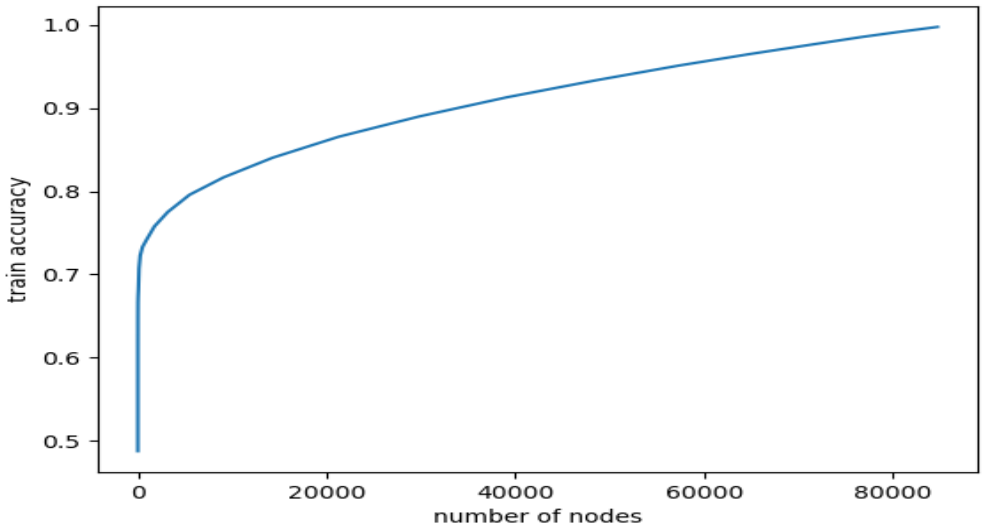
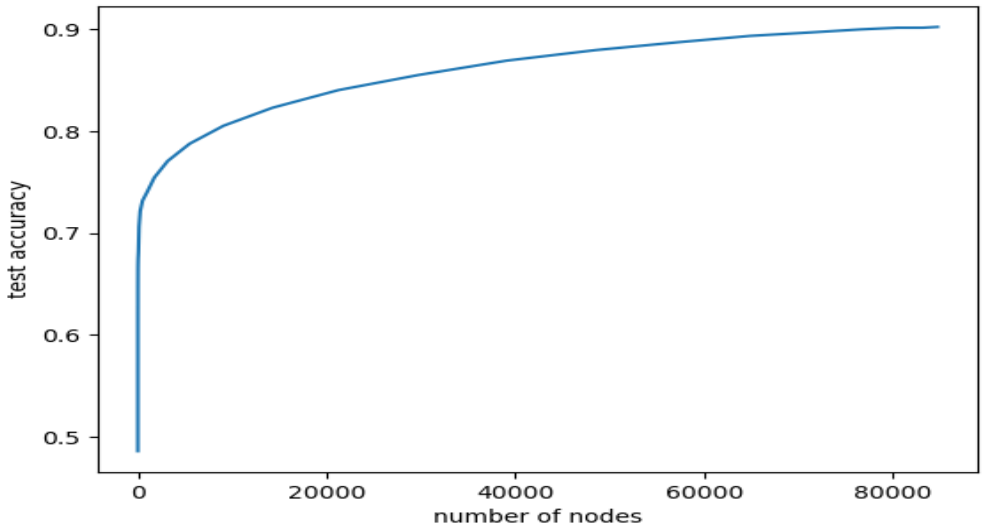
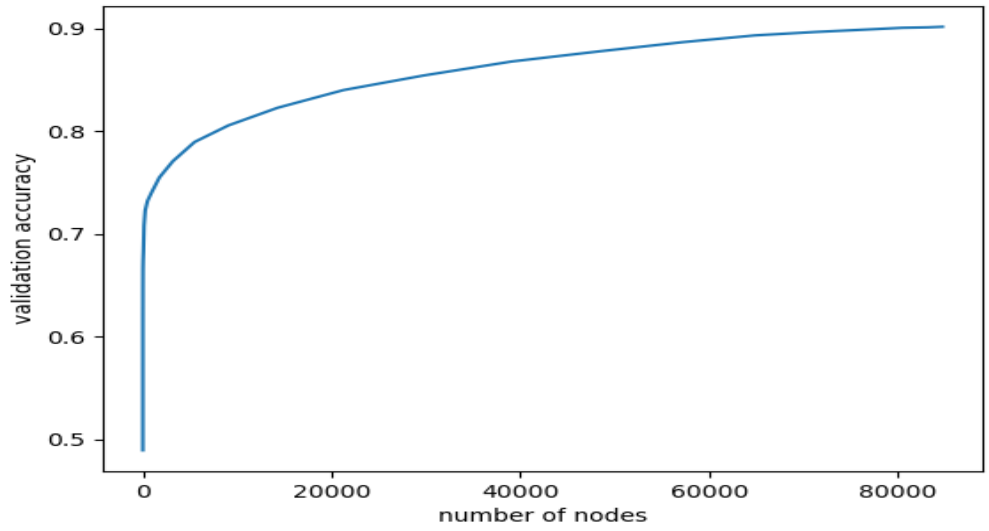
CHIRAG MOHAPATRA
2018CS50403

