ELL784 Machine Learning Assignment 1

Kashish Srivastava (2024AIB2289)

21st August , 2024

1. Generating Boolean Functions for 'n' inputs where n = [1,2,3,4]

The number of inputs, n, can be used to build Boolean functions. There are two 2^n possible input combinations for every n inputs, and the Boolean function's result can be either 0 or 1. Thus, 2^(2^n) is the total number of Boolean functions for n inputs.

For n = 1 :
Input combinations will be [0,1]
Possible Boolean functions 2^(2^1) = 4
Boolean Functions will be :

F(0,1) = (0,0)
F(0,1) = (0,1)
F(0,1) = (1,0)
F(0,1) = (1,1)

For n = 2:
Input combinations will be [00,01,10,11]
Possible Boolean functions 2^(2^2) = 16
Boolean Functions will be :

F(00,01,10,11) = (0,0,0,0)
F(00,01,10,11) = (0,0,0,1)
.
.
.
F(00,01,10,11) = (1,1,1,1)

For n = 3:
Input combinations will be [000,001,010,011, 100,101,110,111]
Possible Boolean functions 2^(2^3) = 256

For n = 4:
Input combinations will be:
[0000,0001,0010,0011, 0100,0101,0110,0111, 1000,1001,1010,1011, 1100,1101,1110,1111]
Possible Boolean functions 2^(2^4) = 65536

All functions were generated using python script 'generateboolean.py' and saved in csv files with columns Xi in range i =1 to n (i corresponds to input) and output column Y.

2. Boolean functions that are learnable by a L-layer ANN for L = [1,2,3,4]

Learnability of a function by an Artificial Neural Network refers to its ability of generalising over the function i.e. learn the specific function or pattern.

Therefore, we are using a ANN with differing layers to observe how many layers are required to learn all the Boolean functions with n inputs, where n = [1,2,3,4].

Activation function used for all the layers in each model is Sigmoid activation function.

Loss function used in all models is Mean Squared Error(MSE).

ANN models used for this task:

1 Layer Model:

Consists of a single neuron which take n inputs .

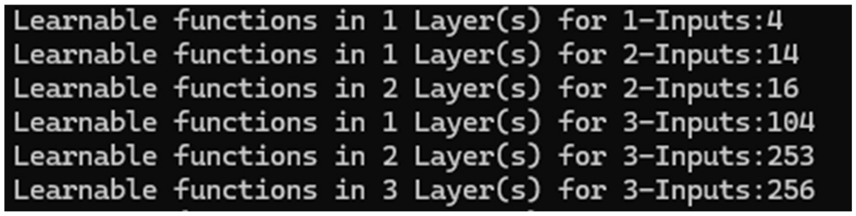
2 Layer Model:

Consists of 1 Hidden Layer with 10 neurons connected to the output layer with single neuron

With Subsequent Models(L = [3,4]) adding 1 Hidden layer with 10 neuron to the front of model

Upon training all models for 10,000 iterations over the functions, observed values are:

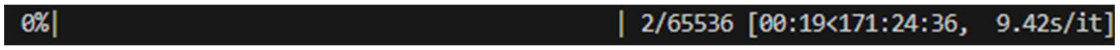
Inputs (n)	Boolean Functions	L(M,d)	Learnable functions by L = 1	L = 2	L = 3	L = 4
1	4	4	4	4	4	4
2	16	14	4	16	16	16
3	256	104	104	253	256	256
4	65536	1882	1882	-	-	-



```
Learnable functions in 1 Layer(s) for 1-Inputs:4
Learnable functions in 1 Layer(s) for 2-Inputs:14
Learnable functions in 2 Layer(s) for 2-Inputs:16
Learnable functions in 1 Layer(s) for 3-Inputs:104
Learnable functions in 2 Layer(s) for 3-Inputs:253
Learnable functions in 3 Layer(s) for 3-Inputs:256
```

Results after Running L-Layer ANN on all Boolean functions

Values for n=4 were not Feasible to compute(within given time constraints) with estimated time > 171hr for a single layer model.



```
0%|                                          | 2/65536 [00:19<171:24:36,  9.42s/it]
```

For 4-Input Boolean functions

Results were observed using learnable_functions.py script

3. Implementation of backpropagation algorithm for L – layer ANN

ANN can be made to learn in 3 steps:

a. Feed Input: Data flows from layer to layer. Retrieve output.

    Y = network(X,w)

b. Calculate the error:

    E = ½(y* - y)^2 (MSE)

c. Adjust the parameters using Gradient Descent.

