NAME: LINDA
SURNAME: MAFUNU
STUDENT NUMBER: MFNLIN003
COURSE : CSC3022F

ASSIGNMENT: ARTIFICIAL NEURAL NETWORKS

INTRODUCTION

Artificial intelligence uses Deep Leaning and Machine Learning approach to solve problems. Deep learning is a technique used to make predictions using data that is heavily depended on neural networks. Artificial Neural Network algorithms can make predictions by taking in input data, making a prediction, comparing the prediction to the desired output and adjusting it's internal state to predict correctly the next time. Process of neural networks start with some random **weights** and **bias** vectors, make a prediction, compare it to the desired output, and adjust the vectors to predict more accurately the next time. The process continues until the difference between the prediction and the correct targets is minimal. The assignment requires to solve the XOR problem using Percentrons and second Task is to design an Artificial Neural Network that can classify handwritten digits from a data set.

Task One : XOR PROBLEM

First initialise the 3 logic gates to be used in the training of a ***XOR*** Gate and their biases. The And gate which takes in 2 inputs, ***OR*** gate takes in 2 inputs and ***NOT*** gate that takes in one input. The expected behaviour of an ***AND*** Gate is that it should produce a high output when both inputs are high, The expected behaviour of and ***OR*** gate it should produce high output when either or the inputs are height while a ***NOT*** gate reverses the input.

First initialise the training sets that will be used to train the logic gates. The training sets for the **NOT** gate is different from that of the **OR and AND** gate since they take in 2 inputs whilst it takes 1. Initialise the training labels array which is the expected output for each of the 3 logic gates. Set the temporary variables that will be used to store the randomly generated training sets and expected output. The ***AND, OR*** and ***NOT*** Gates are all trained so they produce expected outputs. The training data used to train the percentron is generated randomly but the data has to either be centralized 1 or zero. The chosen range was -0.2 to 0.2 for data close to zero and 0.8 to 1.2 for data close to 1. The range was chosen because it make the gates produce the highest level of accuracy compare to the other ranges or just the random generator. If the random generator was used the ***OR*** gate would not be trained probably meaning the output would not match the expected output. How ever for the ***NOT*** gate the random numbers were used to train it . After they are trained they are tested if they have been trained well by calling the ***Gate()*** method passing each percentron gate as input. The output is observed if it is the expected output by calling the ***Perceptron.activate()***

method which returns the output for the input supplied for each Gate. If they all work move on to solving the **XOR** problem

For higher learning rates greater than 0.6 they reduced the accuracy such that the gates would not train to 98% accuracy

<u>Percentron Network:</u>
This is all done in the ***XOR()*** method which takes in user input for XOR gate input.
The expected behaviour of the **XOR** gate is that it produces a high when the one input is high and the other is low. We can observe that the :

$$XOR(x1, x2) = ¿(NOT(¿(x1, x2)), \vee(x1, x2))$$

Biases parameters are :

bAND= -1.0, bNOT=0.5 , bOR= -1.0

<u>Step1:</u> For corresponding weight vector $w:(w1, w2)$ of the input vector $x:(x1, x2)$ to the AND and OR gate , the associated Perceptron Function can be defined as :