---

1. For this question, I implemented a Decision Tree model with the added feature of pruning.
   (a) For any node, the predicted value is the label that occurred the maximum number of times in the training data at that node.
   I have grown the tree in a breadth first manner.
   These are the accuracies I observed as I grew the tree:





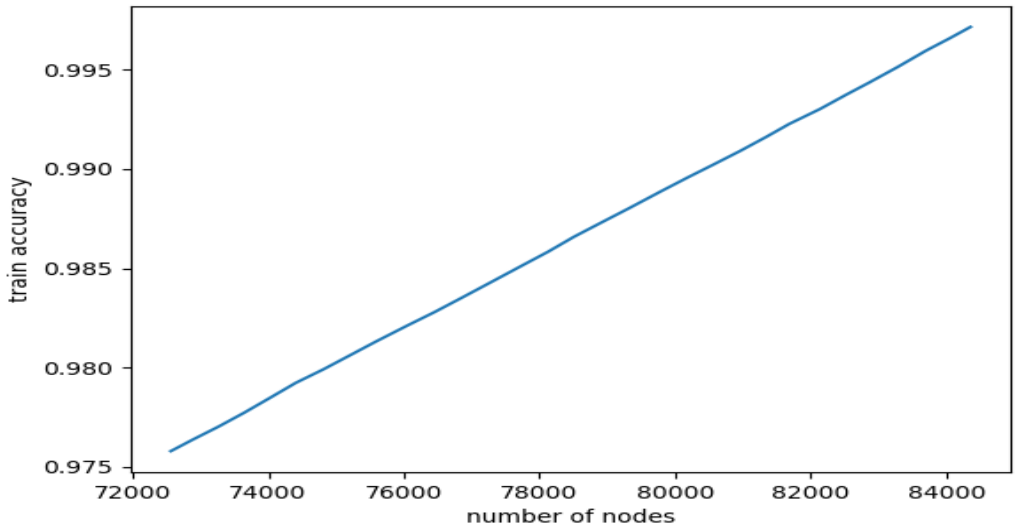


We can observe that with increasing number of nodes, the accuracies increase in a log-like form, rapidly at first and then gradual.

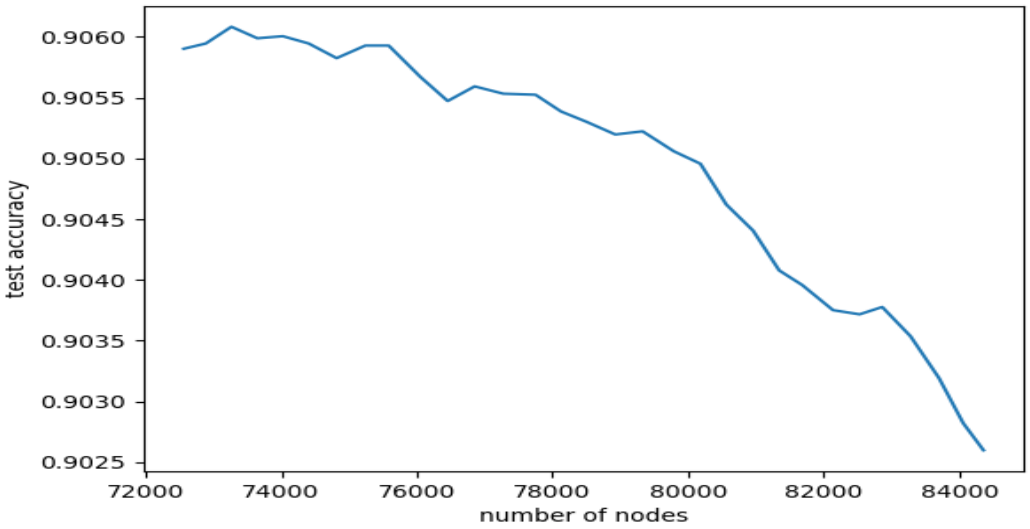
For 0 nodes, I simply predicted the majority class.

