CSE574 Introduction to Machine Learning

Programming Assignment 2

Handwritten Digits Classification

Group Number 22
Nimisha Philip Rodrigues 50248016
Manasi Samir Tamboli 50246653

Aim: to implement a Multilayer Perceptron Neural Network and evaluate its performance in classifying handwritten digits .

Given Files:

- mnist_all.mat
- face_all.pickle
- nnScript.py
- facennScript.py
- deepnnScript.py
- cnnScript.py

Digit Classification using Single Layer Neural Network:

We have the given data of 60,000 out of which 50,000 we use for training and we use the rest 10,000 data points to validate the hyper parameters like lambda. The 10,000 data points is also used to train the data model in our system. The model is tested and then optimized for checking the accuracy for the hyperparameters.

Hyper Parameters vs Accuracy for nnScript:

What we observe from the below table is as the units in hidden layers is increased the learning time also increases the accuracy of the model also increases.

No of hidden units	**Accuracy**
10	90%
20	91.32%
30	93.45%
40	94.30%
50	94.56%
60	94.89%
70	95.10%
80	96.26%
90	95.56%
100	95.98%

For the table above,
Learning time for lamda=10 sec.
The highest accuracy is at 80 units after which there is slight decrease in the accuracy.

Why we choose hyper parameters for Neural Networks:

We observe from the above table that as the number of hidden node increase the prediction accuracy also increases but the fact that the learning time is also increased cannot be overlooked. That is primarly because the network becomes more complex because of more number of nodes, hence the computations increase and hence more time. Since, what is important for us right now is accuracy the incrase in time can be disregarded.This is how we arrived at highest accuracy of 80.

After reaching the highest peak at 80, there is some decrease in the accuracy.This can be attributed to the fact that, the network becomes more complex and thus the model doesn't perform well on new data.This is called as **Overfitting.**

Overfitting of data is observed more in Deep Neural Network as there are more layers with many neurons. We can avoid overfitting by **Regularization.** Regularization factor helps prevent overfitting that is mainly observed in complex models. It manages weight by modifying objective function.

We tried with different values of regularization parameters and found that non-regularized runs take more time to converge than regularized runs.What we did is we tried with different values of lambda and hidden nodes and tried to find values which gives minimal difference between Training Accuracy and Testing Accuracy.

After observing the maximum value obtained is at lambda=10 and hidden nodes=80
The accuracy for the above parameters:

Training Data	96.10%
Validation Data	95.82%
Testing Data	96.26%

Comparison between Facenn and Deepnn Models:

Facenn Model:
We run the model on photographic image data ie CalebA data for testing the accuracy of our Neural Networks.
The accuracy obtained as follows:

Training Data	:86.014%
Validation Data	:84.878%
Test Data	: 86.638%

Deep Neural Network:
The same data set that we used for facenn ie CelebA dataset we now evaluate on deep neural network to learn the accuracy, the effect of layers increased and the learning time. We use the tensorflow library here.

The table below shows our result:

Hidden Layers	Accuracy
2	81.20%
3	77.67%
5	75.10%
7	77.10%

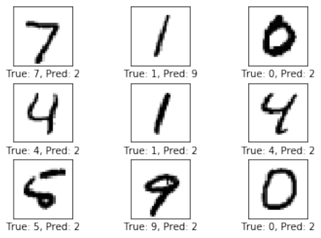
Convolutional Neural Networks:

We executed the cnnScript to make observations on Convolutional Neural Networks using a training set of size 55000, test set of size 10000 and validation set of size 5000.

Initially – Iteration 0

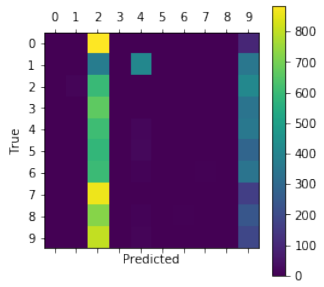
Accuracy on Test-Set: 8.1% (806 / 10000)

Example errors:



Confusion Matrix:

```
[[  0   0 886   0   0   0   0   0   0  94]
 [  0   0 372   0 409   0   0   1   0 353]
 [  0  11 602   0   3   0   1   0   0 415]
 [  0   0 666   0   2   0   0   0   0 342]
 [  0   0 611   0  14   0   2   0   0 355]
 [  0   0 586   0  18   0   0   0   0 288]
 [  0   0 605   0   7   0   0   4   0 342]
 [  0   0 862   0   3   0   0   0   0 163]
 [  0   2 725   0   7   0   4   0   0 236]
 [  0   0 806   0  12   0   1   0   0 190]]
```



Optimization Iteration: 1,

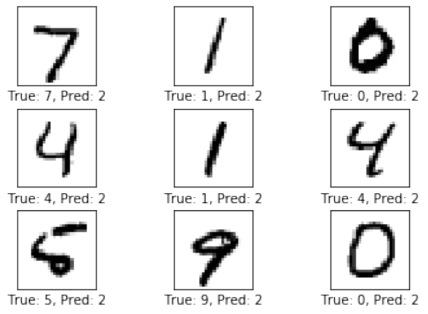
Training Accuracy: 10.9%

Time usage: 0:00:00

After 1 iteration:

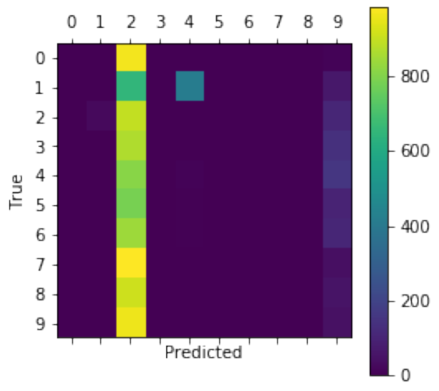
Accuracy on Test-Set: 9.5% (950 / 10000)

