CSE 574: Introduction to Machine Learning
Programming Assignment 2

Group Members:

Name: Karan Rajendra Shah
UB Number: 50354506

Name: Pranav Moreshwar Sorte
UB Number: 50413353

Name: Thelma Melissa Gomes
UB Number: 50416894

Abstract

We have implemented code for the below problem statements –

1) Single Layer Neural Network on MNIST dataset

2) Single Layer Neural Network on CelebFaces Attributes dataset (CelebA)

3) Deep Layer Neural Network on CelebFaces Attributes dataset (CelebA)

4) Convolutional Neural Network on MNIST dataset

For (1), we have performed hyper parameter tuning on the below parameters –

a) Number of Hidden Nodes

b) Number of Iterations

c) Lambda value

For (2) and (3), we have compared the output of the Single Layer Neural network on the CelebA dataset with the deep layer neural network on the CelebA dataset. For comparison, we have compared a single layer Neural Network with 2,3,4,5,6 layers deep neural network.

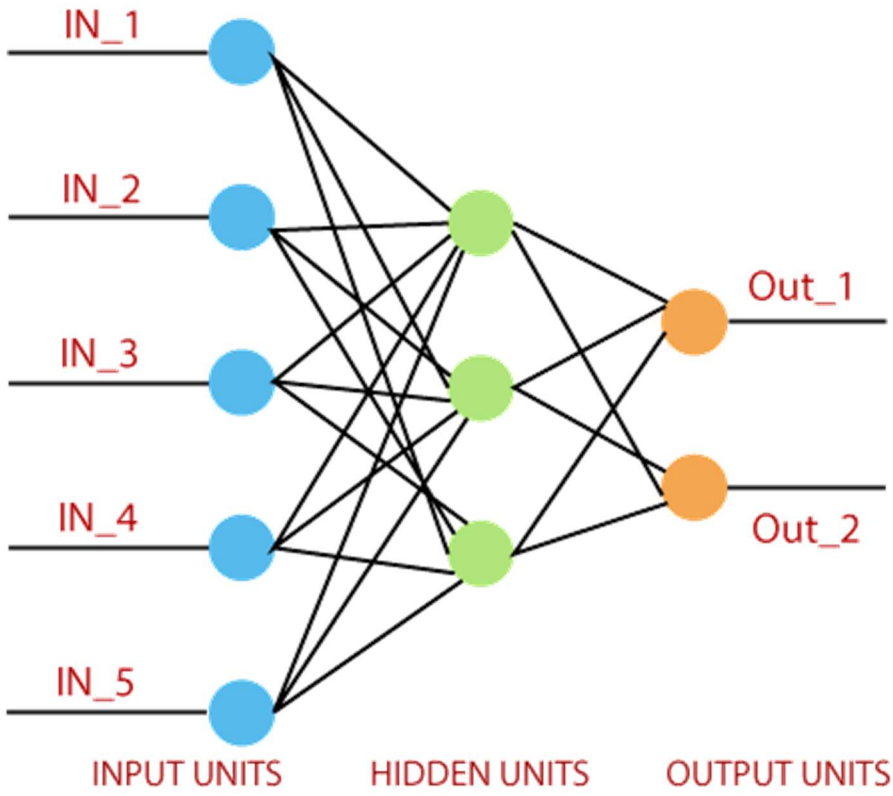
For (4), we have implemented CNN architecture on MNIST dataset.

Single Layer Neural Network on MNIST dataset

Introduction --
In this assignment, your task is to implement a Multilayer Perceptron Neural Network and evaluate its performance in classifying handwritten digits.
We have implemented a single layer neural network and implemented feedforward propagation and backward propagation.

Learning Outcomes --

- How does a Neural Network work and use Feed Forward, Back Propagation to implement Neural Network?
- How to set up a Machine Learning experiment on real data?
- How regularization plays a role in the bias-variance tradeoff?
- How to use TensorFlow library to deploy deep neural networks and understand how having multiple hidden layers can improve the performance of the neural network?
- How to use the TensorFlow library to deploy convolutional neural networks and understand the benet of convolutional neural networks compared to fully connected neural networks?



(Reference of above image -- Google Images)

A. Sigmoid Function

$$\sigma(a_j) = \frac{1}{1 + \exp(-a_j)}$$

Implemented the below equation to obtain the sigmoid function.

