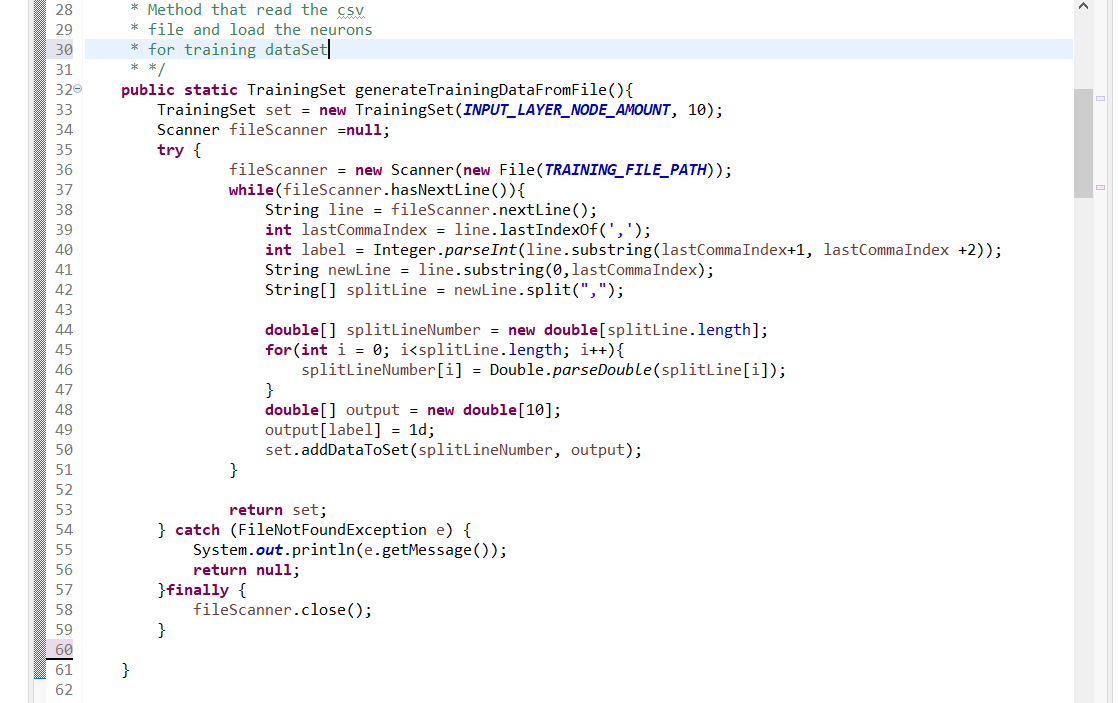
TRAINING\_BATCH\_SIZE = 32;

FIRST\_HIDDEN\_LAYER\_NODE\_AMOUNT = 26; SECOND\_HIDDEN\_LAYER\_NODE\_AMOUNT = 15;

INPUT\_LAYER\_NODE\_AMOUNT = 64; // since batch size is 32 and we have 2 hidden layers





The MLP neural network topology consist of four (4) different layers which are the input layer, two hidden layers and an output layer. Given a network size of four.

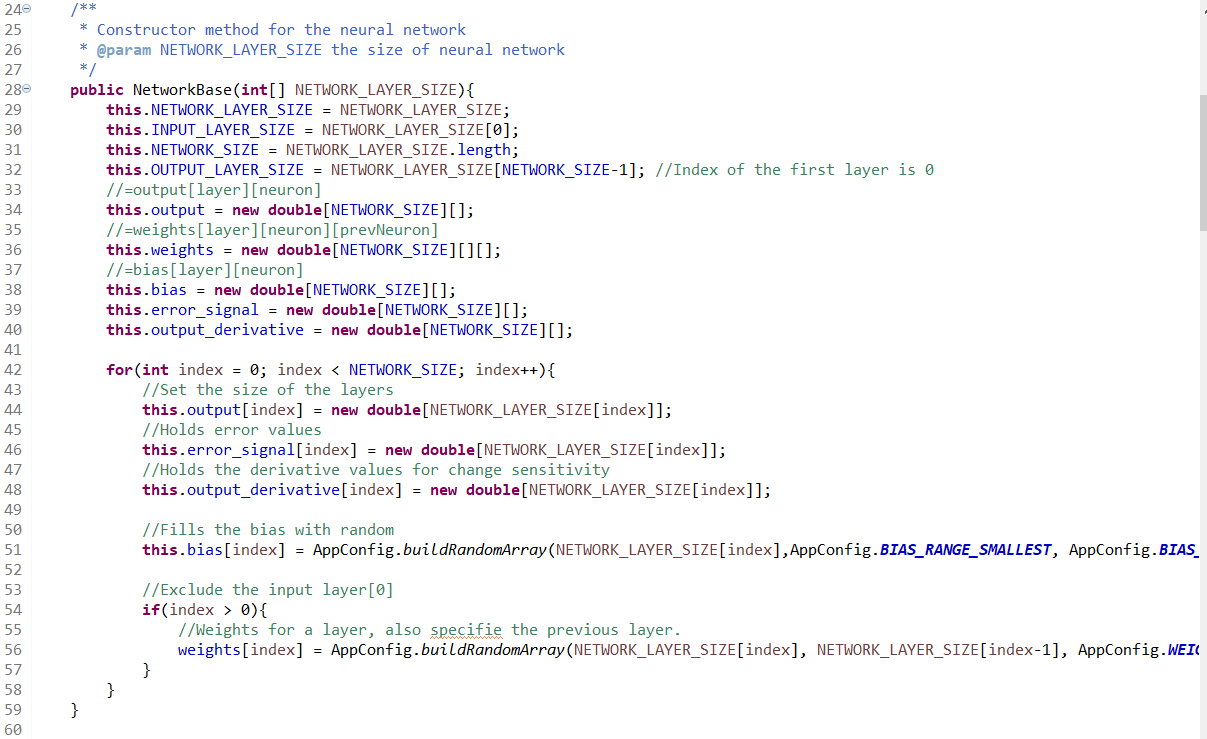
The actual layers the neurons are expected to pass are the two hidden layers in the network topology. Creating the network topology, we initiate the weight, derivative, bias (value of hidden layers), and error constant as a multi-dimensional array.