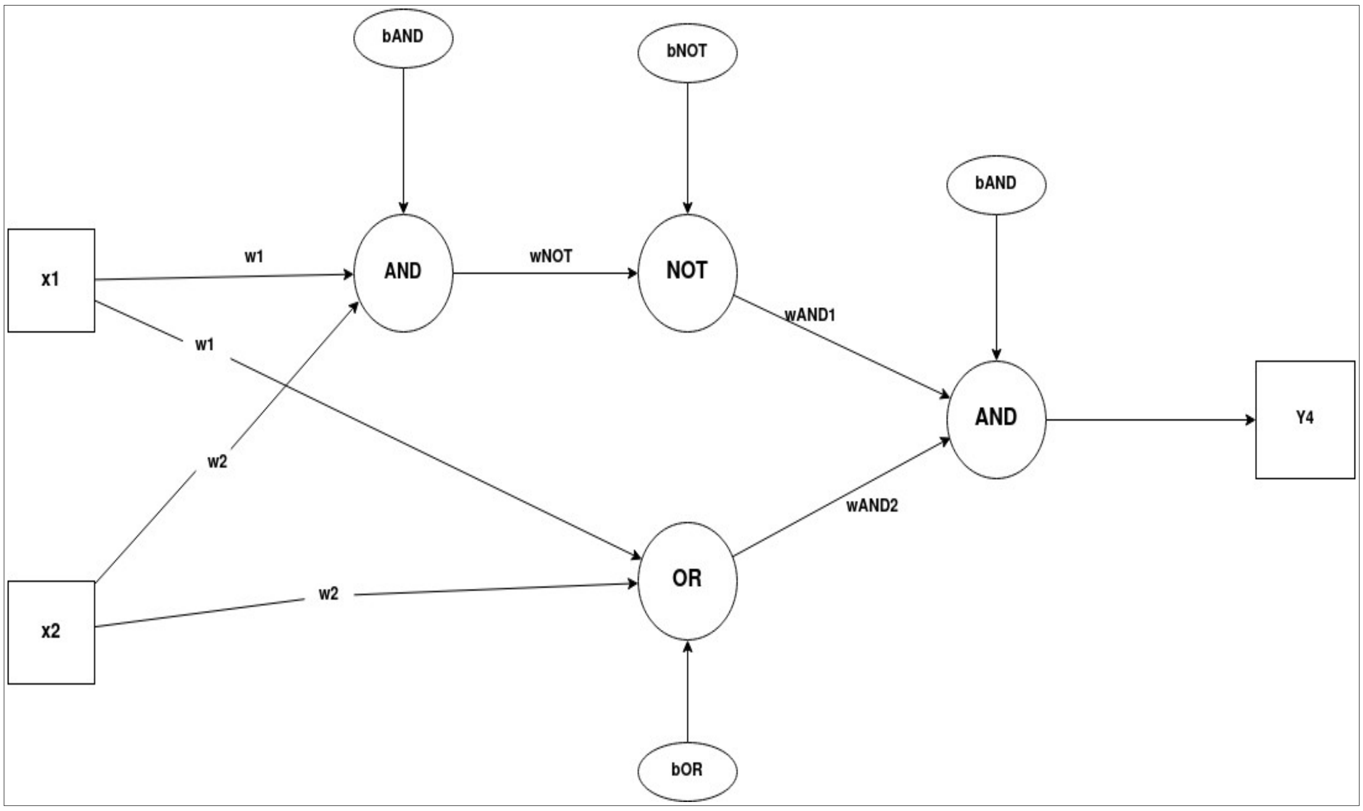
$$Y1 = \Theta(w1 * x1 + w2 * x2 + bAND)$$
$$Y2 = \Theta(w1 * x1 + w2 * x2 = bOR)$$

<u>Step2:</u> The output Y1 from **AND** Gate will be inputed to the **NOT** gate with weight **w_{NOT}** and the associated Perceptron Function can be defined as :