    W = W - alpha * dE/dW

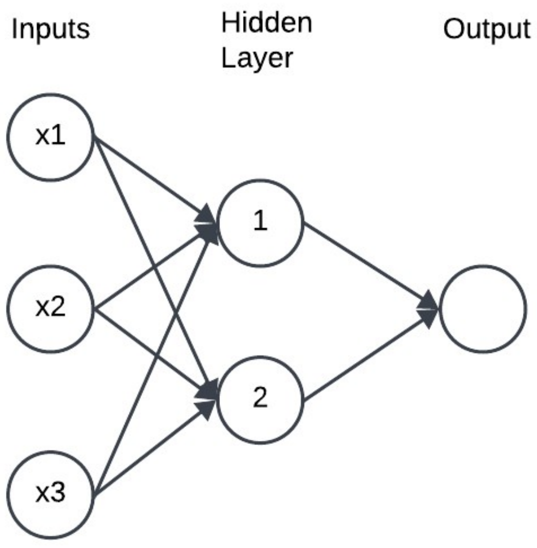
And then repeat the steps.

To make the code modular, Neural Network will be a list of Layers followed by Activation Function for the Layer till the complete architecture is defined.

```
For example NN = [Dense(3,2),
                  Sigmoid(),
                  Dense(2,1),
                  Sigmoid()]
```

Here Dense is a class which will take Number of inputs and Number of connecting Neurons. And Sigmoid Function is the Activation Function used for the layers.

The above mentioned neural network will give the following architecture.



Now, Backward propagation can be defined for dense layer using the equations:

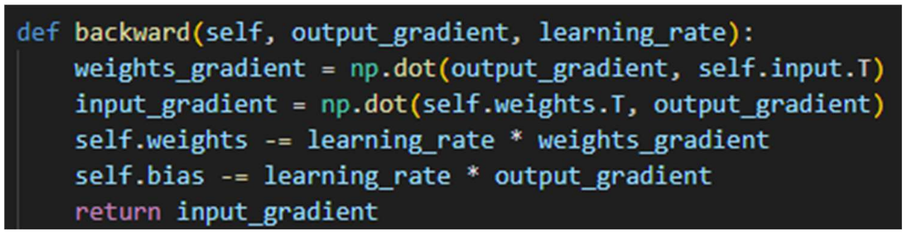
dE/dW = dE/dY . Xt

dE/dB = dE/dY (output gradient)

dE/dX = Wt . dE/dY (derivative of the error with respect to input)

where ,

E = loss functions, W = Weights matrix, X = Inputs, Y = output, t = transpose, B = Bias



```python
def backward(self, output_gradient, learning_rate):
    weights_gradient = np.dot(output_gradient, self.input.T)
    input_gradient = np.dot(self.weights.T, output_gradient)
    self.weights -= learning_rate * weights_gradient
    self.bias -= learning_rate * output_gradient
    return input_gradient
```

Snippet from implement of Dense Layer

Backward propagation for activation function layer(eg Sigmoid) can be defined by:

dE/dX = dE/dY . f'(X)

And for Loss function:

E = Mean((y*-yi)^2)

dE/dY = 2/n (Y – Y*)

Finally, the train() function was defined that takes Neural Network , loss function , Input data, Epoch and learning rate as input and returns a timeseries list of error , accuracy for each epoch.

For each epoch it iterates over the input dataset , calculates the error and initiates the back-propagation

4. Training ANN for classification of handwritten images in MNIST dataset

a. Dataset (MNIST):

The dataset used in this assignment is a subset of MNIST database of handwritten digits has 48,000 data points. It was further divided into training(46000) and testing set(2000).

Each data point in this dataset corresponds to 784 pixel values as columns and a Label for each row i.e. value in 0-9 which tells the digit which is represented by the row.

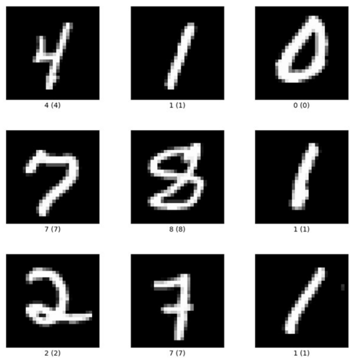


Image: Subset of dataset (9 examples)

Labels for the dataset had to be OneHotEncoded to be used with the Model.

b. Neural Network Architecture:

Neural Network used for this Dataset consists of 3 Layers

1. Hidden Layer 1 :

   Consists of 200 neurons accepting 784 Boolean inputs (pixel value)

