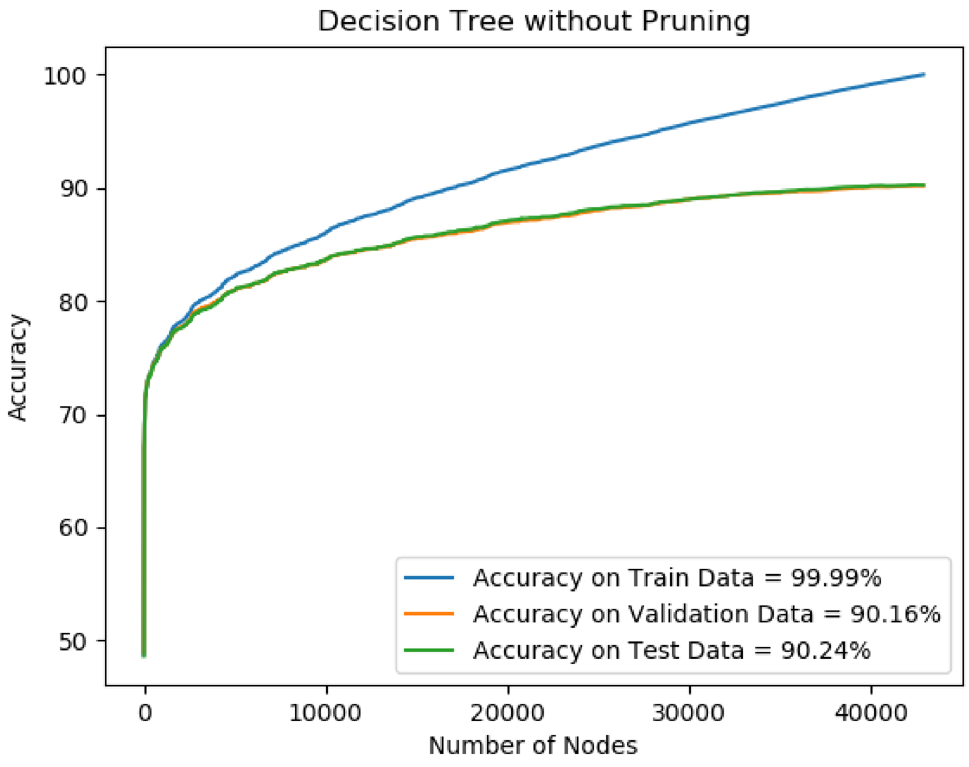
# COL774 Assignment 3

Dipen Kumar (2018CS50098)