$$Y3 = \Theta(wNOT * Y1 + bNOT)$$

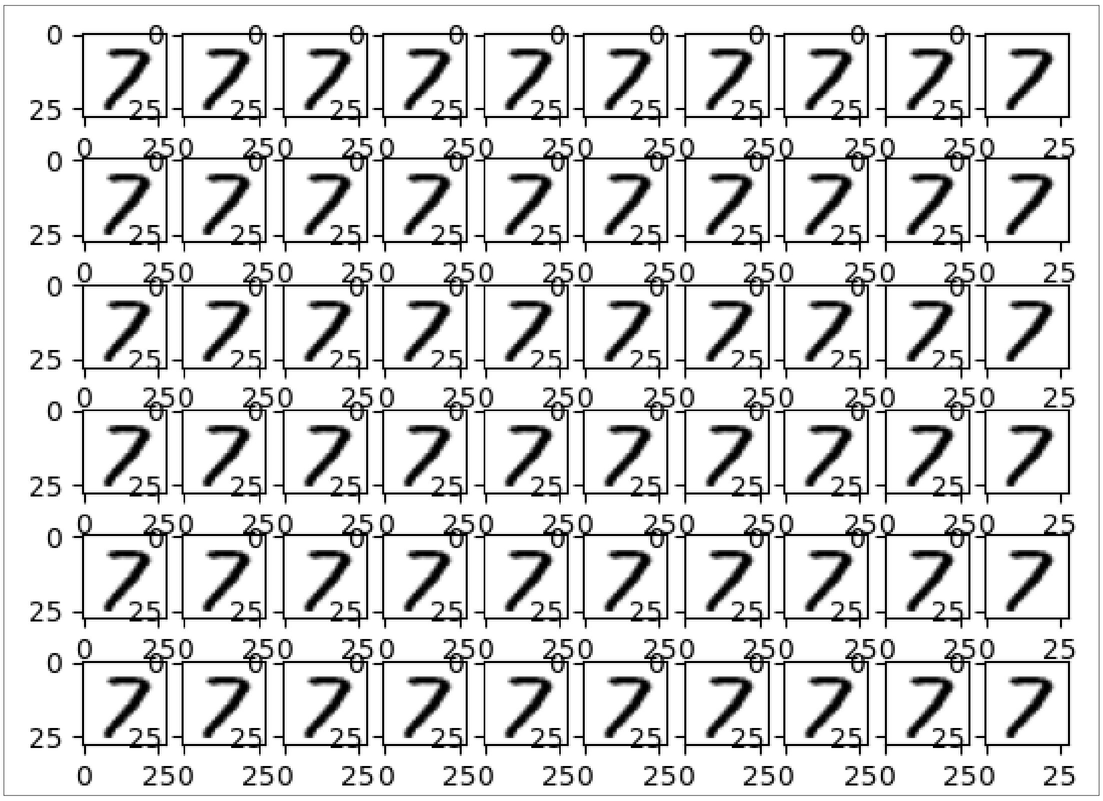
<u>Step3:</u> The output **Y2** from the **OR** gate and the output **Y3** from **NOT** gate as mentioned in step two will be inputed to the **AND** Gate with weight $(wAND1, wAND2)$. The corresponding **Y4** is the final output of the **XOR** logic function. The associated Perceptron Function can be defined as:

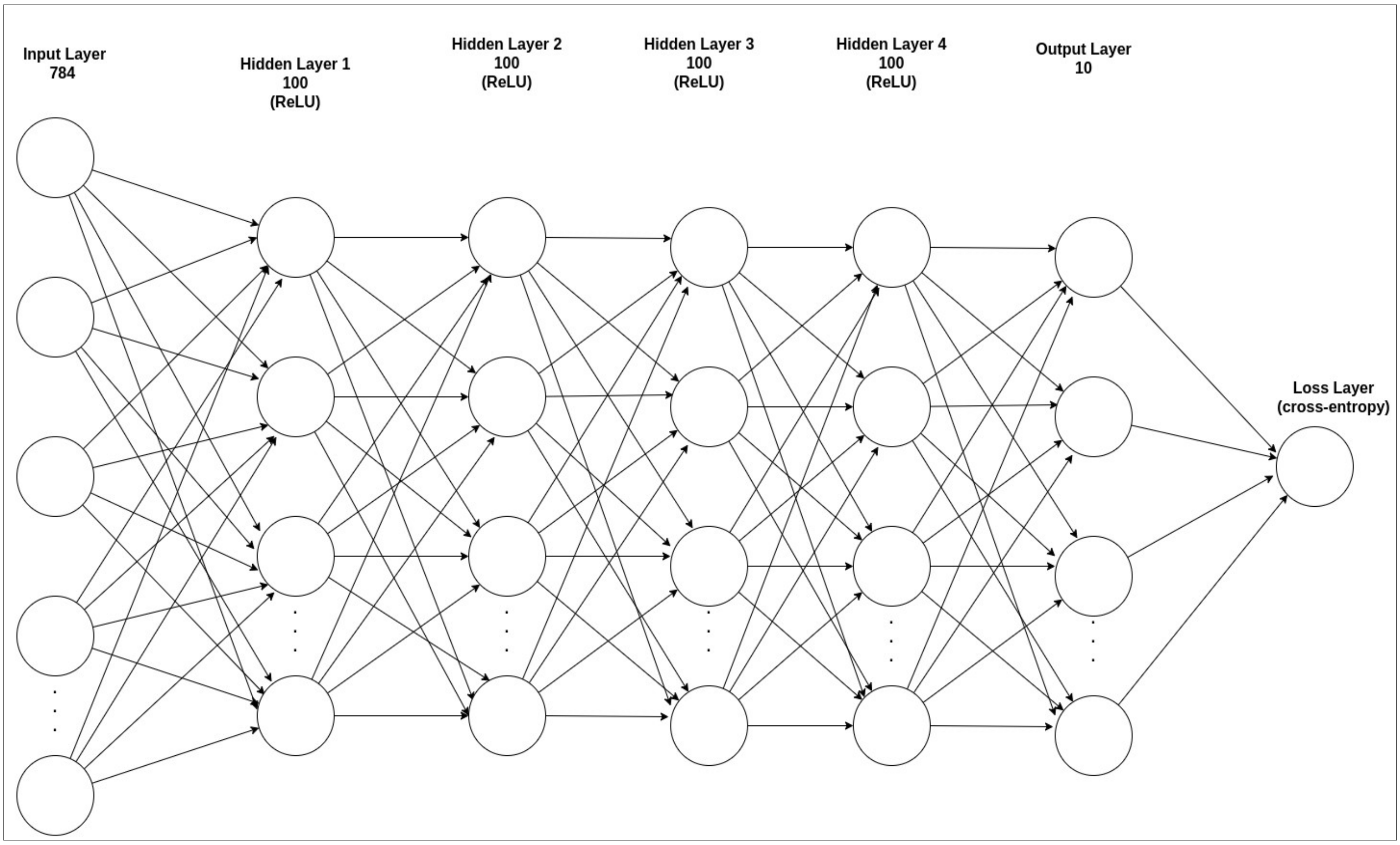
$$Y4 = \Theta(wAND1 * Y3 + wAND2 * Y2 + bAND)$$

