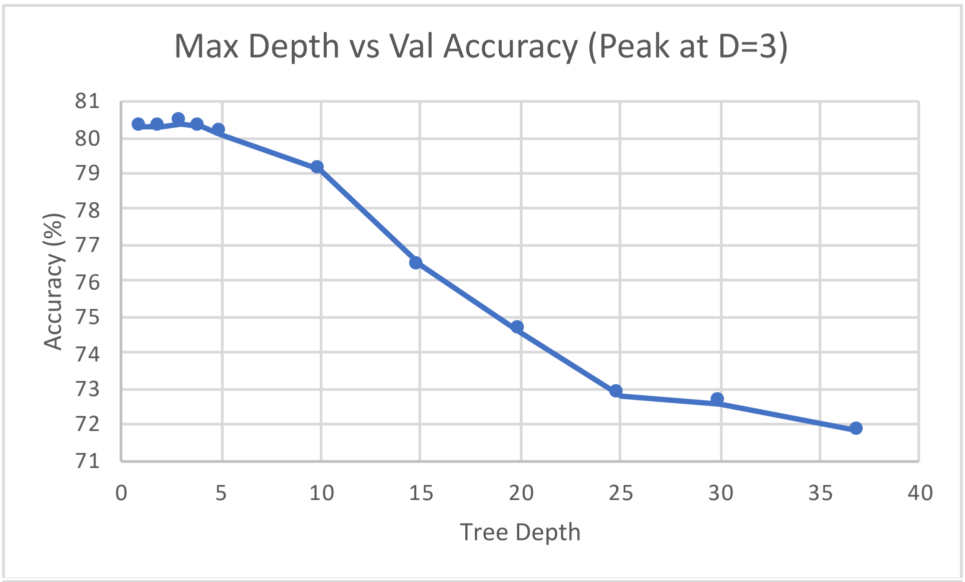
COL774 – Machine Learning

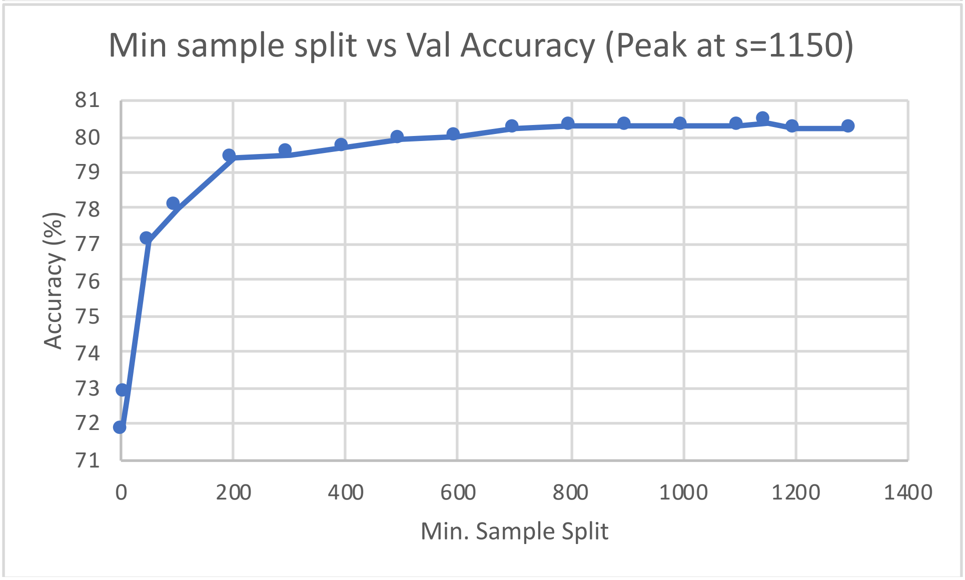
Assignment 3

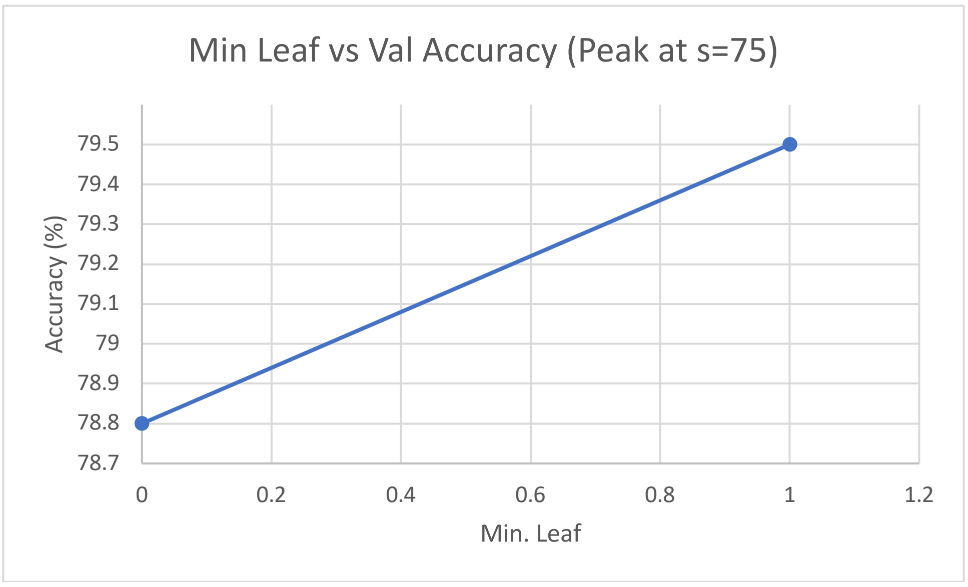
Nilaksh Agarwal
2015PH10813

Part 1 – Decision Trees:

Part (d) – Sklearn:







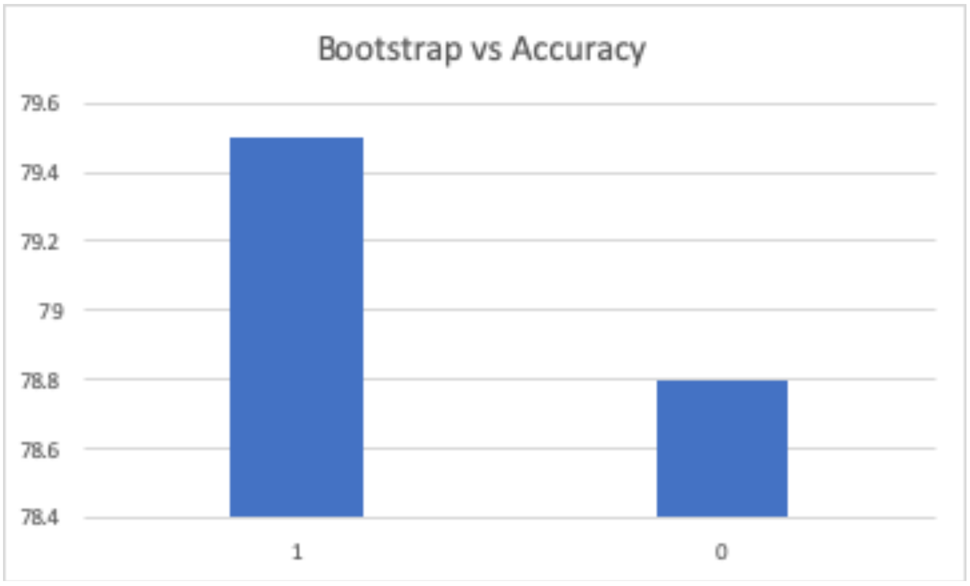
Best Parameters (Max Depth = 3, Min Split = 1150, Min Leaf = 100)
Train Set Accuracy – 83.05
Val Set Accuracy – 80.41
Test Set Accuracy – 81.13

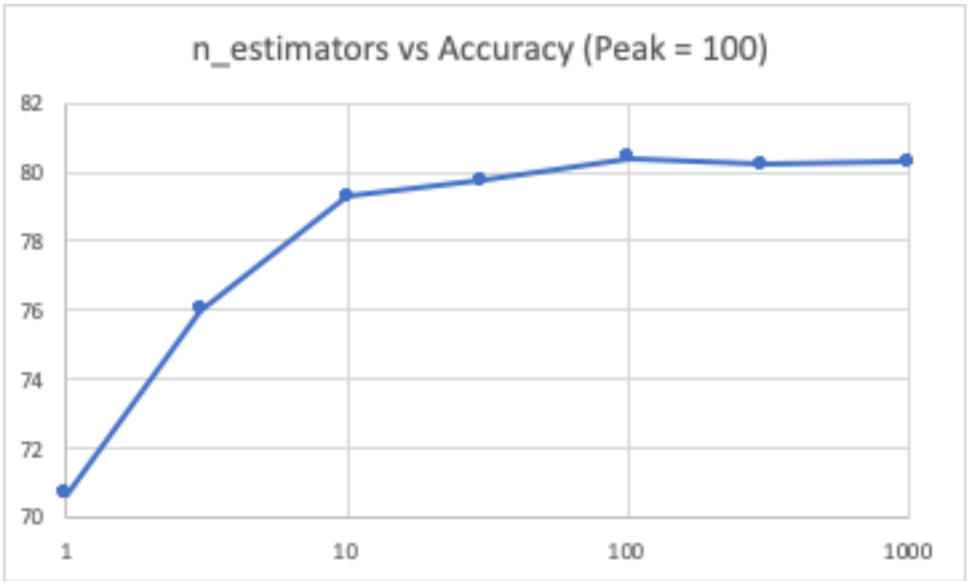
Part (e) – Categorical (Sklearn):

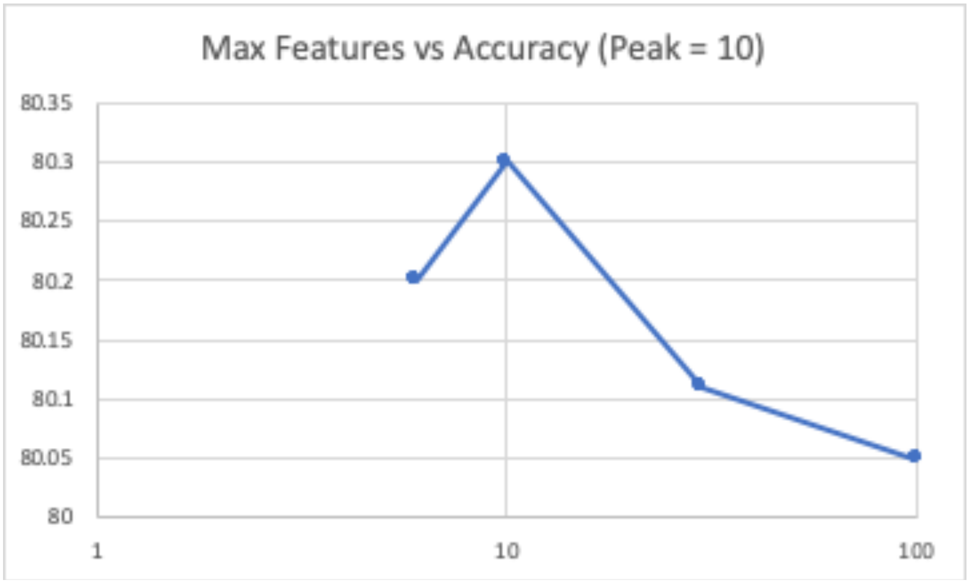
After converting all categorical variables to one-hot encoding (98 features)
Same parameters as previous part.
Train Set Accuracy – 82.9
Val Set Accuracy – 80.06
Test Set Accuracy – 80.8

This might be due to the changed samples per feature ratio. So, our min-sample split or min-sample leaf values from the previous part might not work.
We don't see any improvement in accuracy.