(b) I implemented pruning in the manner where if the sum of errors in the children is greater than the parent for the validation data, then prune them out.

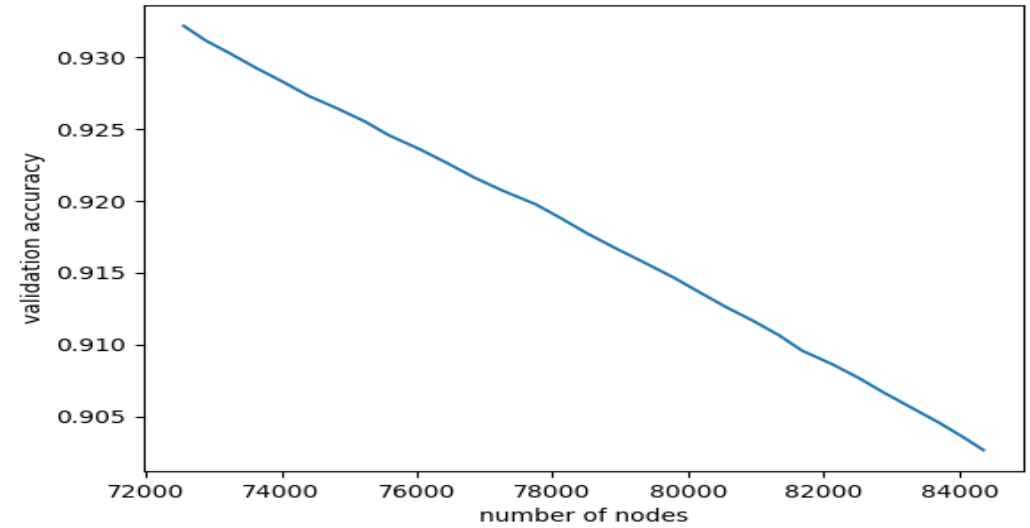
This is the plot I observed as I pruned the tree:



Training accuracy almost linearly decreases while pruning, this is expected since the tree had been grown initially increasing the training accuracy.



For test accuracy, the behaviour is somewhat erratic, but finally after complete pruning, we can see that the test accuracy has increased.



Validation accuracy increases almost linearly while pruning which is as expected.

(c) I used the Random Forest Classifier and trained it 125 times, calculating the out of bag scores to get the final set of optimal parameters.

The final set of optimal parameters that I obtained are:

$n\_estimators = 450$
$max\_features = 0.7$
$min\_samples\_split = 2$

For this optimal set of parameters,

$out\_of\_bag\ score = 0.9635649157277499$
$train\_accuracy = 1.0$
$test\_accuracy = 0.9638162142617319$

validation accuracy = 0.963529724640077

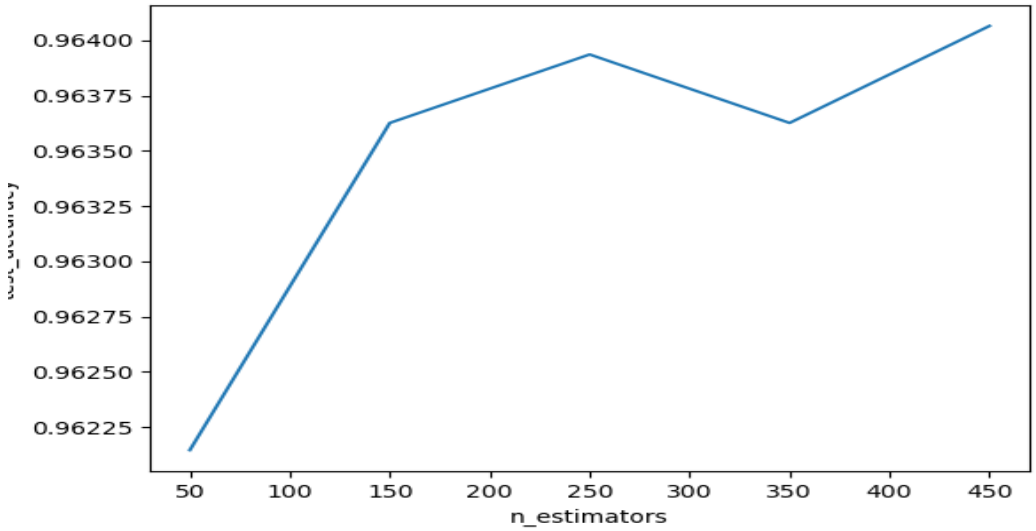
It takes around 12 hours to test all the parameters.