B. Pre-process

We have a total of 60000 training data points and 10000 testing data points. This is provided with a matlab file. We load the matlab file and store the contents of the matlab file in a dictionary. We divide the training set into 50000 train data points and 10000 validation data points.

For increasing accuracy, we apply feature selection to our data by applying the below techniques --

1. Removing common columns having values as 0.
2. Normalization of data i.e bring the values inside the data between the range 0 and 1.

From the below screenshot, we can observe the indices of columns which are of no importance and are redundant throughout the data.

Indices = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 52, 53, 54, 55, 56, 57, 82, 83, 84, 85, 111, 112, 140, 168, 476, 560, 644, 671, 672, 673, 699, 700, 701, 727, 728, 729, 730, 754, 755, 756, 757, 758, 759, 780, 781, 782, 783]

We also normalize the values of the data using the below snippet of code --

```
#         print(len(test_data))
        #Normalize Data
    train_data = train_data/255.0
    validation_data = validation_data/255.0
    test_data = test_data/255.0
    print(train_data[0])
#         print(train_data[0])
```

C) nnObjFunction

We have implemented two parts in nnObjFunction. In first part, we have implemented Feed Forward propagation in which we add bias to input node and pass the input data to the hidden node.
At the hidden node, we calculate the output using the sigmoid function and this output is sent to the output nodes which again calculate the outputs using the sigmoid function after adding the bias at hidden nodes.

In the second part, we calculate the error by using the below formula --

$$J(W^{(1)}, W^{(2)}) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{l=1}^{k}(y_{il}\ln o_{il} + (1-y_{il})\ln(1-o_{il}))$$

After calculating the error, we start back propagation. We apply partial derivative with respect to each weight vector at the hidden node and output nodes and update the weights accordingly.

The updation of weight is achieved using the minimize function provided by scipy library.

D) nnPredict

After we get the best weights, we will predict the outcome of the training data, testing data and validation data and calculate the accuracy.

Hyper-Parameter Tuning

We have three parameters which have been used in our single layer neural network architecture--

1. Lambda value (Regularization)
2. Number of Hidden Nodes
3. Number of iterations

Observations --

After running the program for multiple combinations of hyper parameter we have chosen the below optimal values

Number of Hidden Nodes	Number of iterations	Training Set Accuracy	Testing Set Accuracy	Validation Set Accuracy
50	50	95.008%	94.56%	93.92%
50	**100**	**97.558%**	**96.27%**	**96.35000000000001%**
50	150	98.11999999999999%	96.82%	96.67%
50	200	98.59%	97.17%	97.21%

100	50	95.77%	95.42%	95.32000000000001%
100	100	98.042%	96.92%	96.66%
100	150	99.056%	97.55%	97.37%
100	200	99.412%	97.78%	97.61999999999999%

We will choose the below hyper-parameters –

Number of hidden nodes – **50**

Number of iterations – **100**

It is observed that when we increase the number of iterations and the number of hidden nodes, the accuracy increases. However, we have to make sure that the hyper parameters we have chosen do not cause overfitting as we can see that for 100 hidden nodes and 200 iterations we get the training data accuracy as 99.412%, but the validation and testing accuracy have decreased significantly.

Now we choose the best lambda value for regularization purposes

Number of Hidden nodes – 50

Number of Iterations – 100

Lambda Value	Training Set Accuracy	Testing Set Accuracy	Validation Set Accuracy
0	97.39999999999999%	96.35000000000001%	96.3%
5	97.214%	96.15%	96.23%

10	**97.342%**	**96.21%**	**96.19%**
15	97.012%	96.12%	96.21%
20	96.272%	95.83%	95.52000000000001%
25	96.49600000000001%	96.04%	95.64%
30	95.828%	95.38%	95.46%
35	95.74000000000001%	95.41%	95.06%
40	94.88799999999999%	94.52000000000001%	94.06%
45	94.792%	94.54%	94.54%
50	94.358%	93.95%	93.78%
55	94.43%	94.27%	93.43%
60	93.808%	93.67%	93.66%

As we can observe from the above table, **10** is the best lambda value i.e best hyper parameter for the above dataset.