Part (f) – Random Forests:







Here, we obtain best accuracies for bootstrapping (since it's better able to generalise to unseen data per tree) and around 100 trees (since greater than that starts heavily overfitting) and max_featues = 10 since taking more than those again leads to overfitting as each tree gets build deeper and more specific.

Train Set Accuracy – 99.96
Val Set Accuracy – 80.15
Test Set Accuracy – 80.3

Part 2 – Neural Networks:

Part (a) – One Hot Encoding:

```
[ 1, 10,  1, 11,  1, 13,  1, 12,  1,  1] -> Converted to :
[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0
, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0,
0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

Similarly, done for the output

Test Data
Train Data
```

Part (b) – Network Architecture:

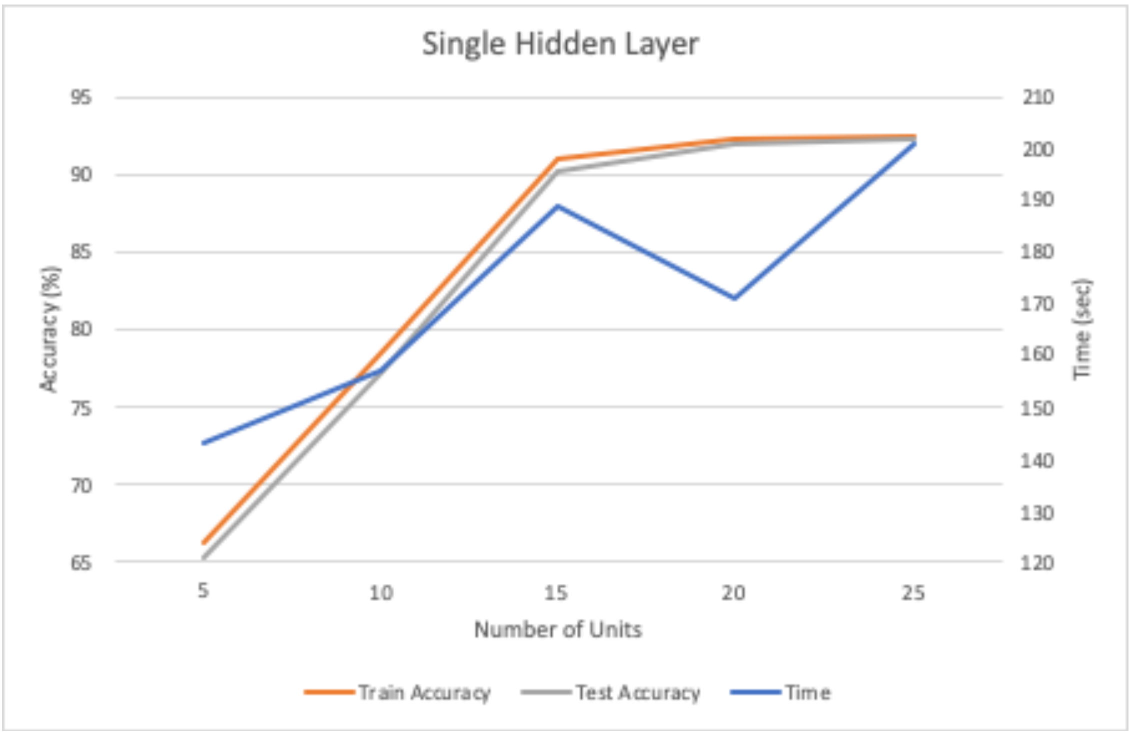
Inputs to network include:

- Size of batch for mini-batch Gradient Descent
- Number of inputs
- Number of hidden layers + size of each layer
- Number of outputs
- Non-linearity (Sigmoid or ReLU)
- Learning Rate (Fixed or variable)

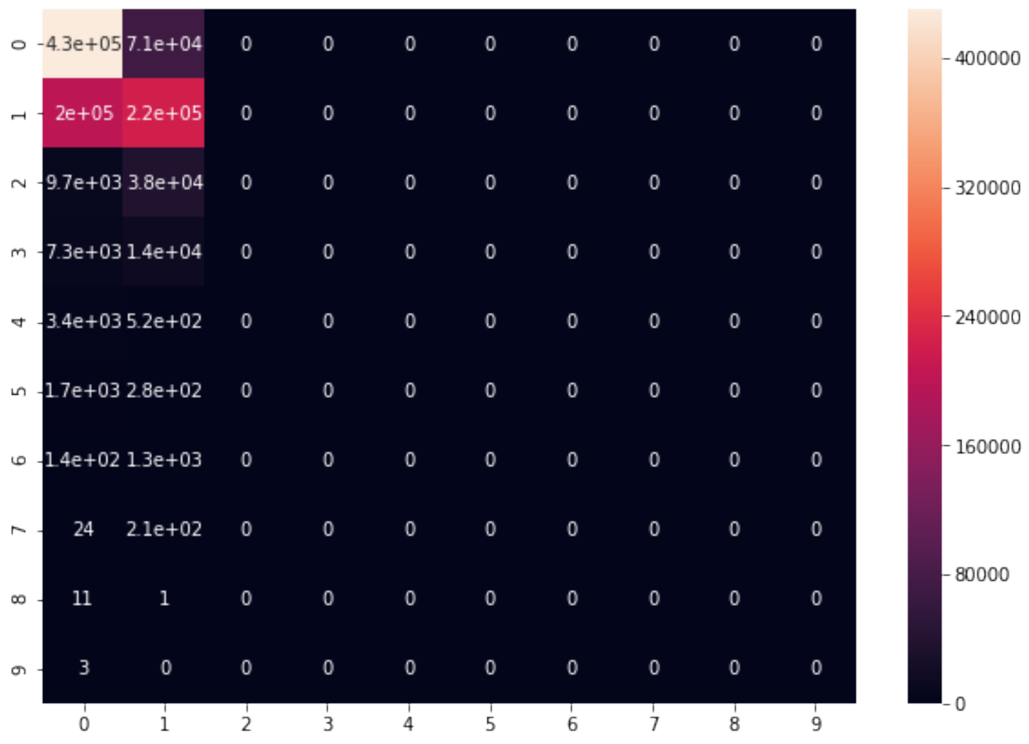
All such specifications given in a config file.

Loss – Cross Entropy (Softmax activation)

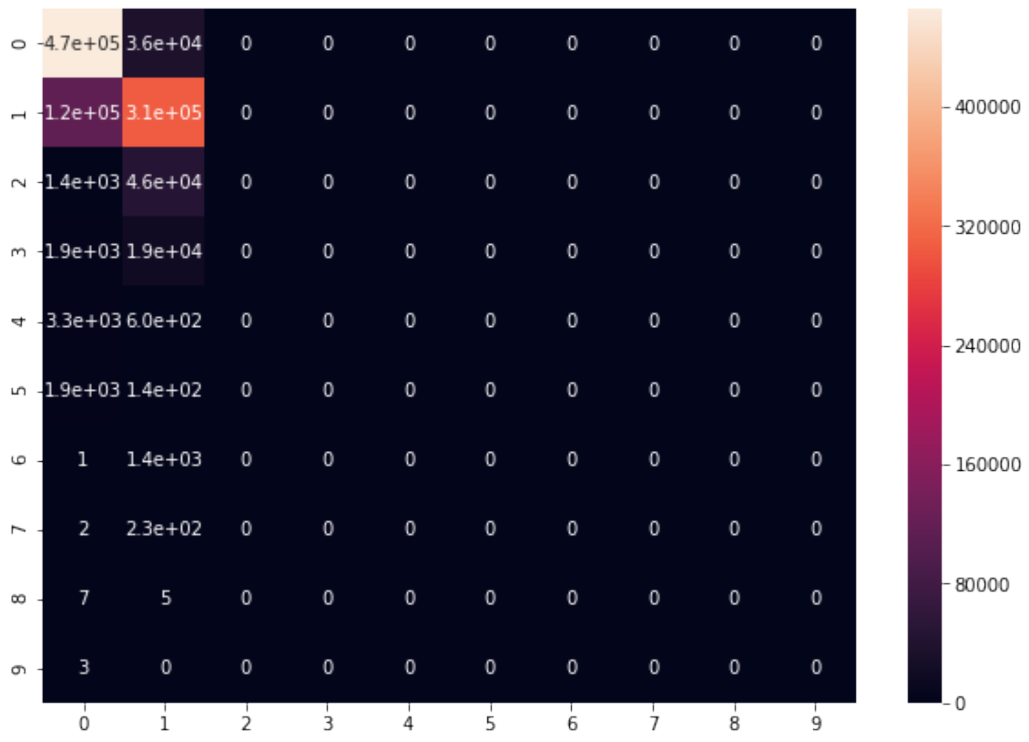
Part (c) – One Hidden Layer:



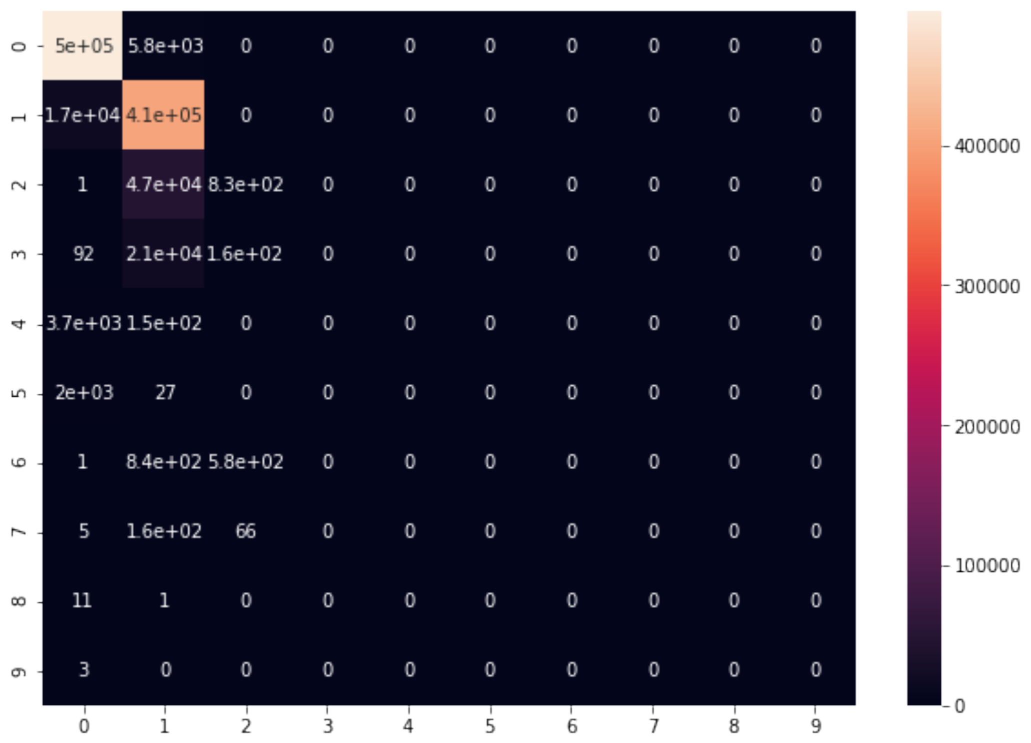
Confusion Matrices:



5 Units

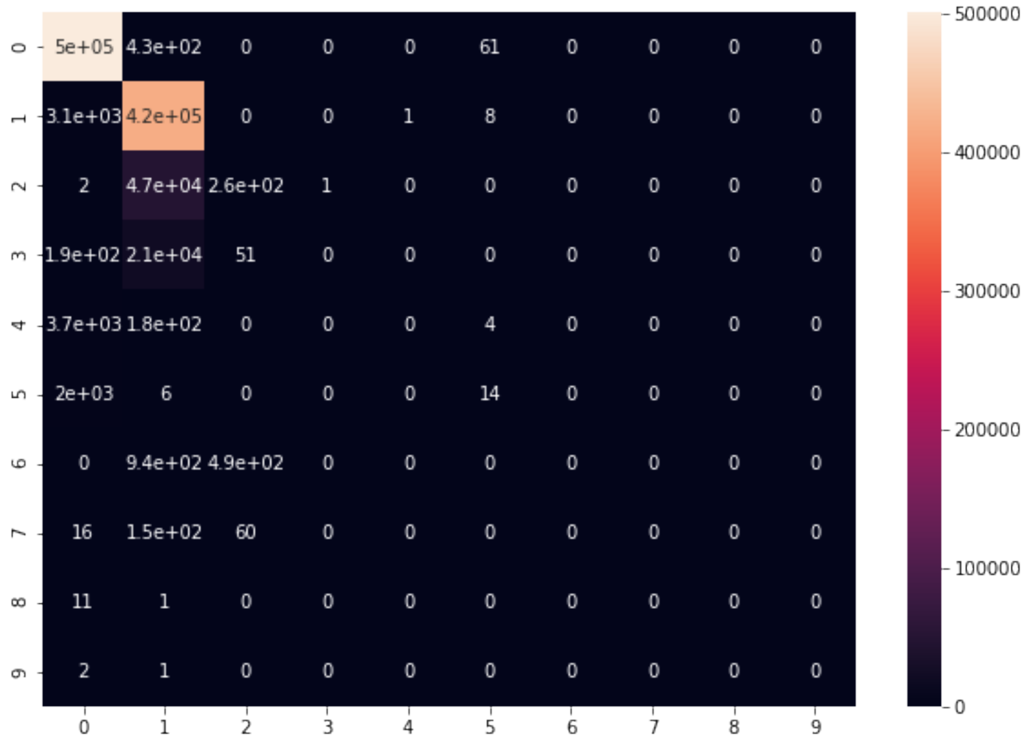


10 Units



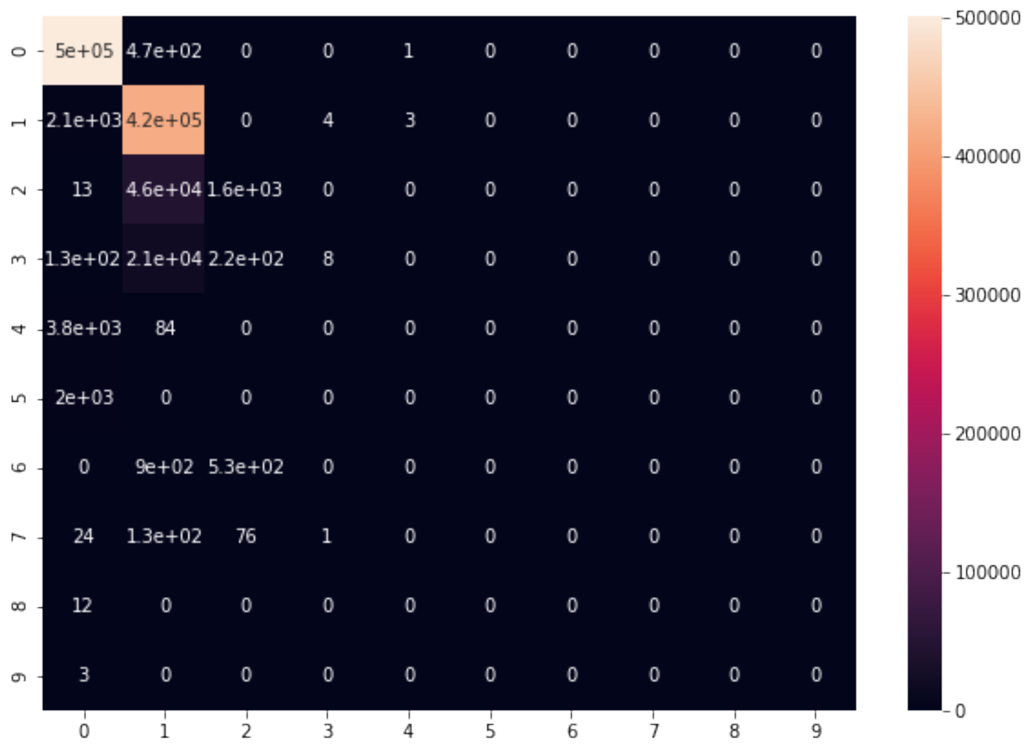
	0	1	2	3	4	5	6	7	8	9
0	5e+05	5.8e+03	0	0	0	0	0	0	0	0
1	1.7e+04	4.1e+05	0	0	0	0	0	0	0	0
2	1	4.7e+04	8.3e+02	0	0	0	0	0	0	0
3	92	2.1e+04	1.6e+02	0	0	0	0	0	0	0
4	3.7e+03	1.5e+02	0	0	0	0	0	0	0	0
5	2e+03	27	0	0	0	0	0	0	0	0
6	1	8.4e+02	5.8e+02	0	0	0	0	0	0	0
7	5	1.6e+02	66	0	0	0	0	0	0	0
8	11	1	0	0	0	0	0	0	0	0
9	3	0	0	0	0	0	0	0	0	0

15 Units



	0	1	2	3	4	5	6	7	8	9
0	5e+05	4.3e+02	0	0	0	61	0	0	0	0
1	3.1e+03	4.2e+05	0	0	1	8	0	0	0	0
2	2	4.7e+04	2.6e+02	1	0	0	0	0	0	0
3	1.9e+02	2.1e+04	51	0	0	0	0	0	0	0
4	3.7e+03	1.8e+02	0	0	0	4	0	0	0	0
5	2e+03	6	0	0	0	14	0	0	0	0
6	0	9.4e+02	4.9e+02	0	0	0	0	0	0	0
7	16	1.5e+02	60	0	0	0	0	0	0	0
8	11	1	0	0	0	0	0	0	0	0
9	2	1	0	0	0	0	0	0	0	0

20 units



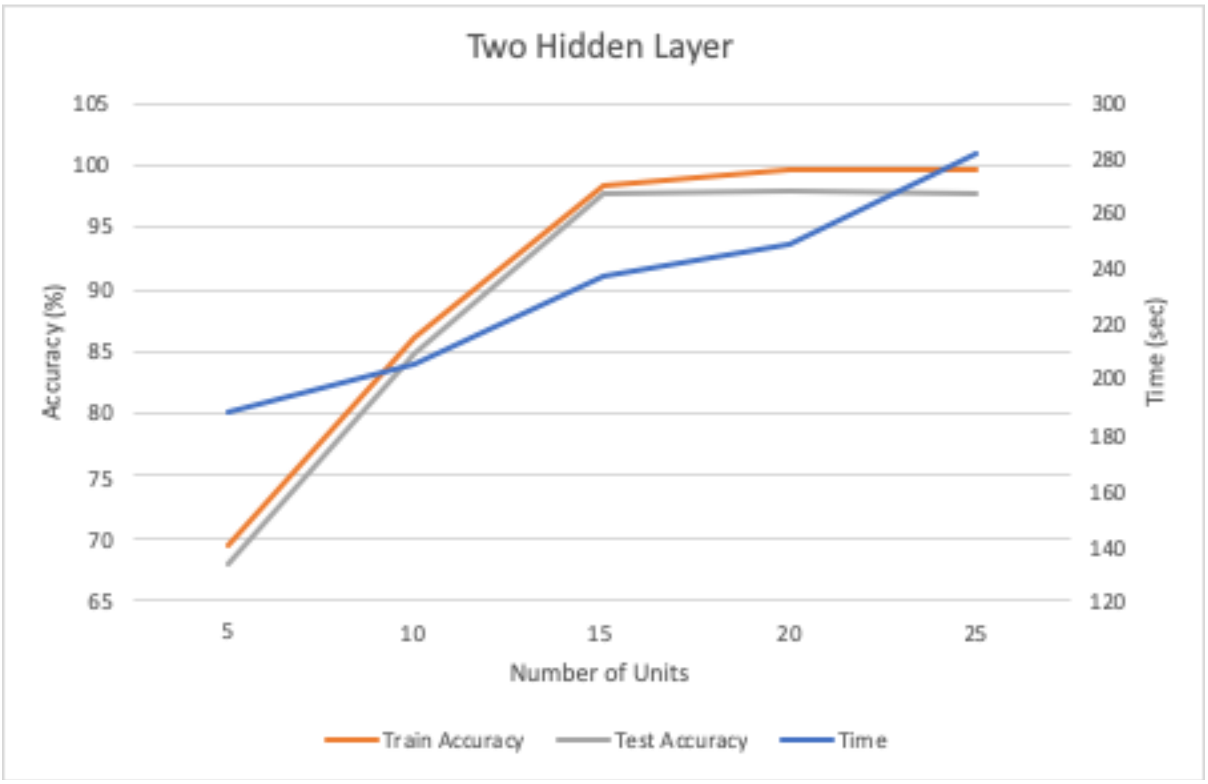
25 units

Observations:

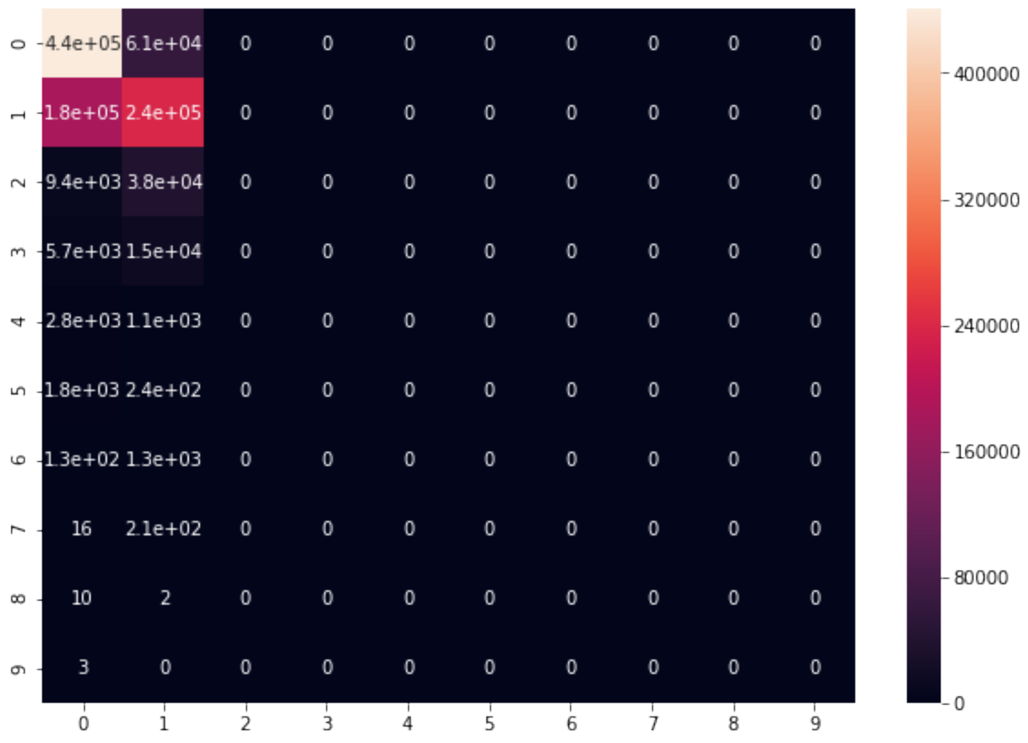
On increasing the number of units from 5-25, the training time increases and the accuracy also increases. Looking at the confusion matrix, the classifier starts learning how to classify more classes apart from the dominant 2.

Moreover, both training and testing accuracies are increasing, so it is not overfitting yet.

Part (d) – Two Hidden Layers:

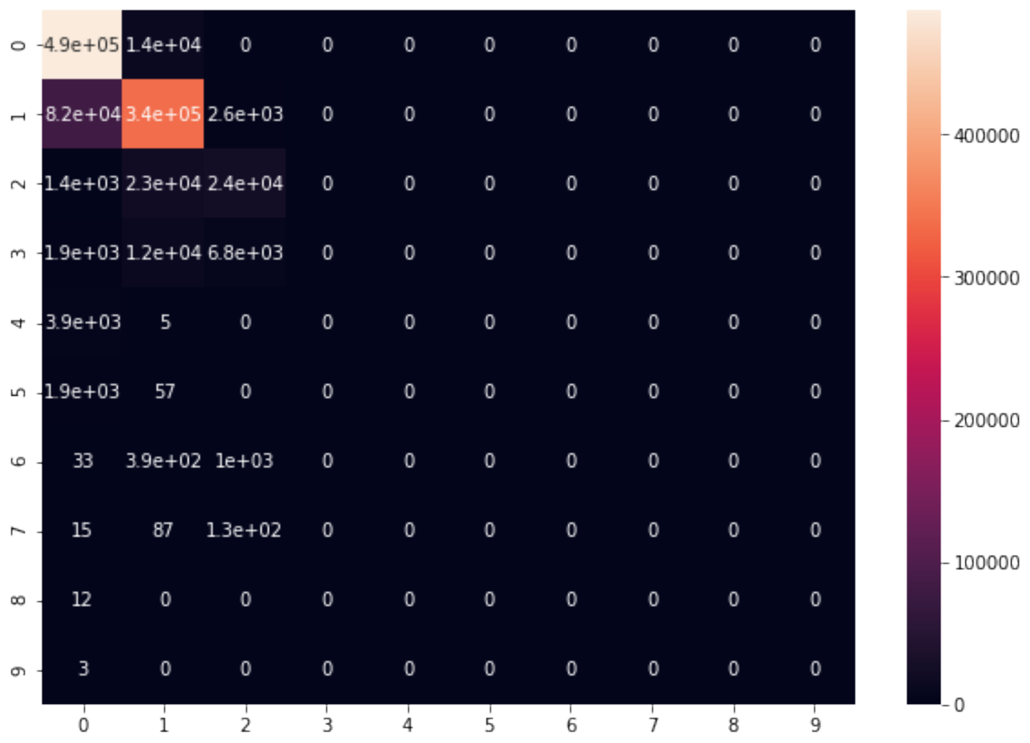


Confusion Matrices:

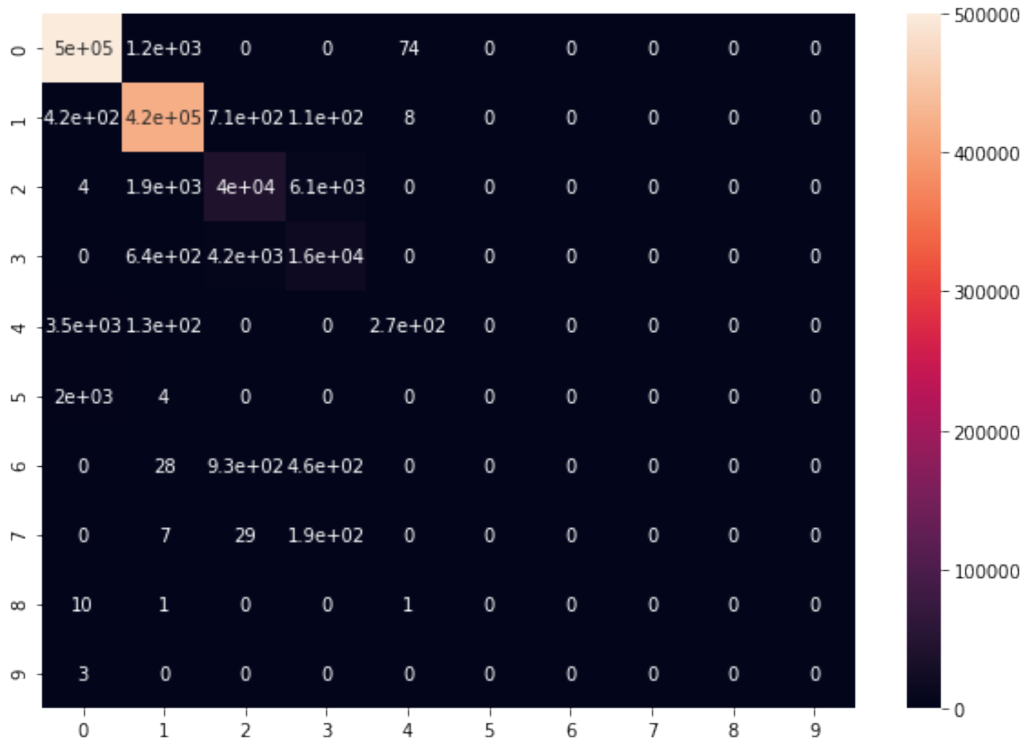


	0	1	2	3	4	5	6	7	8	9
0	4.4e+05	6.1e+04	0	0	0	0	0	0	0	0
1	1.8e+05	2.4e+05	0	0	0	0	0	0	0	0
2	9.4e+03	3.8e+04	0	0	0	0	0	0	0	0
3	5.7e+03	1.5e+04	0	0	0	0	0	0	0	0
4	2.8e+03	1.1e+03	0	0	0	0	0	0	0	0
5	1.8e+03	2.4e+02	0	0	0	0	0	0	0	0
6	1.3e+02	1.3e+03	0	0	0	0	0	0	0	0
7	16	2.1e+02	0	0	0	0	0	0	0	0
8	10	2	0	0	0	0	0	0	0	0
9	3	0	0	0	0	0	0	0	0	0