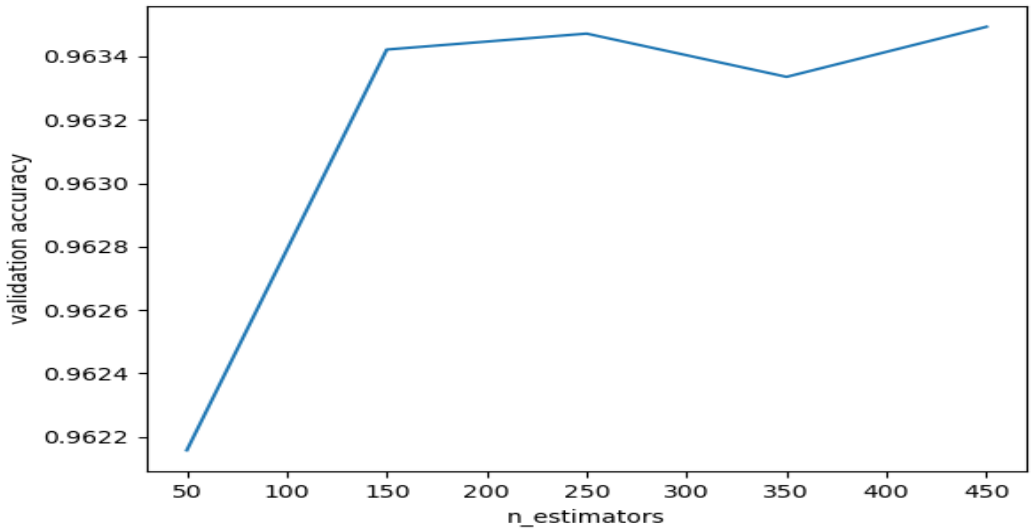
We can see that the accuracies obtained using random forests are greater than the ones we obtained using decision trees after pruning.

(d) While fixing two parameters, I varied the last one in the same range as give in part (c) above. The results obtained are:

n estimators	Test Accuracy	Validation Accuracy
50	0.9621	0.9621
150	0.9636	0.9634
250	0.9639	0.9634
350	0.9636	0.9633
450	0.9640	0.9635

The graphs are:

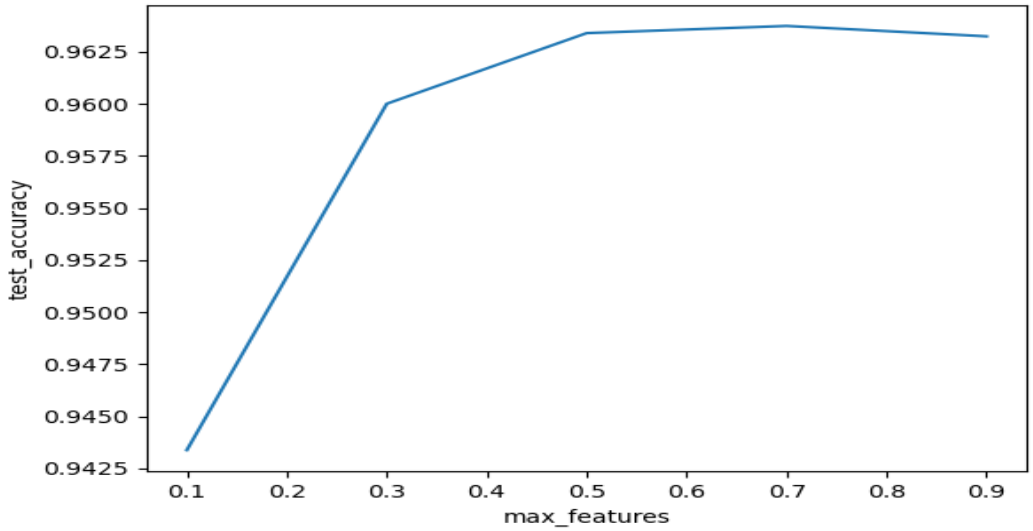


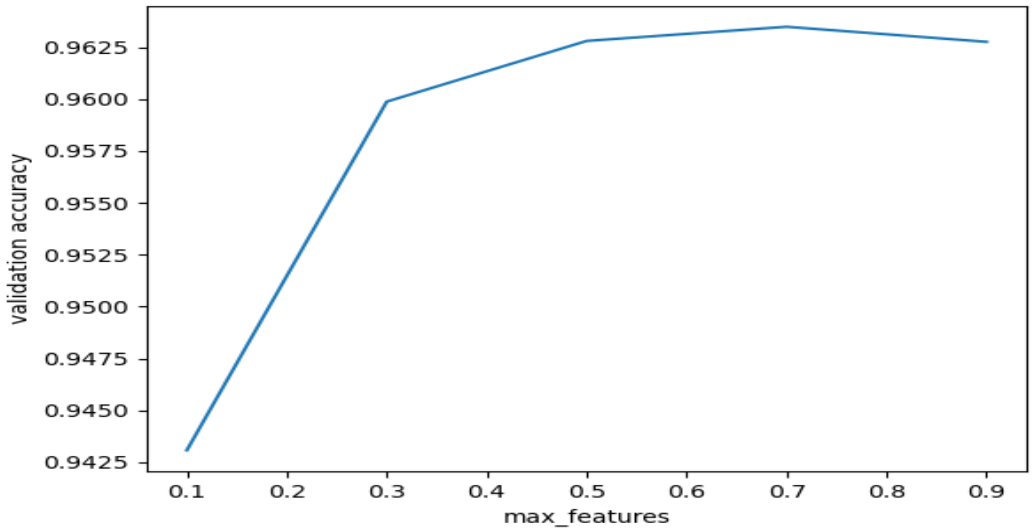


We can observe that varying *n_estimators* does not actually change the accuracies a lot.

max features	Test Accuracy	Validation Accuracy
0.1	0.9434	0.9430
0.3	0.9600	0.9598
0.5	0.9634	0.9628
0.7	0.9637	0.9635
0.9	0.9632	0.9627

These are the graphs:

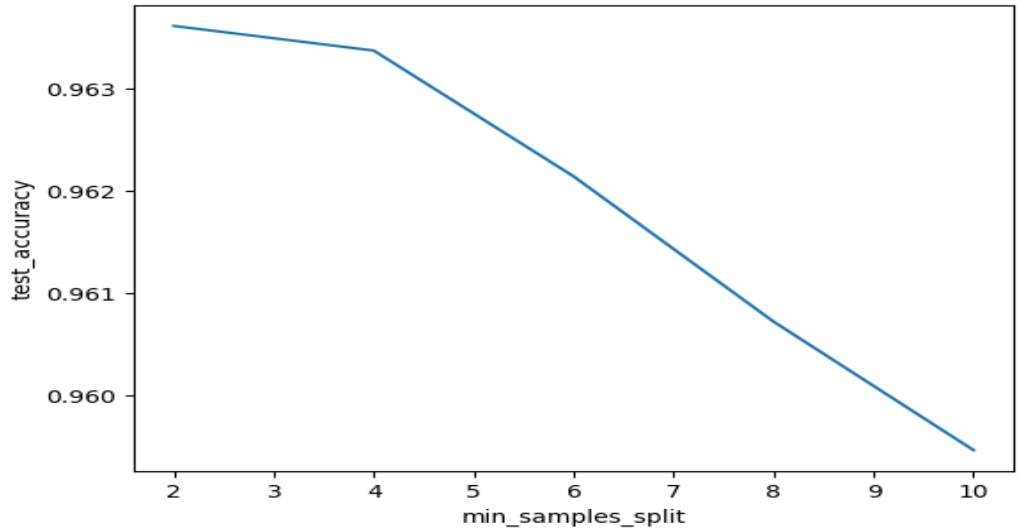


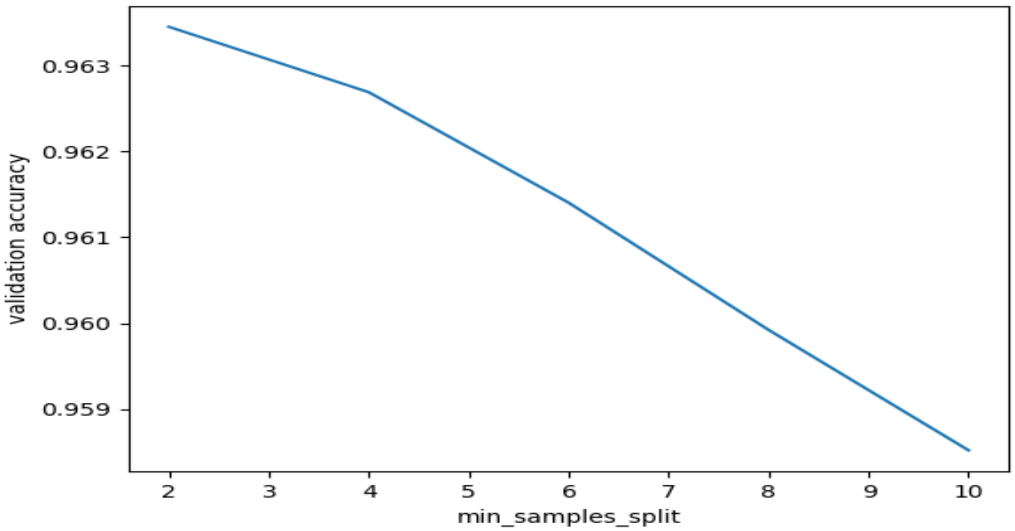


The *max_features* is slightly more senstive than n estimators and we observe greater changes.

min samples split	Test Accuracy	Validation Accuracy
2	0.9636	0.9634
4	0.9634	0.9627
6	0.9621	0.9614
8	0.9607	0.9599
10	0.9594	0.9581

This is the corresponding graph:





This parameter is also not very sensitive.