Analysis: Neural Network vs Deep Neural Network

In this section,we'll be comparing and analyzing the Single hidden layer implementation and Deep Neural Network with multiple layers options on the CelebA dataset.
As per the requirement, we'll be using the same neural network consisting of feed forward and back propagation algorithm we created for handwritten digits on this dataset too.

Below are the observations.

Performance of Neural network implementation on CelebA face dataset:

Dataset	Accuracy
Training Set	85.43%
Validation Set	84.17%
Testing Set	85.58%

Time taken for the implementation: 222 seconds.

Performance of Deep Neural Network implementation on CelebA face dataset (using Tensorflow library):

Number of Hidden Layers	Time taken (in secs)	Accuracy
2	239 s	82.85%
3	247 s	79.52%
4	272 s	75.85%
5	303 s	76.23%
6	310 s	77.25%

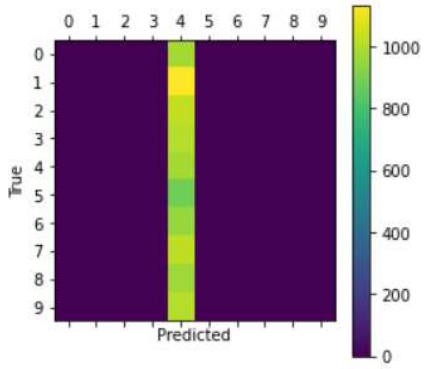
Analysis:

- From the results we obtained, we can see that the single layer neural network has more accuracy than a deep neural network.
- For deep neural networks, the accuracy of the network decreases (sometimes fluctuates slightly) as the number of hidden layers increases.
- In terms of the time taken, the single layer neural network takes less time as compared to the deep neural networks.
- The architecture of the network becomes more complex as layers increase and which could be the reason for the above observation.
- The 6 hidden layer network has less accuracy as compared to a 2 layer deep network due to a possible overfitting on the trained data on its architecture. The 2 layer network is better in generalizing on the trained data.
- Deep Neural Networks are observed to perform best with a lesser number of hidden layers.
- More the number of hidden layers, greater the execution time.

Analysis of Convolutional Neural Networks

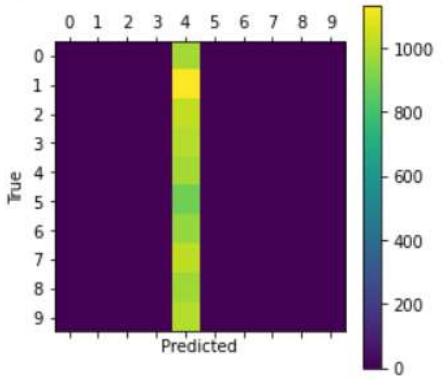
Confusion Matrix for each iterations --

```
Confusion Matrix:
[[   0    0    0    0  980    0    0    0    0    0]
 [   0    0    0    0 1135    0    0    0    0    0]
 [   0    0    0    0 1032    0    0    0    0    0]
 [   0    0    0    0 1010    0    0    0    0    0]
 [   0    0    0    0  982    0    0    0    0    0]
 [   0    0    0    0  892    0    0    0    0    0]
 [   0    0    0    0  958    0    0    0    0    0]
 [   0    0    0    0 1028    0    0    0    0    0]
 [   0    0    0    0  974    0    0    0    0    0]
 [   0    0    0    0 1009    0    0    0    0    0]]
```



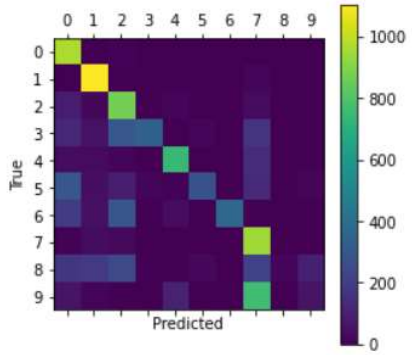
Iteration 1 -

```
Confusion Matrix:
[[   0    0    0    0  980    0    0    0    0    0]
 [   0    0    0    0 1135    0    0    0    0    0]
 [   0    0    0    0 1032    0    0    0    0    0]
 [   0    0    0    0 1010    0    0    0    0    0]
 [   0    0    0    0  982    0    0    0    0    0]
 [   0    0    0    0  892    0    0    0    0    0]
 [   0    0    0    0  958    0    0    0    0    0]
 [   0    0    0    0 1028    0    0    0    0    0]
 [   0    0    0    0  974    0    0    0    0    0]
 [   0    0    0    0 1009    0    0    0    0    0]]
```