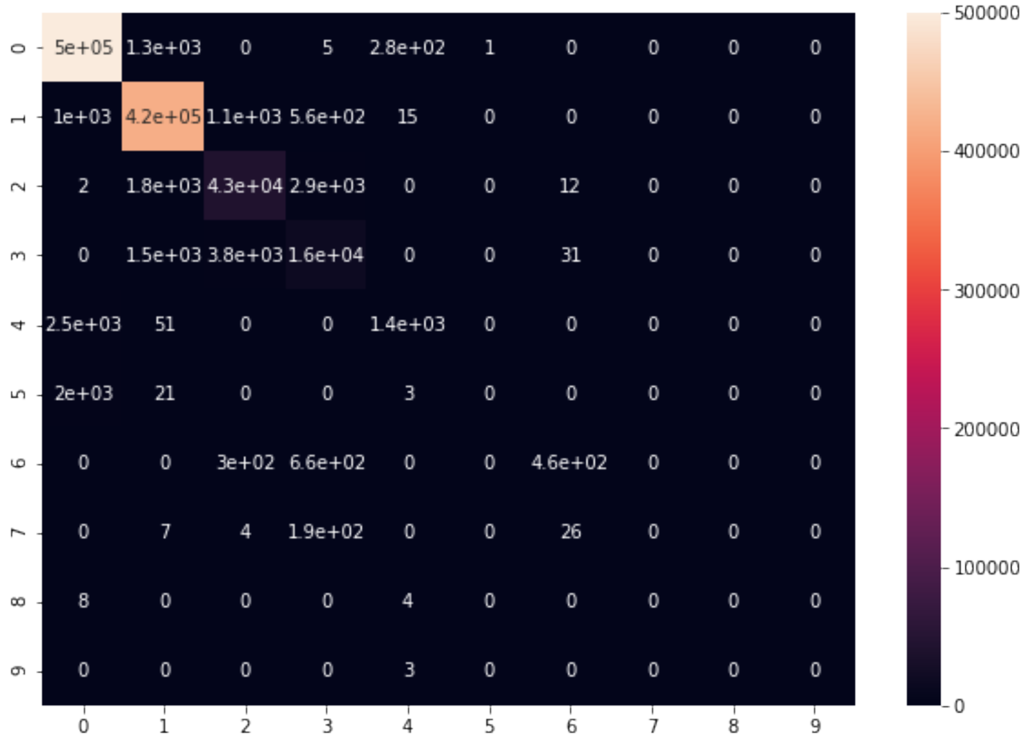
5 Units



10 Units

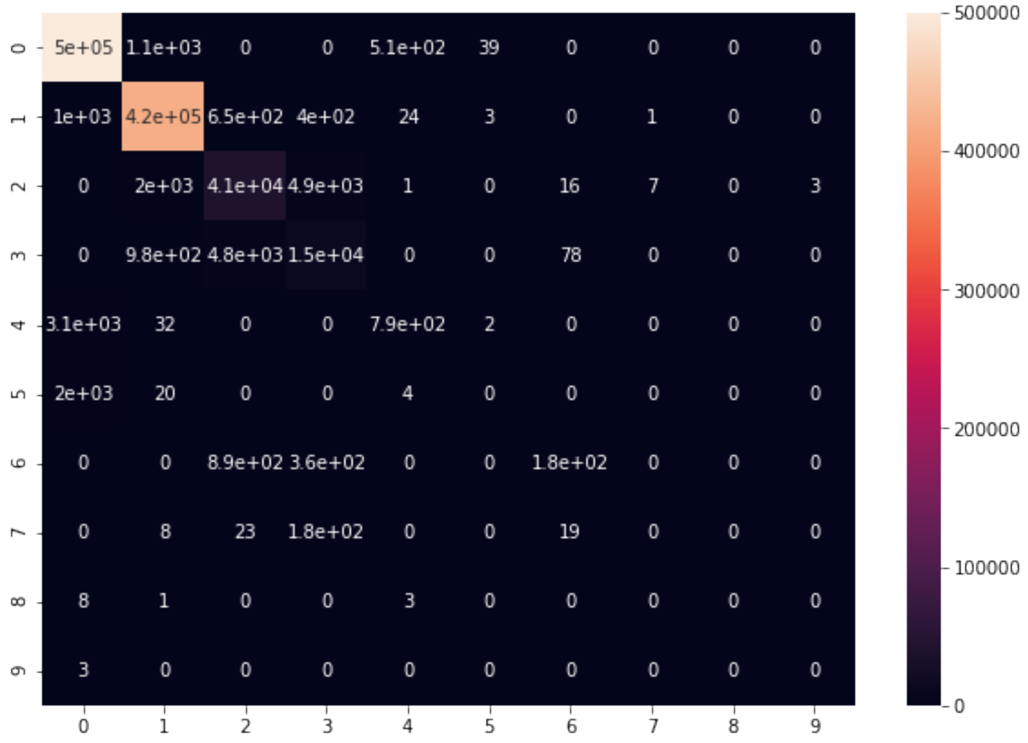


15 Units



	0	1	2	3	4	5	6	7	8	9
0	5e+05	1.3e+03	0	5	2.8e+02	1	0	0	0	0
1	1e+03	4.2e+05	1.1e+03	5.6e+02	15	0	0	0	0	0
2	2	1.8e+03	4.3e+04	2.9e+03	0	0	12	0	0	0
3	0	1.5e+03	3.8e+03	1.6e+04	0	0	31	0	0	0
4	2.5e+03	51	0	0	1.4e+03	0	0	0	0	0
5	2e+03	21	0	0	3	0	0	0	0	0
6	0	0	3e+02	6.6e+02	0	0	4.6e+02	0	0	0
7	0	7	4	1.9e+02	0	0	26	0	0	0
8	8	0	0	0	4	0	0	0	0	0
9	0	0	0	0	3	0	0	0	0	0

20 units



	0	1	2	3	4	5	6	7	8	9
0	5e+05	1.1e+03	0	0	5.1e+02	39	0	0	0	0
1	1e+03	4.2e+05	6.5e+02	4e+02	24	3	0	1	0	0
2	0	2e+03	4.1e+04	4.9e+03	1	0	16	7	0	3
3	0	9.8e+02	4.8e+03	1.5e+04	0	0	78	0	0	0
4	3.1e+03	32	0	0	7.9e+02	2	0	0	0	0
5	2e+03	20	0	0	4	0	0	0	0	0
6	0	0	8.9e+02	3.6e+02	0	0	1.8e+02	0	0	0
7	0	8	23	1.8e+02	0	0	19	0	0	0
8	8	1	0	0	3	0	0	0	0	0
9	3	0	0	0	0	0	0	0	0	0

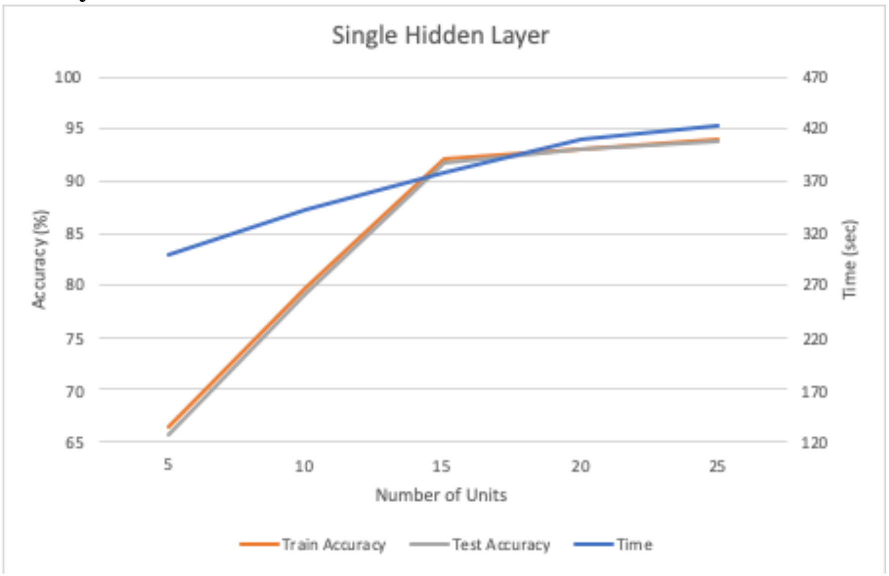
25 units

Observations:

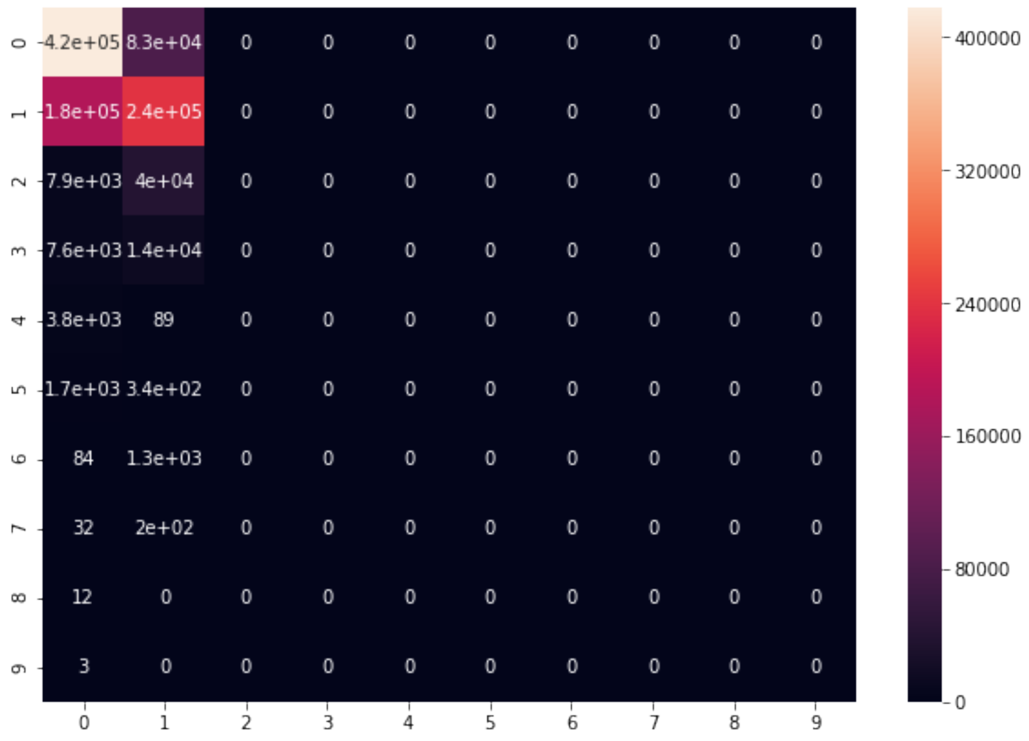
Here we see, as we increase the number of layers, accuracy is again increasing. The classifier starts predicting some classes like (6) and (4) which it didn't for the single layer one.
So it is learning more non-linear functions to better approximate this poker hand dataset.
However, after 20+ neurons per layer, we see the test accuracy falls for 25 neurons. This can be due to overfitting in this case, where train accuracy is improving, but it isn't generalizing well to the test case.

Part (e) – Variable Learning Rate:

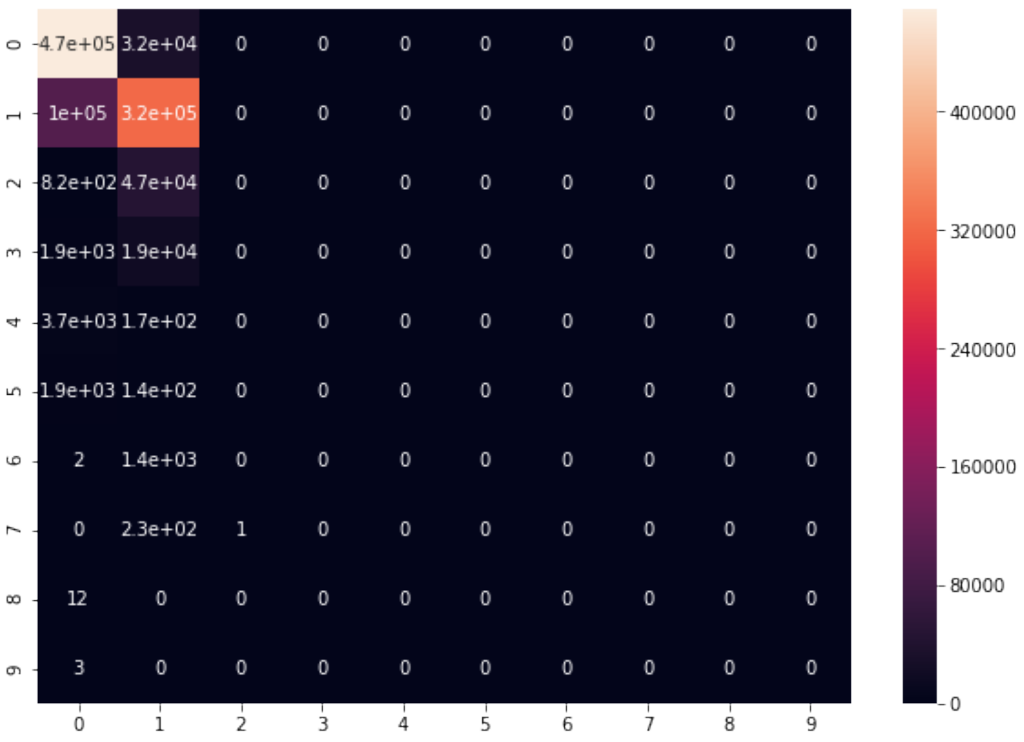
Single Hidden Layer:



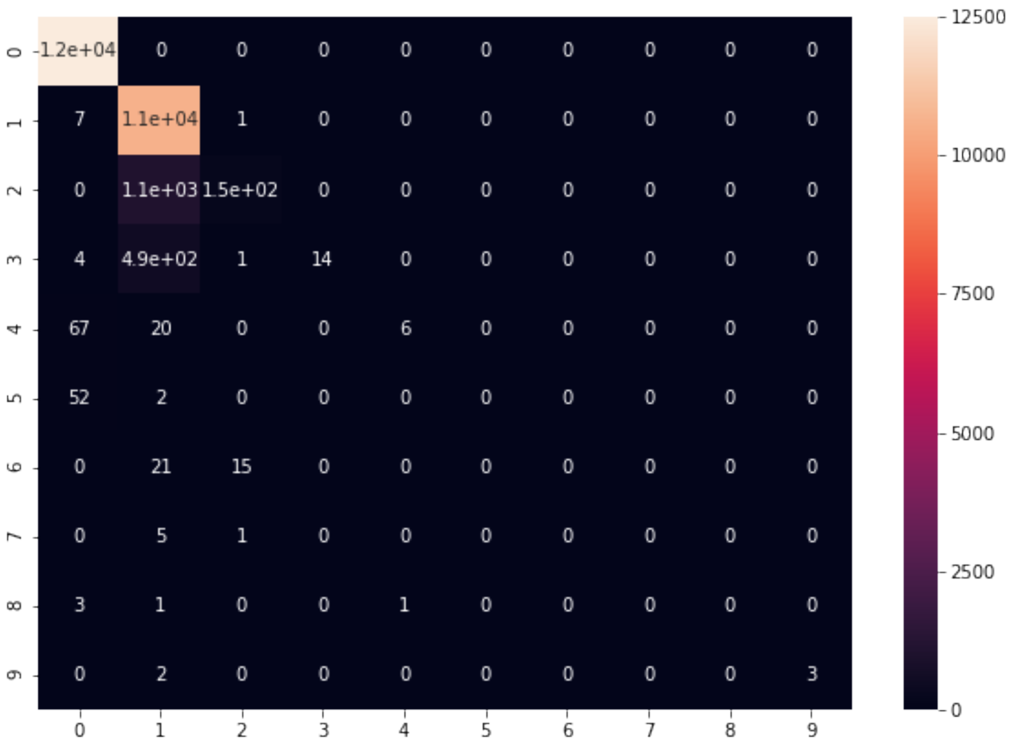
Confusion Matrices:



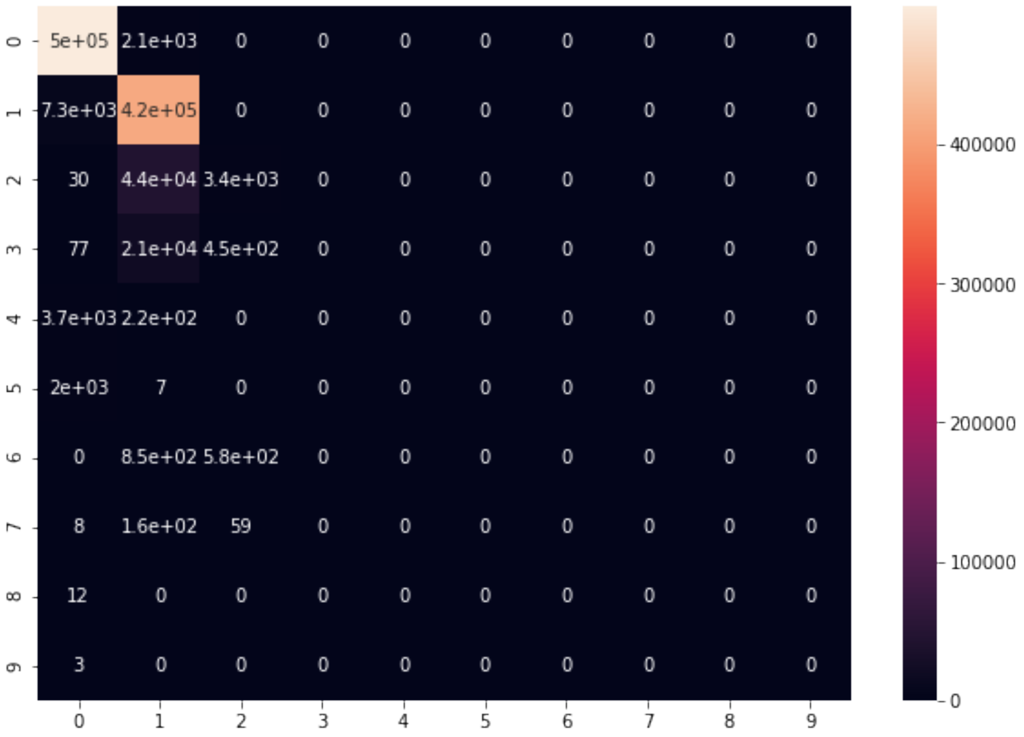
5 Units



10 Units

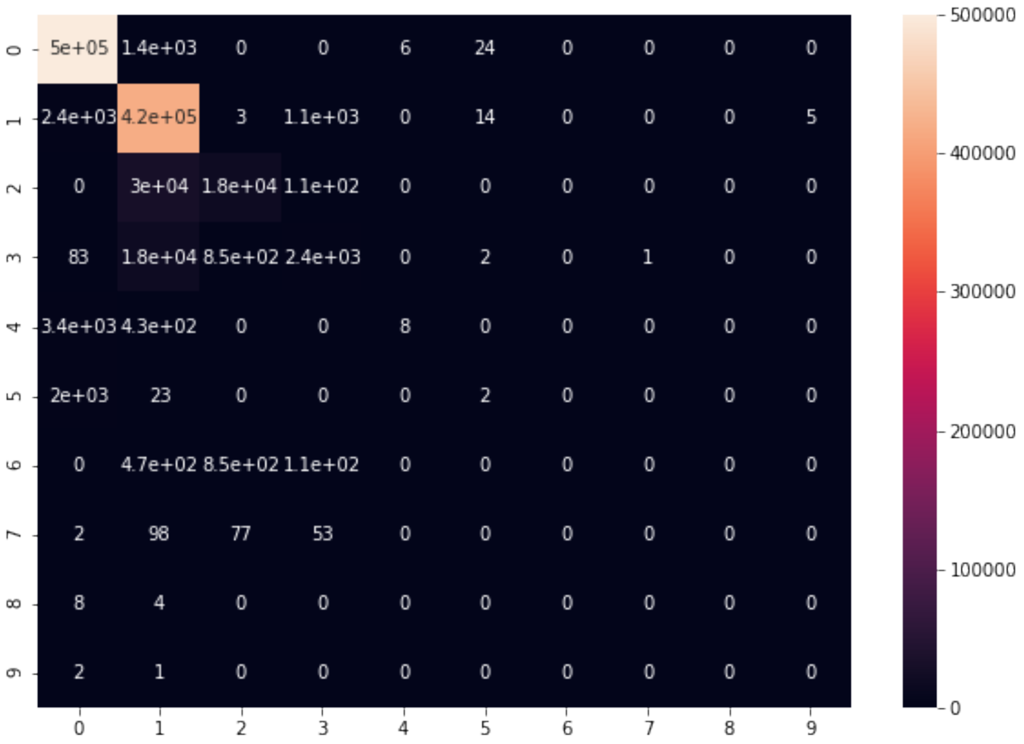


15 Units



	0	1	2	3	4	5	6	7	8	9
0	5e+05	2.1e+03	0	0	0	0	0	0	0	0
1	7.3e+03	4.2e+05	0	0	0	0	0	0	0	0
2	30	4.4e+04	3.4e+03	0	0	0	0	0	0	0
3	77	2.1e+04	4.5e+02	0	0	0	0	0	0	0
4	3.7e+03	2.2e+02	0	0	0	0	0	0	0	0
5	2e+03	7	0	0	0	0	0	0	0	0
6	0	8.5e+02	5.8e+02	0	0	0	0	0	0	0
7	8	1.6e+02	59	0	0	0	0	0	0	0
8	12	0	0	0	0	0	0	0	0	0
9	3	0	0	0	0	0	0	0	0	0