2. In this question, we had to code a neural network from scratch and then use the Kannada digits dataset to train our network.

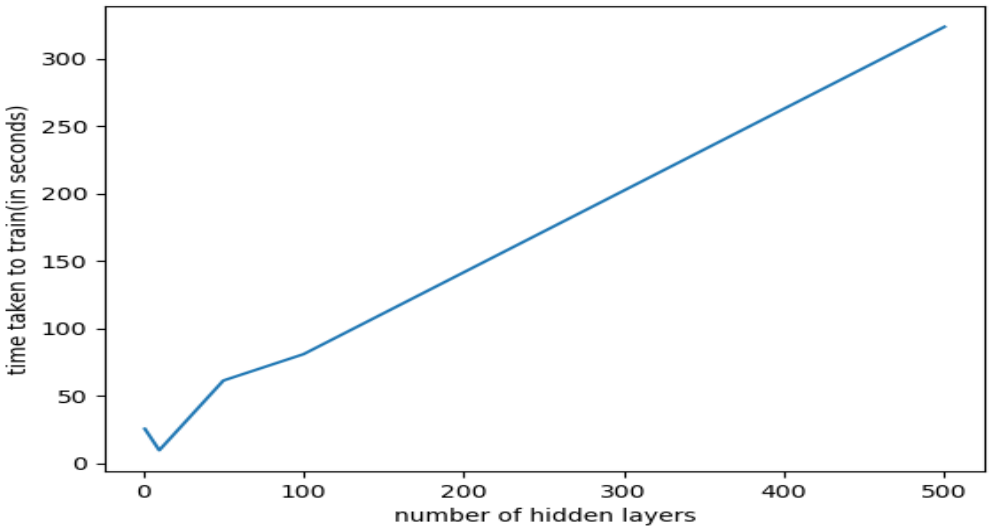
   (a) I have successfully implemented a generic neural network which can take the parameters batch size for gradient descent, the number of features of data, the number of target classes we want to classify into, the list of hidden layers, information about the learning rate(whether static and if it is, then the value or dynamic) and finally info about the activation function to be used: sigmoid or ReLu. My neural network is a fully connected one.

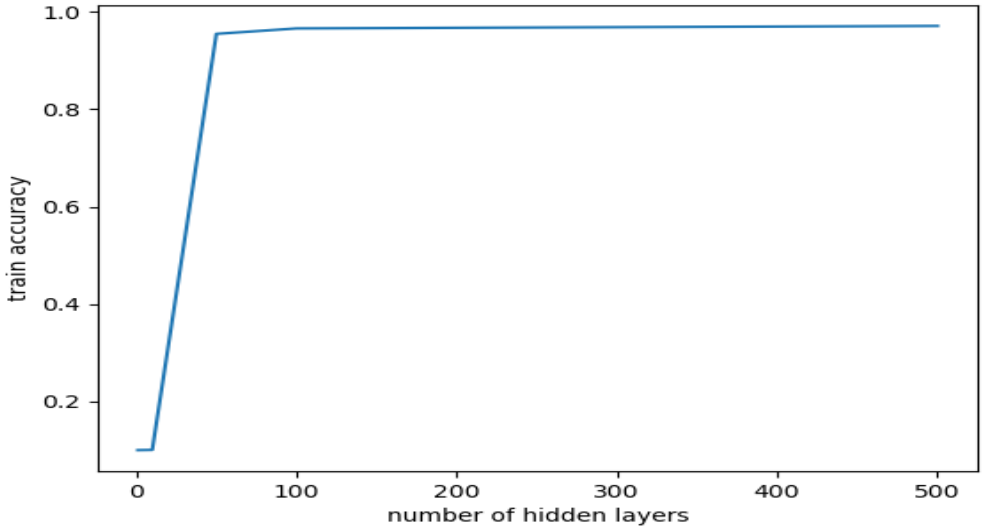
   (b) I have chosen the stopping criterion as $\epsilon = 10^{-4}$, in other words if the difference in successive values of the average mean-squared error is less than ϵ, then converge or iterate till max number of epochs(100 in my case). The learning rate was set as 0.001.