Example errors:



Confusion Matrix:

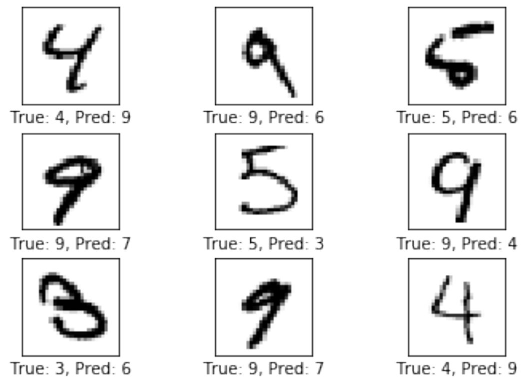
```
[[  0   0 970   0   0   0   0   0   0  10]
 [  0   1 652   0 417   0   0   0   0  65]
 [  0  26 897   0   2   0   1   0   0 106]
 [  0   1 871   0   2   0   0   0   0 136]
 [  0   0 812   0   9   0   3   0   0 158]
 [  0   0 786   0   7   0   0   0   0  99]
 [  0   1 844   0   5   0   0   0   0 108]
 [  0   0 989   0   0   0   0   0   0  39]
 [  0   3 915   0   0   0   3   0   0  53]
 [  0   0 965   0   1   0   0   0   0  43]]
```



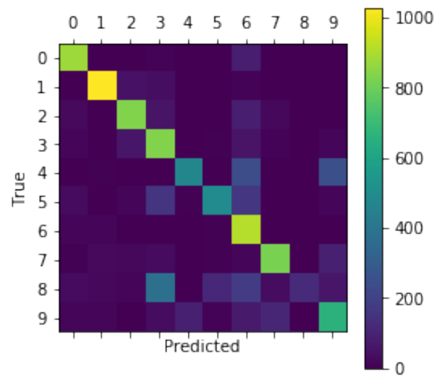
Time usage: 0:00:07

<u>After 99 Iterations:</u>

Accuracy on Test-Set: 70.8% (7076 / 10000)
Example errors:



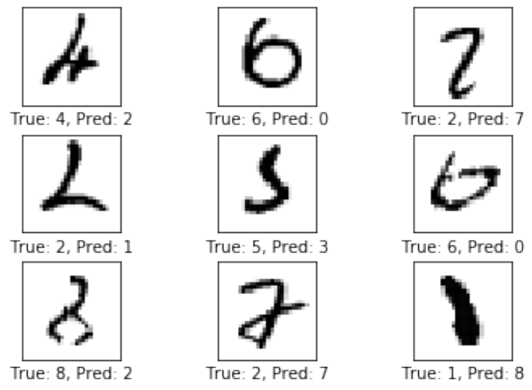
Confusion Matrix:
```
[[ 878    0    4   10    0    2   85    1    0    0]
 [   0 1030   52   39    0    0   10    0    1    3]
 [  28    2  836   53    0    0   87   23    0    3]
 [  18    4   63  836    0    8   55   10    0   16]
 [   0    5    2    2  471    2  242    4    1  253]
 [  32    2   13  156    8  497  164    1    1   18]
 [  14   13    2    1    2    7  917    0    2    0]
 [   7   26   24   32    2    7   18  823    0   89]
 [  32   21   18  379    4  113  185   33  127   62]
 [  13   13    1   33   91   10   76  111    0  661]]
```



Time usage: 0:01:07

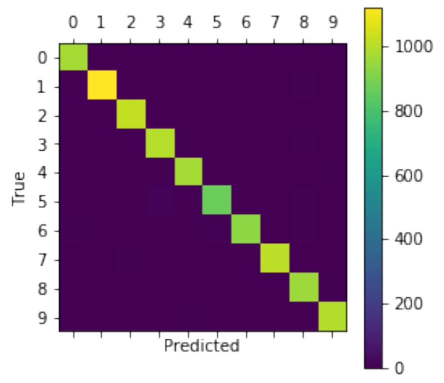
After 9000 Iterations:

Accuracy on Test-Set: 98.8% (9861 / 10000)
Example errors:



Confusion Matrix:

```
[[ 972    0    0    1    0    0    2    1    3    1]
 [    0 1124    2    0    0    0    1    1    7    0]
 [    2    3 1019    1    0    0    0    4    2    1]
 [    1    0    1 1000    0    1    0    1    5    1]
 [    0    0    1    0  974    0    0    0    0    7]
 [    2    0    0   11    0  870    1    1    5    2]
 [    7    3    0    1    3    5  934    0    5    0]
 [    0    2    6    1    0    0    0 1014    1    4]
 [    2    0    1    2    1    1    0    2  959    6]
 [    1    3    0    1    6    2    0    1    0  995]]
```



Observations:
We found the accuracy of test set by running convolutional neural network as 98.8%.The accuracy increased monotonically upto 900 iterations and then started alternating between increase and decrease. At some iterations like 2400 and 2600, the accuracy was 100%. The execution time increased with increase in the number of iterations. The total execution time was 12 minutes and 31 seconds.

Conclusion:
After executing the above experiment, we learned the following things,
How Neural Networks works and use Feed Forward and Back Propagation.
How to setup Machine Learning Experiment on real data.
The role of regularization
The use of tensorflow library