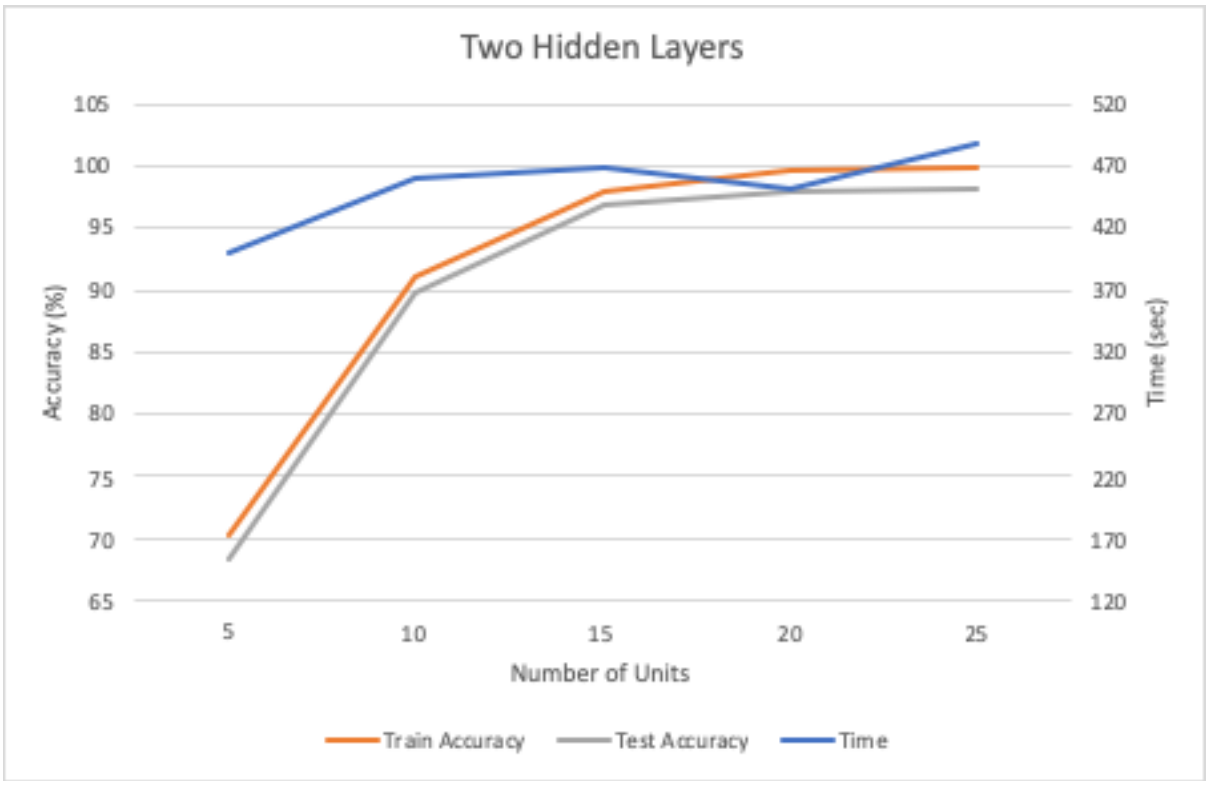
20 units



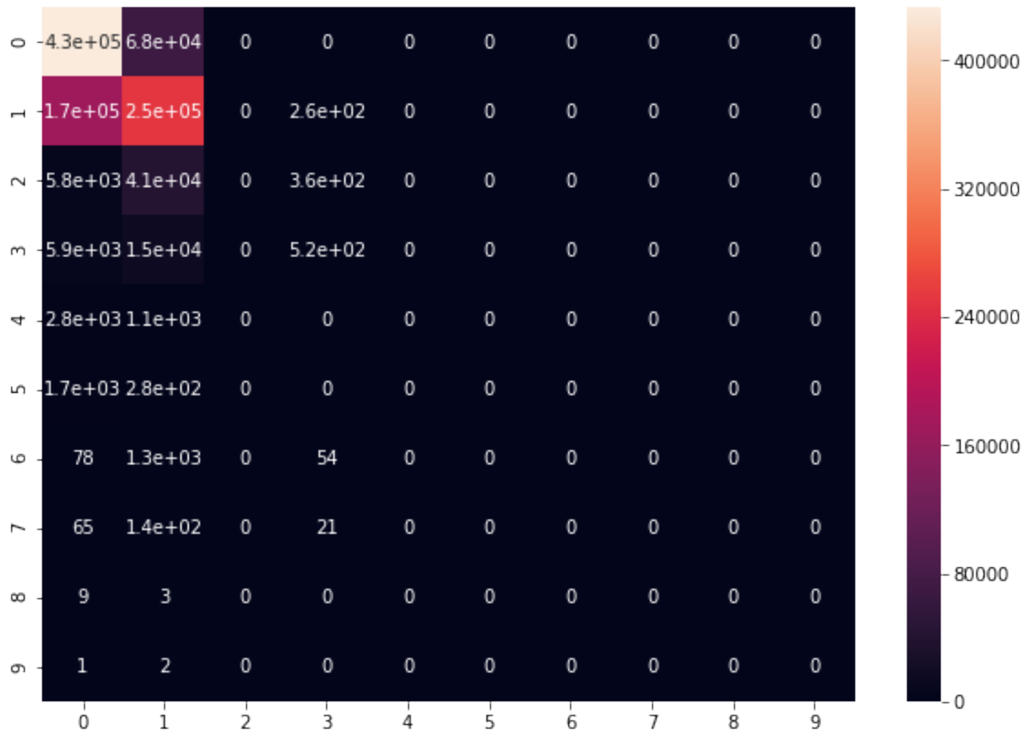
	0	1	2	3	4	5	6	7	8	9
0	5e+05	1.4e+03	0	0	6	24	0	0	0	0
1	2.4e+03	4.2e+05	3	1.1e+03	0	14	0	0	0	5
2	0	3e+04	1.8e+04	1.1e+02	0	0	0	0	0	0
3	83	1.8e+04	8.5e+02	2.4e+03	0	2	0	1	0	0
4	3.4e+03	4.3e+02	0	0	8	0	0	0	0	0
5	2e+03	23	0	0	0	2	0	0	0	0
6	0	4.7e+02	8.5e+02	1.1e+02	0	0	0	0	0	0
7	2	98	77	53	0	0	0	0	0	0
8	8	4	0	0	0	0	0	0	0	0
9	2	1	0	0	0	0	0	0	0	0

25 units

Two Hidden Layers:

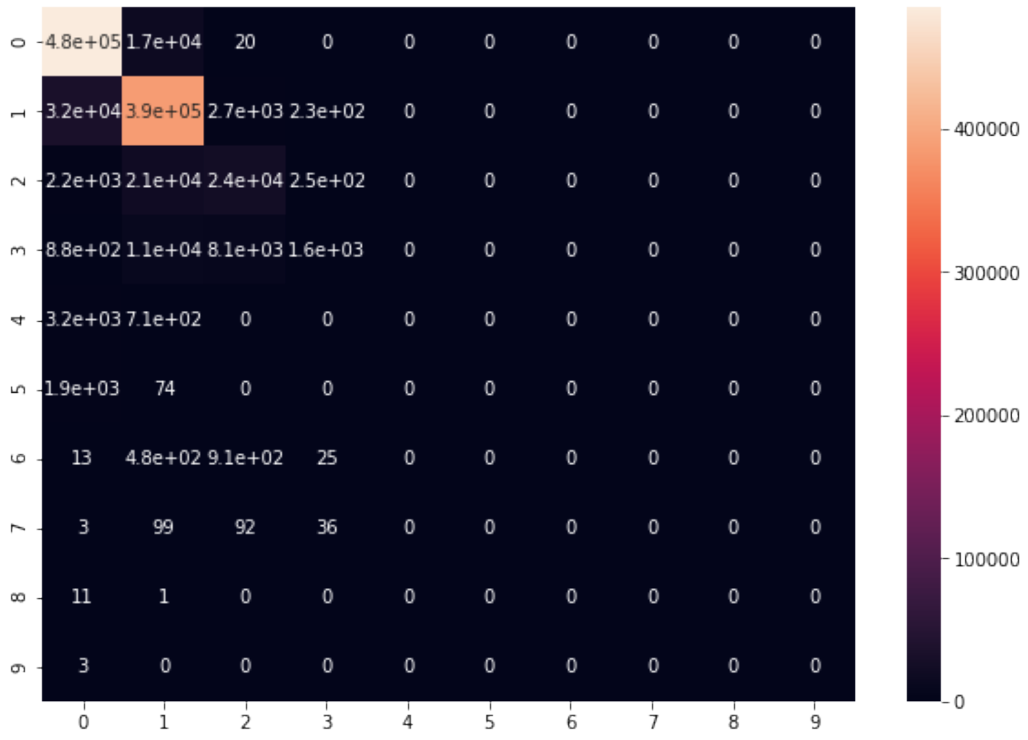


Confusion Matrices:



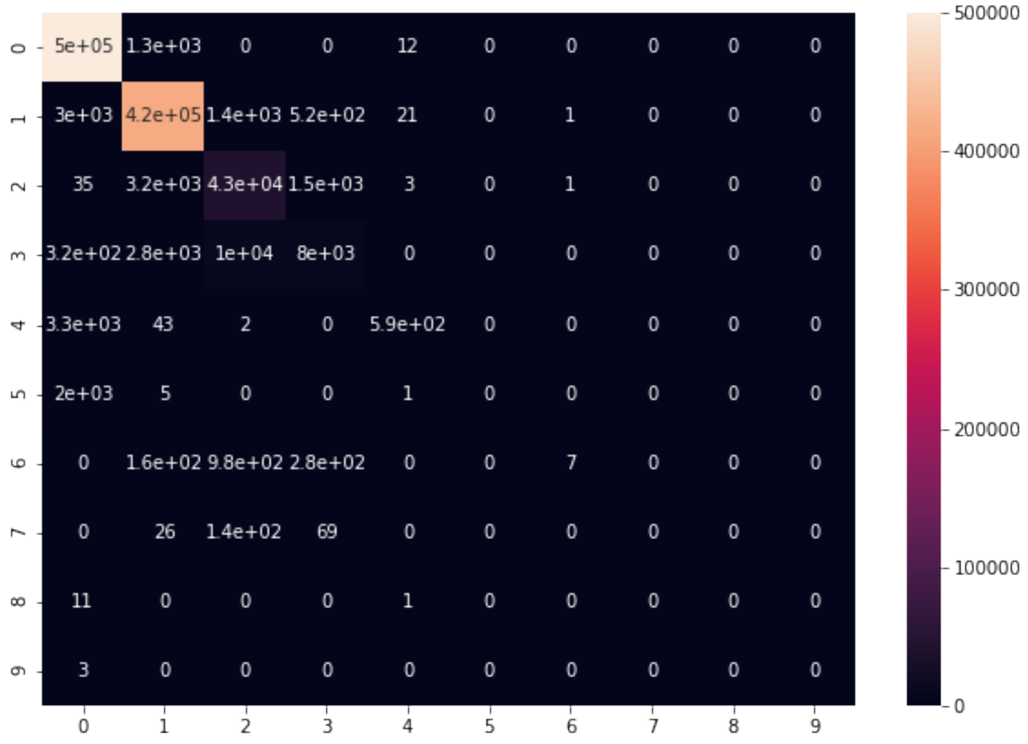
	0	1	2	3	4	5	6	7	8	9
0	4.3e+05	6.8e+04	0	0	0	0	0	0	0	0
1	1.7e+05	2.5e+05	0	2.6e+02	0	0	0	0	0	0
2	5.8e+03	4.1e+04	0	3.6e+02	0	0	0	0	0	0
3	5.9e+03	1.5e+04	0	5.2e+02	0	0	0	0	0	0
4	2.8e+03	1.1e+03	0	0	0	0	0	0	0	0
5	1.7e+03	2.8e+02	0	0	0	0	0	0	0	0
6	78	1.3e+03	0	54	0	0	0	0	0	0
7	65	1.4e+02	0	21	0	0	0	0	0	0
8	9	3	0	0	0	0	0	0	0	0
9	1	2	0	0	0	0	0	0	0	0