   Activation Function: Sigmoid

2. Hidden Layer 2 :

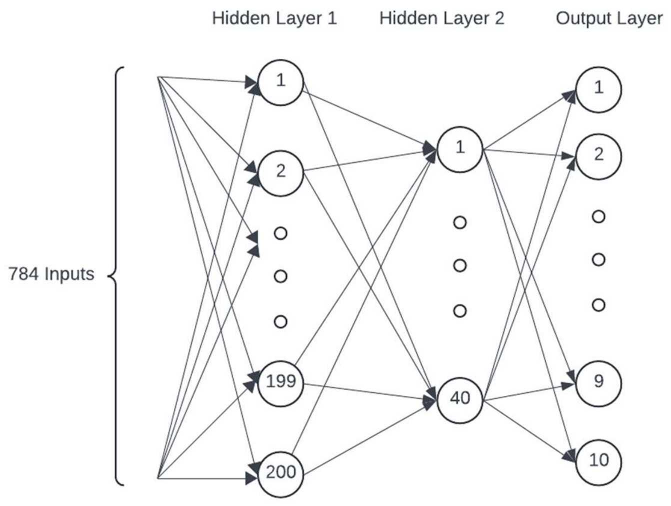
   Consists of 40 neurons accepting 200 inputs from previous hidden layer.

   Activation Function: Sigmoid

3. Output Layer :

   Consists of 10 neurons corresponding to each possible output (0-9)

   Activation Function: Softmax



Artificial Neural Network Architecture

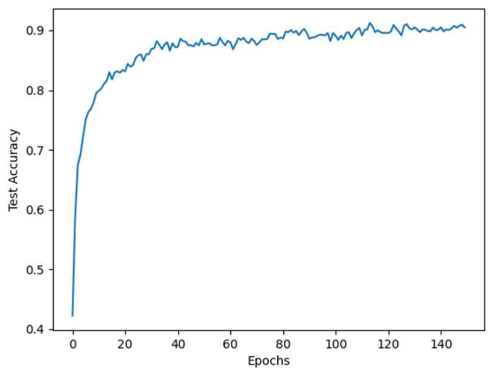
c. Training the ANN on dataset

The ANN was trained on the dataset for 150 epoch with learning rate = 0.03.

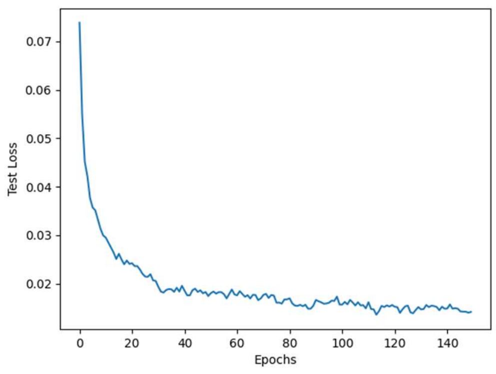
Loss function used for the same is Mean Squared Error.

Observations from the training the model are as follows:

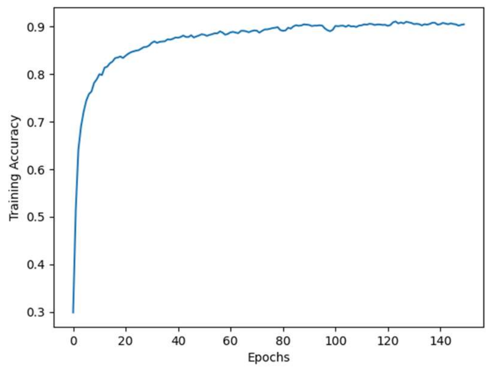
Metric	**Value**
Training Accuracy	0.911
Training Loss	0.0135
Validation Accuracy	0.9125
Validation Loss	0.014

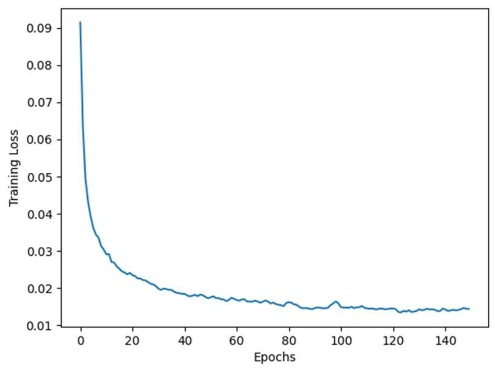


Validation Accuracy vs Epochs



Validation Loss vs Epochs





Training Accuracy vs Epochs

Training Loss vs Epochs

These results were observed by running mnist.py script