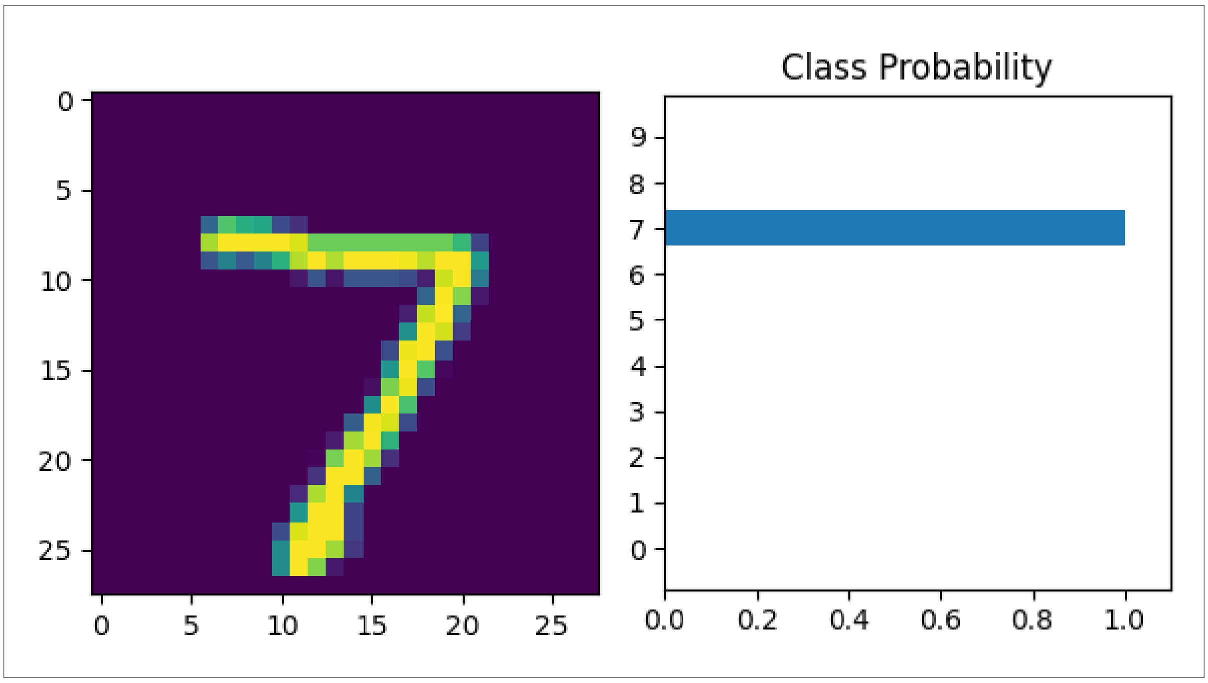


Task2 Handwriting Recognition :

Data set sample, Topology used and output from trained sample :







device which is used to put all data and model into a ***GPU*** to train both them inside it was declared. ***GPU*** is useful when training huge amount of data, it makes the training faster and you get the reserves faster. The hyper parameters are declared which can be manipulated for better performance i.e. leaning rate, number of epochs and batch size.

MNIST database is a collection of handwritten digits split into training and test set images respectively. The datasets from the database was used in the training of the neural network. The data sets were shuffled and transformed each of them. The data sets were loaded to a Data Loader which combines the data set and sampler and provides single or multi-process iterators over the data set. b***atch_size*** is the number of images the user wants to read in one go.

In this phase, some exploratory data analysis on images and tensors was done. The shape of the images and the labels were checked. The ***torch.Size([64, 1, 28, 28]) torch.Size([64])*** there are 64 images,64 labels in each batch and each image has a dimension of 28 x 28 pixels . Images were displayed in a random order. The train data images was viewed to see what the it look like

Neural Network contains a input layer which is the first layer, an output layer of ten neurons and 4 hidden layers. 4 hidden layers were used to implement deep learning which helps in reducing losses few layers results in shallow neural network. Torch .nn module allows the building of the above network very simply and it is easy to understand. The nn.Sequential() wraps the layers in the network. There are four **linear layers** with **ReLU activation which is a** a simple function which allows positive values to pass through, whereas negative values are modified to zero. The output layer is a linear layer with LogSoftmax _activation because this is a classification problem.

Technically, a ***LogSoftmax*** function is the logarithm of a ***Softmax f***unction as the name says and it looks like this, as shown below:

$$LogSoftmax(xi) = \log\left(\frac{\exp(xi)}{\Sigma\, j \exp(xi)}\right)$$

The loss is defined using the cross Entropy loss because when it is multi class classification but if it was binary class use Binary Class entropy loss. The model was loaded inside the device so that it could be trained. A neural network learns by iterating multiple times over the available data. *learn* refers to the adjustment of weights of the network to minimize the loss. Neural network iterates over the training set and updates the weights. optimize the model, perform gradient descent and update the weights by back-propagation. Each ***ep*** is the number of times you iterate over the training set see the a gradual decrease in training loss.

A function *viewClassification ()* to show the image and class probabilities that were predicted. The image to the trained model from test data set was passed to see how the model works. Iterated through the test set and calculate the total number of correct predictions. This is how we can calculate the accuracy.

Conclusion:

All the logic gates used had an accuracy close to 0.98 when trained to prove that they have been trained properly and they did produce expected behaviour. AND Gate- 97% , OR gate 85%, NOT gate -90% As for the handwriting recognition the mode had an accuracy of 97.38% which shows that the data was properly trained .