5 Units



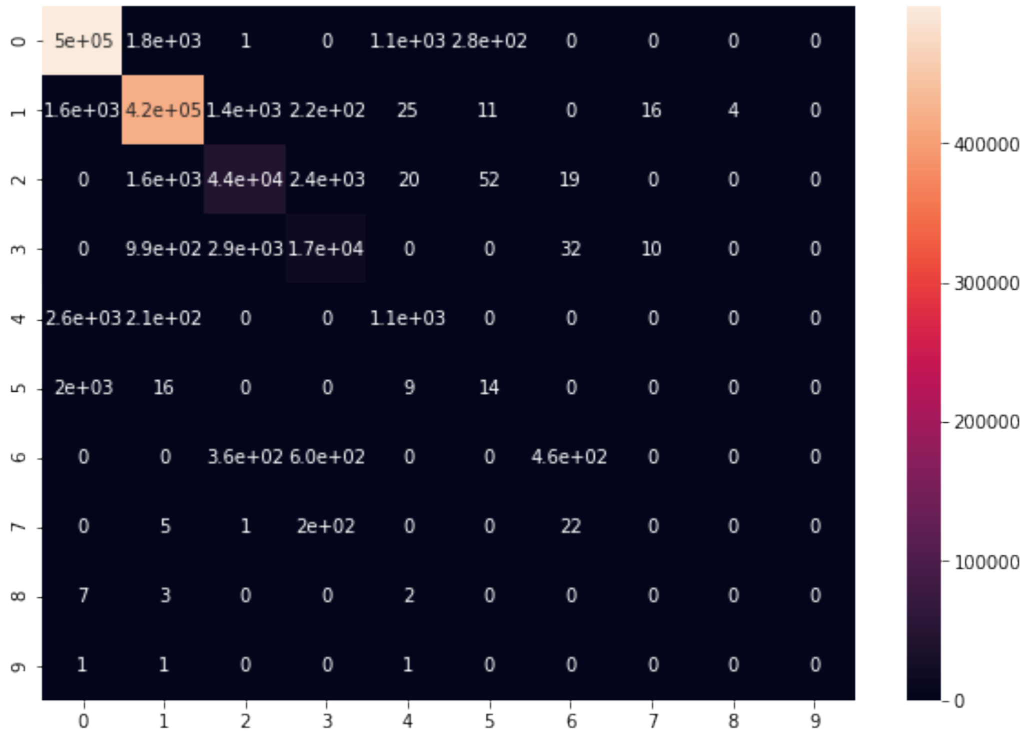
	0	1	2	3	4	5	6	7	8	9
0	4.8e+05	1.7e+04	20	0	0	0	0	0	0	0
1	3.2e+04	3.9e+05	2.7e+03	2.3e+02	0	0	0	0	0	0
2	2.2e+03	2.1e+04	2.4e+04	2.5e+02	0	0	0	0	0	0
3	8.8e+02	1.1e+04	8.1e+03	1.6e+03	0	0	0	0	0	0
4	3.2e+03	7.1e+02	0	0	0	0	0	0	0	0
5	1.9e+03	74	0	0	0	0	0	0	0	0
6	13	4.8e+02	9.1e+02	25	0	0	0	0	0	0
7	3	99	92	36	0	0	0	0	0	0
8	11	1	0	0	0	0	0	0	0	0
9	3	0	0	0	0	0	0	0	0	0

10 Units

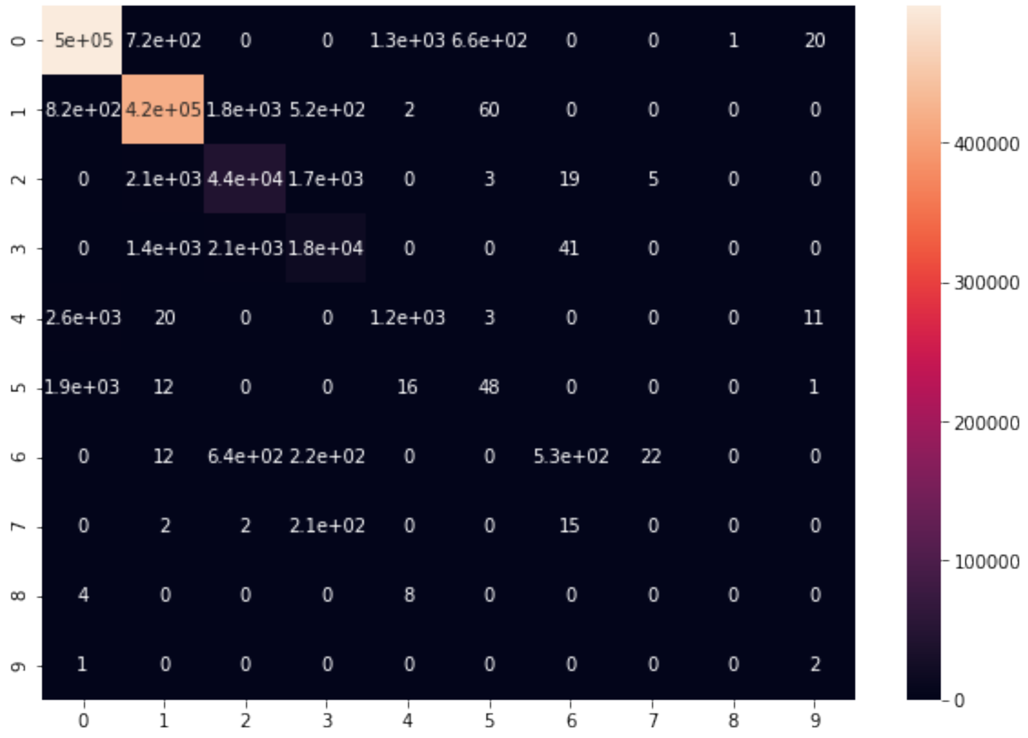


	0	1	2	3	4	5	6	7	8	9
0	5e+05	1.3e+03	0	0	12	0	0	0	0	0
1	3e+03	4.2e+05	1.4e+03	5.2e+02	21	0	1	0	0	0
2	35	3.2e+03	4.3e+04	1.5e+03	3	0	1	0	0	0
3	3.2e+02	2.8e+03	1e+04	8e+03	0	0	0	0	0	0
4	3.3e+03	43	2	0	5.9e+02	0	0	0	0	0
5	2e+03	5	0	0	1	0	0	0	0	0
6	0	1.6e+02	9.8e+02	2.8e+02	0	0	7	0	0	0
7	0	26	1.4e+02	69	0	0	0	0	0	0
8	11	0	0	0	1	0	0	0	0	0
9	3	0	0	0	0	0	0	0	0	0

15 Units



20 units



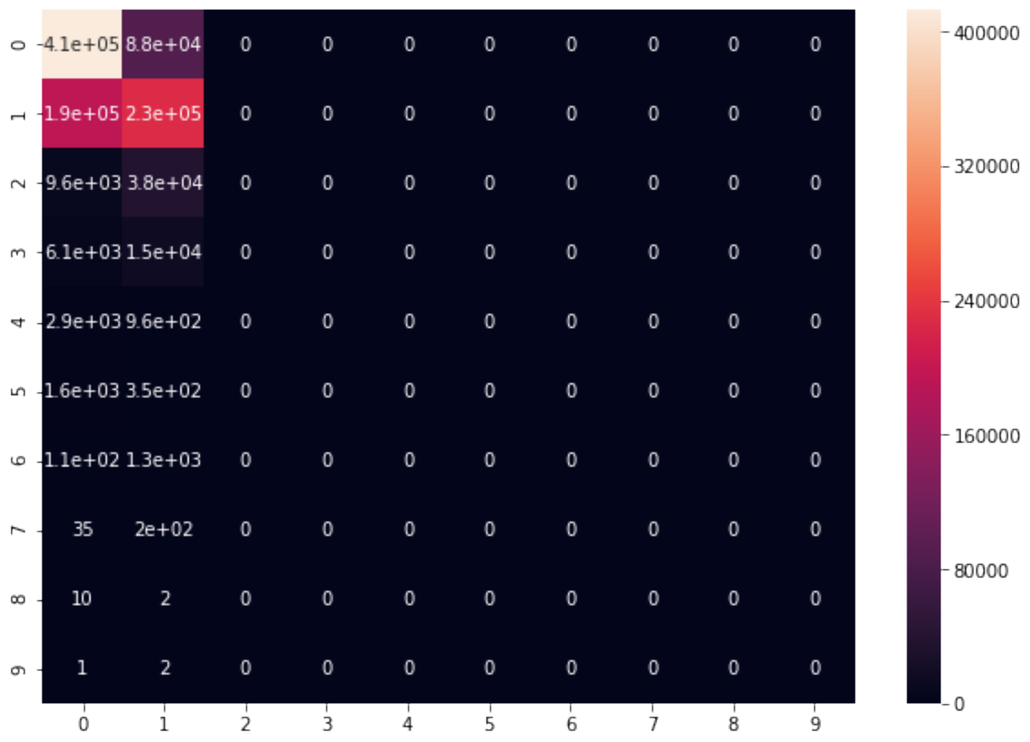
25 units

Observations:

Here, we see the variation of learning rate increases our accuracy. The plots are almost similar as compared to the fixed LR ones, except the accuracies are higher. This might be due to the function not overshooting the minima anymore so being able to find a closer approx. for the minima.

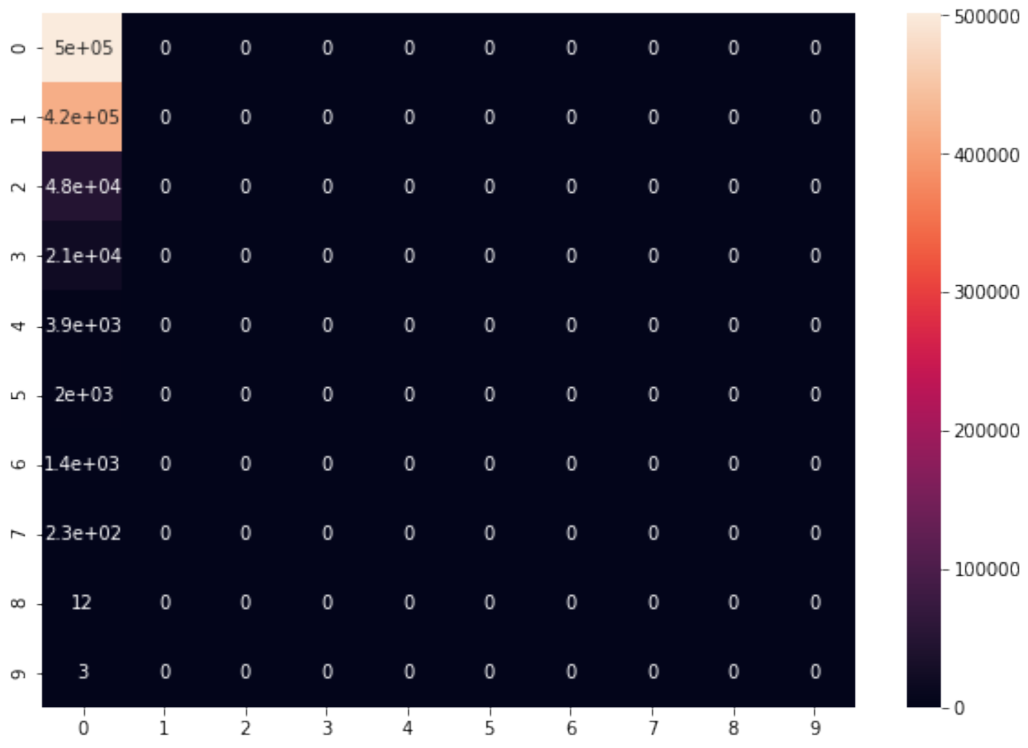
Part (f) – ReLU Activation Function:

In ReLU, the exploding gradient problem becomes apparent when our number of hidden units is large. So, for all the 2 layer networks, as well as for the large single layer network, we get an overflow in the calculations. So, it just ends up predicting the majority prediction for 50% accuracy.



This is. The confusion matrix for a 5 neuron single hidden layer.

The predictions are similar to the one for sigmoid. Since this network isn't deep, we don't encounter the vanishing gradient problem much. Hence ReLU isn't an improvement.



Here we have the confusion matrix for a network running into the exploding gradient problem, this reduces to the majority classifier. (50%)

This can be improved by gradient clipping, but without that implementation, the code outputs a majority prediction.