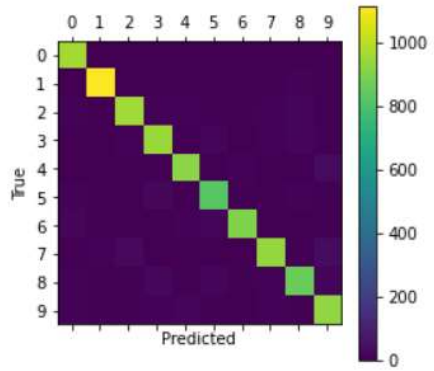
Iteration 100 --

```
Confusion Matrix:
[[ 967    1    9    0    0    0    0    3    0    0]
 [   0 1105    7    0    0    0    0   21    0    2]
 [  88   22  874    0   14    0    0   34    0    0]
 [ 122   56  302  344    1   14    0  169    0    2]
 [  37   33   16    0  748    0    1  141    0    6]
 [ 301   44   92   18    9  277    2  133    0   16]
 [ 198   51  294    0   36    4  366    8    0    1]
 [  11   38   26    0    6    0    0  947    0    0]
 [ 179  186  243    6    5   22    0  223   15   95]
 [  57   19    7    0  104    0    0  765    0   57]]
```



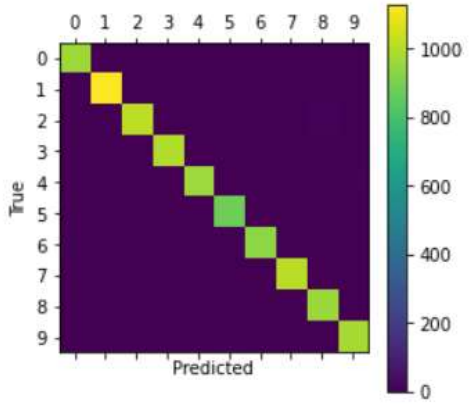
Iteration 1000 --

```
Confusion Matrix:
[[ 964    0    4    1    0    4    3    2    2    0]
 [   0 1115    4    2    0    1    3    0   10    0]
 [  12    2  960   13   10    1    5   13   14    2]
 [   2    2   11  947    0   14    0   10   16    8]
 [   1    3    7    0  923    0   10    2    2   34]
 [  10    3    2   25    4  825    7    1   10    5]
 [  14    6    5    1   10   16  904    0    2    0]
 [   2    9   28    4    4    1    0  937    5   38]
 [   9    7    5   26    7   17    8    9  866   20]
 [  10    6    6   12   18    5    0   12    7  933]]
```



Iterations 10000 --

```
Confusion Matrix:
[[ 971    0    1    1    0    3    1    0    3    0]
 [   0 1130    1    0    0    1    0    1    2    0]
 [   2    2 1017    0    0    0    0    2    9    0]
 [   1    0    0  999    0    7    0    1    2    0]
 [   1    1    2    0  966    0    1    2    3    6]
 [   1    0    1    2    0  884    1    0    1    2]
 [   5    3    0    1    2    8  938    0    1    0]
 [   0    3    5    3    0    0    0 1012    2    3]
 [   3    0    1    2    0    2    0    1  962    3]
 [   2    4    0    3    3    8    0    3    4  982]]
```



Iterations	Training Time	Test Accuracy
1	0:00:00	9.8% (982 / 10000)
100	0:00:06	57.0% (5700 / 10000)
1000	0:00:58	93.7% (9374 / 10000)
10000	0:09:49	98.6% (9861 / 10000)

Analysis --

We move towards a better accuracy if we increase the number of iterations in a CNN architecture.
When the number of iterations is 10000, we get an accuracy of 98.6%, however the time taken to train the neural networks is much more as compared to CNN with less number of iterations.

Conclusions --

Hyper-parameters for Single Layer neural network on MNIST

Lambda Value -- 10
Number of Hidden Nodes -- 50
Number of iterations -- 100