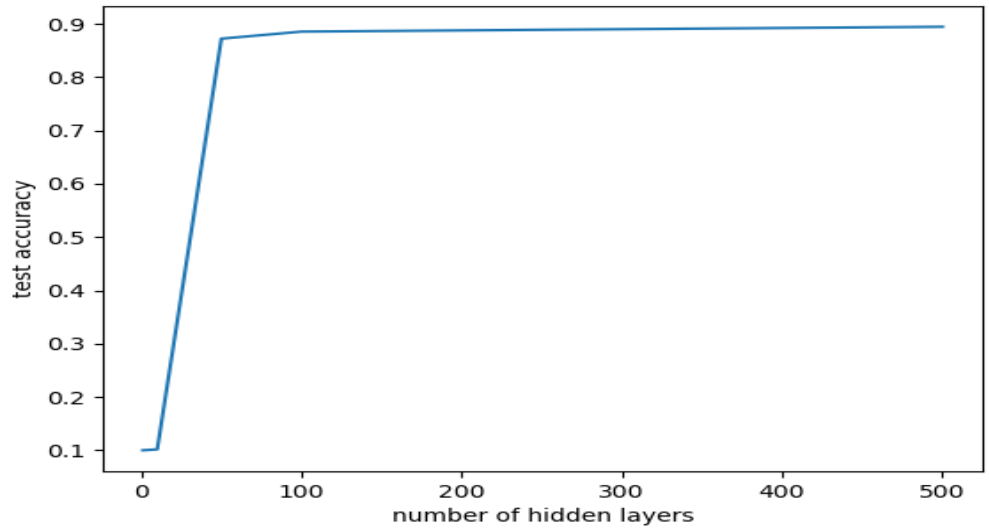
   This is the output I observed for one hidden layer(the accuracies are in decimal(max 1), not in percentage):

Hidden layer	Time Taken(in seconds)	Train Accuracy	Test Accuracy
1	25.784	0.1	0.1
10	9.867	0.1004	0.1017
50	61.541	0.9548	0.8722
100	81.072	0.96595	0.8854
500	323.711	0.9711	0.8945

   These are the corresponding graphs:


time taken to train(in seconds)
300
250
200
150
100
50
0
0
100
200
300
400
500
number of hidden layers


train accuracy
1.0
0.8
0.6
0.4
0.2
0
100
200
300
400
500
number of hidden layers



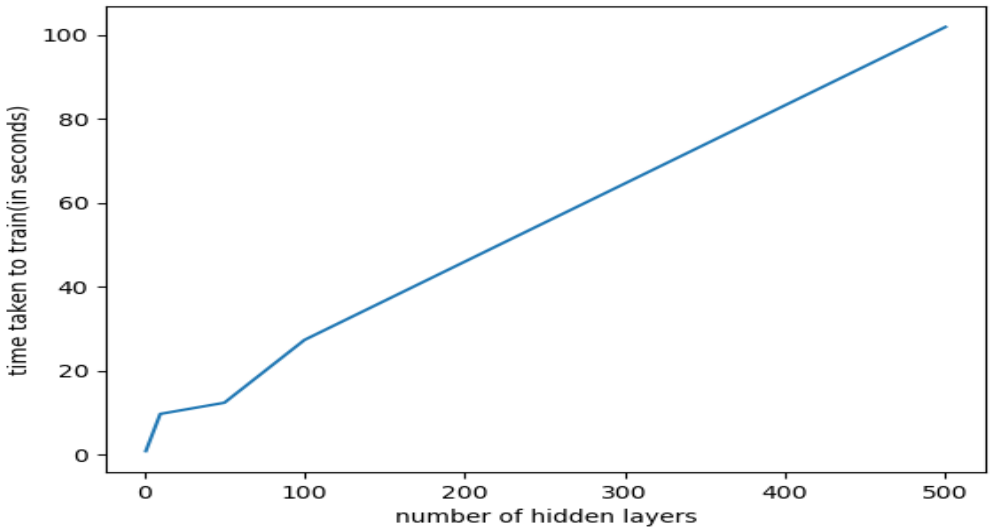
We can see that the time taken to train the model almost linearly increases with the increase in the number of units in the single hidden layer.