Weights represent scalar multiplications. Their job is to assess the importance of each input, as well as directionality. We have generated random values for the weight required to train the network.

Where weight contain the current layer, neurons of current layer and previous neurons of the previous layer**. = weights[layer][neuron][prevNeuron]**

Every hidden layer also is defined in a multi-dimensional array with layer label and neuron. **= bias[layer][neuron]**

We also define the final output layer as = **output[layer][neuron]**



**Feed Forward / Transfer Function**

The transfer function is different from the other components in that it takes multiple inputs. The job of the transfer function is to combine multiple inputs into one output value so that the activation function can be applied. This is usually done with a simple summation of all the inputs to the transfer function.

On its own, this scalar value is supposed to represent some information about the soil content. This value has already factored in the importance of each measurement, using the weights.

; where N= neurons and W = weight

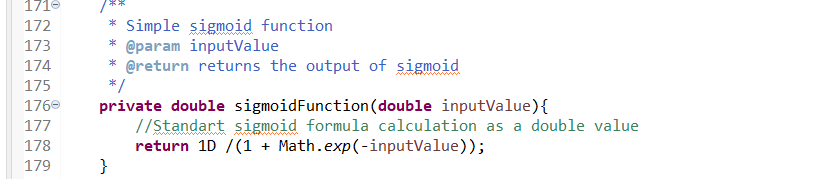
Now it is a single value that we can actually use.



Before the value is sent out of the perceptron as the final output, however, it is transformed using an activation function called the sigmoid function which is called at line 92.

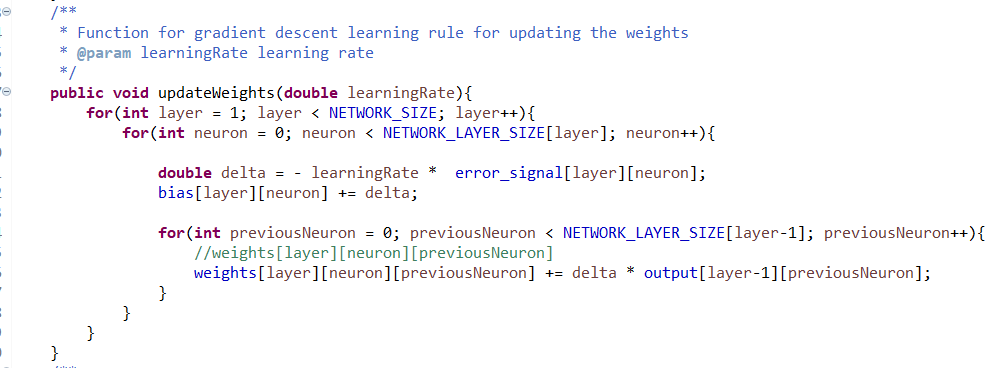
Sigmoid function is among the several activation function that can be use in machine learning to estimate the next input neuron to the next layer. We choose the sigmoid function over other activation because It’s easy to work with and has all the nice properties of activation functions: it’s non-linear, continuously differentiable, monotonic, and has a fixed output range.

In this algorithm the Sigmoid function takes a real value (Which is the accumulated sum of estimate neuron weight computation) as input and then outputs another value between 0 and 1.

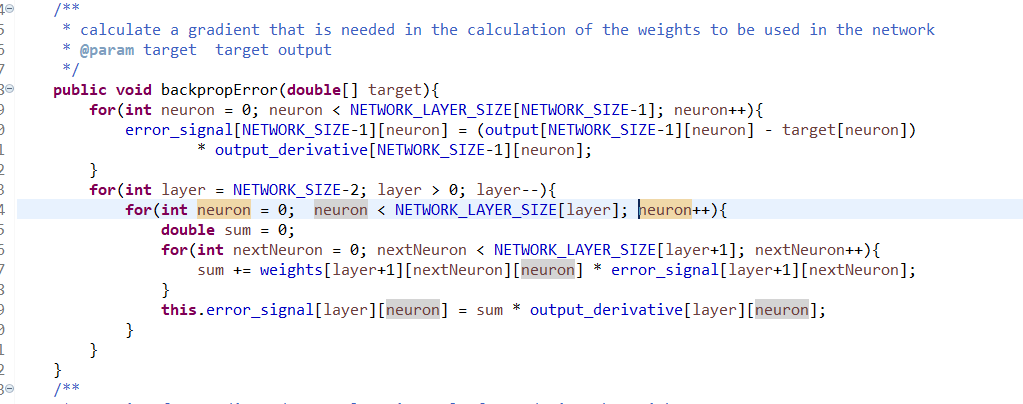


We have a learning function that define the rule for adjust the weight in every network layer, we are using a learning rate of 0.1

**public** **static** **final** **double** ***LEARNING\_RATE*** = 0.1;

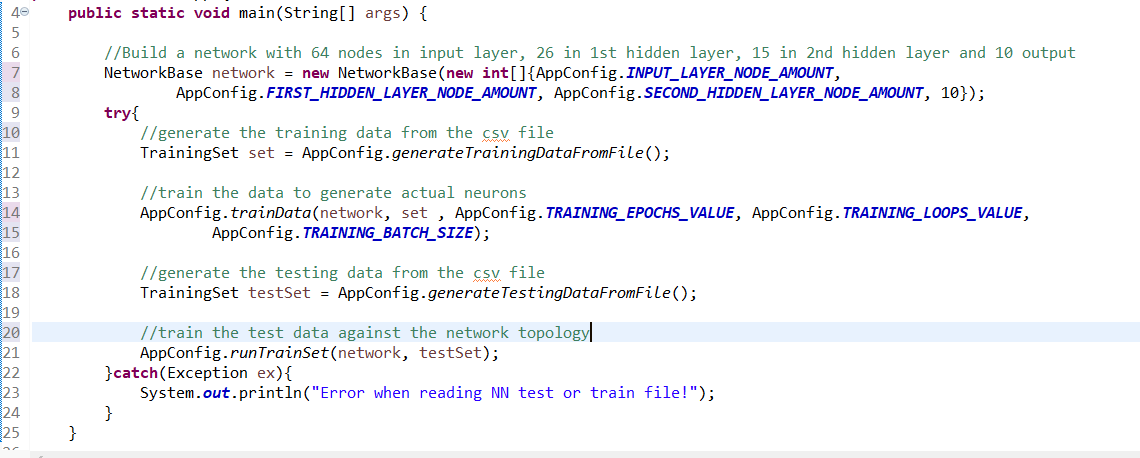


We also define the back propagation of error to calculates the gradient of the error function with respect to the neural network's weights. We first calculate the gradient of the final layer of weights and then finally calculate the gradient of the first layer of weights and we then use the partial computations of the gradient from one layer in the computation of the gradient for the previous layer.



**Training the Data with the Algorithm**

1. We first create the network topology for training the data.
2. We load the csv file containing the training dataset and fill into train dataset as input neurons
3. We then train the data in (2) to become the actual neurons required for the network
4. We then load the csv file containing the test dataset and fill into test dataset as neurons
5. We then train our test data against the network.

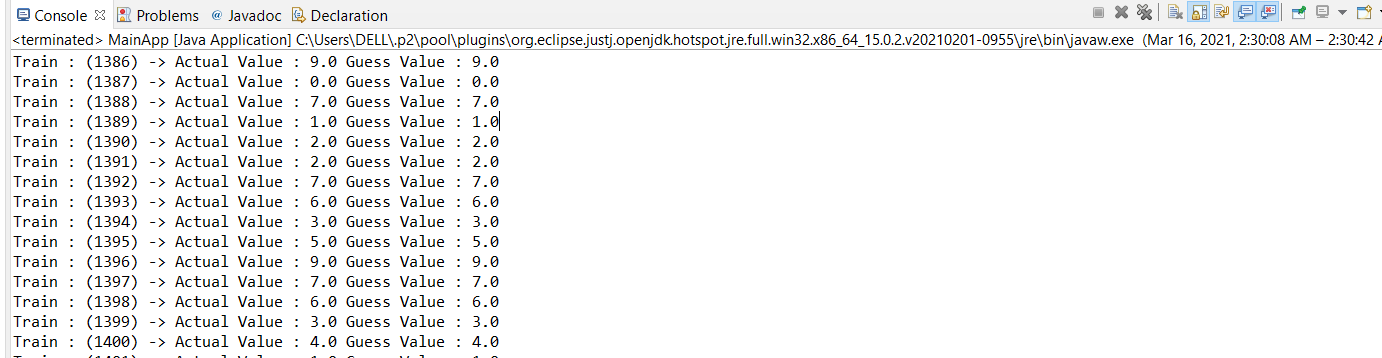


**Algorithm / Training Out Put**

**Output of training dataset**



**Output test dataset training**



**Final Result of Train Dataset Using the MLP Algorithm**