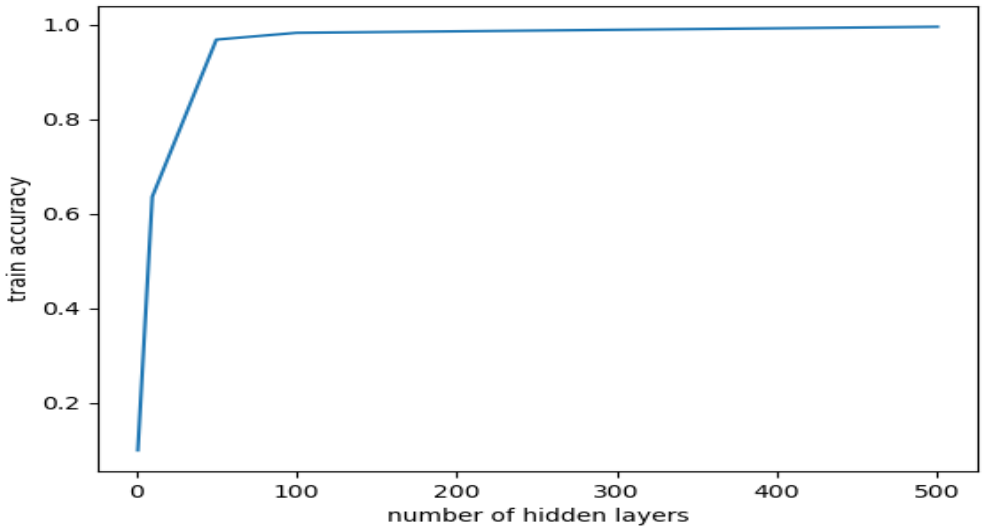
On the other hand, the train and test accuracies almost hit a plateau and there is not a significant increase.

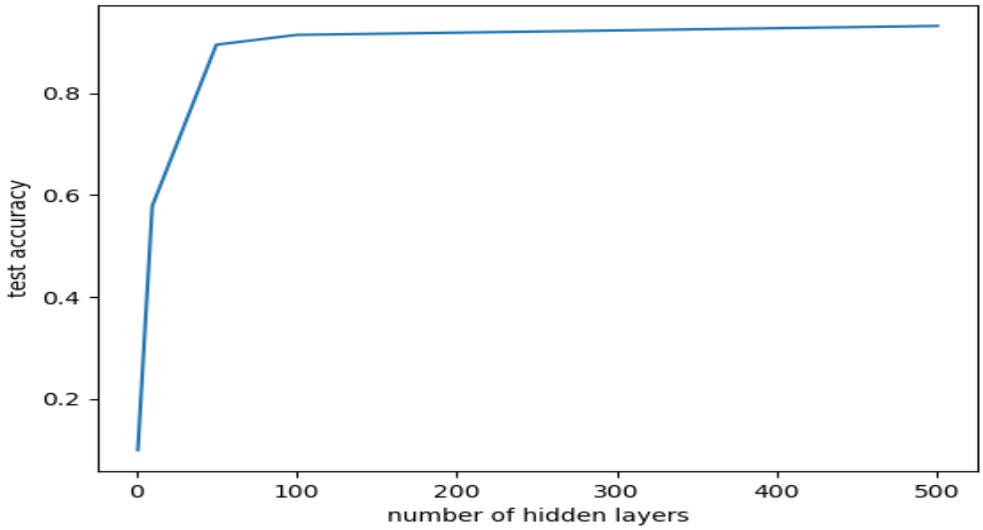
(c) I used an adaptive learning rate with the same architecture as 2(b). The stopping criterion was also the same. This is the output I observed:

Hidden layer	Time Taken(in seconds)	Train Accuracy	Test Accuracy
1	0.7912	0.1	0.1
10	9.670	0.636	0.5785
50	12.331	0.9686	0.8948
100	27.312	0.9829	0.914
500	101.963	0.9956	0.9316

These are the corresponding graphs:







We observe that the time taken to train the model is significantly reduced and also, the corresponding accuracies obtained are higher. Clearly, an adaptive learning rate performs better than static learning rate of 0.001.

(d) We have two kinds of network:

Using sigmoid function for all layers

Using ReLu for hidden layers and sigmoid for output layers

These are the outputs I observed for hidden layers (100,100) and an adaptive learning rate:

Type	Time Taken(in seconds)	Train Accuracy	Test Accuracy
Sigmoid	44.0233	0.9754	0.8938
ReLu	23.969	0.99336	0.9403

We can see that the ReLu model is trained faster and also performs better than the sigmoid model. Comapared to the single hidden layer case with sigmoid, ReLu clearly performs better.

(e) This is the output I got when training using the MLP classifier with solver 'sgd' and hidden layers (100,100):

Time taken: 211.7088 seconds
Train Accuracy: 1.0
Test Accuracy: 0.9181

The model we have designed is better than the MLP classifier since it is trained faster and also gets better test accuracy.