December 31, 2020

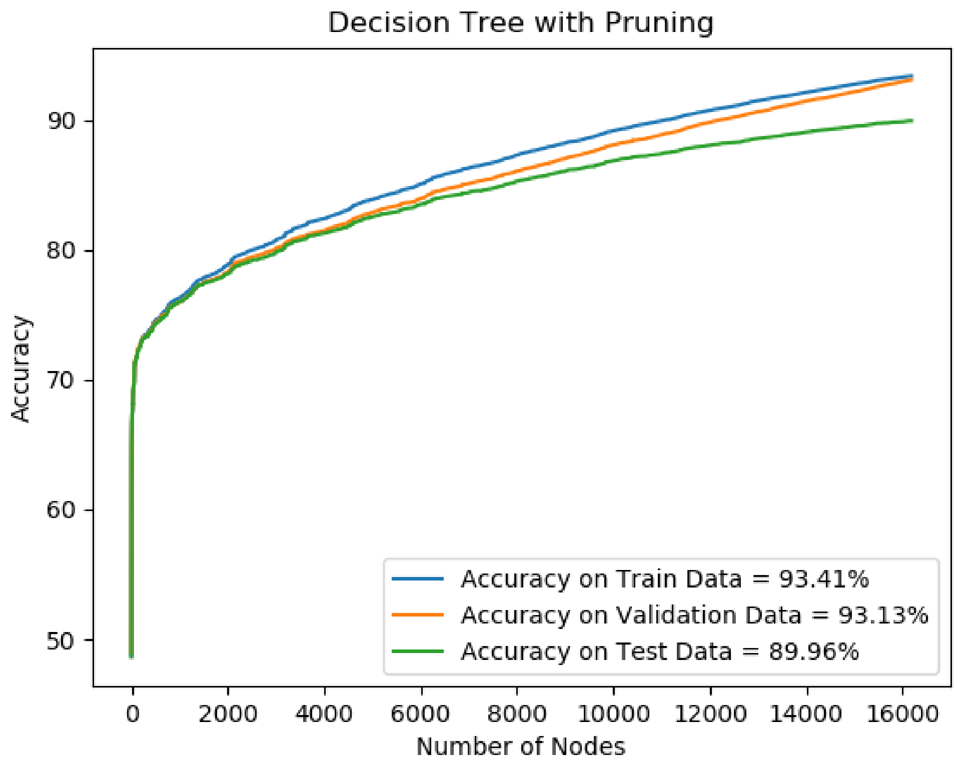
## 1. Decision Tree

1. Constructed a Decision Tree where it was allowed to grow till end where entropy no further decreases. Given below is the plot for the train, validation and test set accuracy against the number of nodes in the tree as we grow the tree.



The accuracy for the training set shows that we have over-fitted our training data in the model and hence we need to do something to make it suited best for the prediction on unknown data. Accuracy on validation and test set were almost same that was close to 90% and for training set it was close to 100%

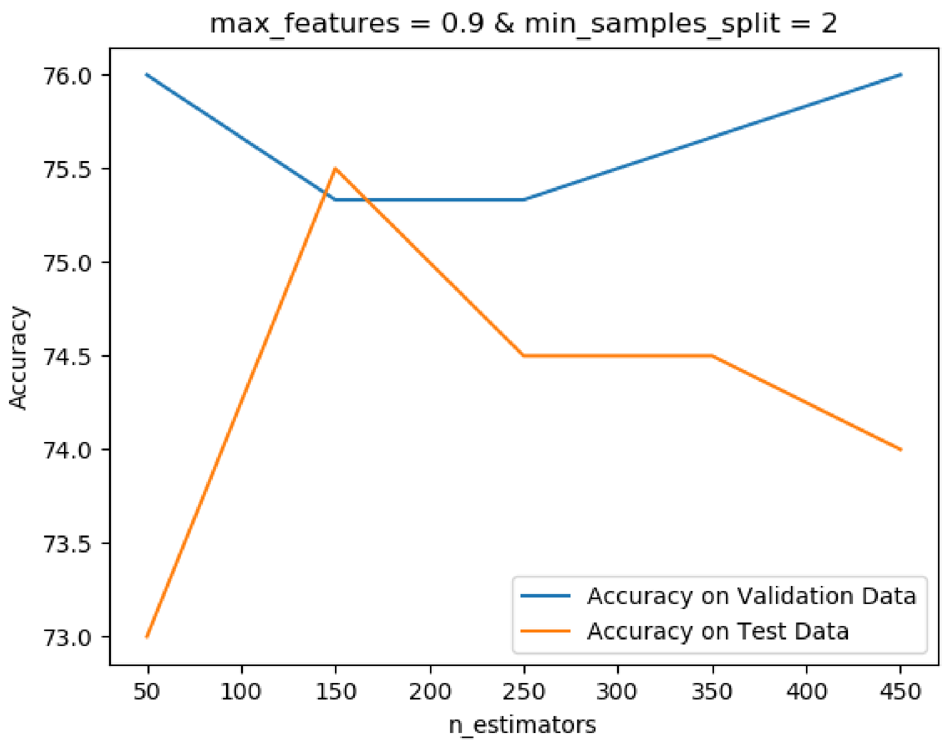
2. Post pruned the tree obtained in step (a) above using the validation set to reduce over-fitting. Given below is the plot for the train, validation and test set accuracy against the number of nodes in the tree.

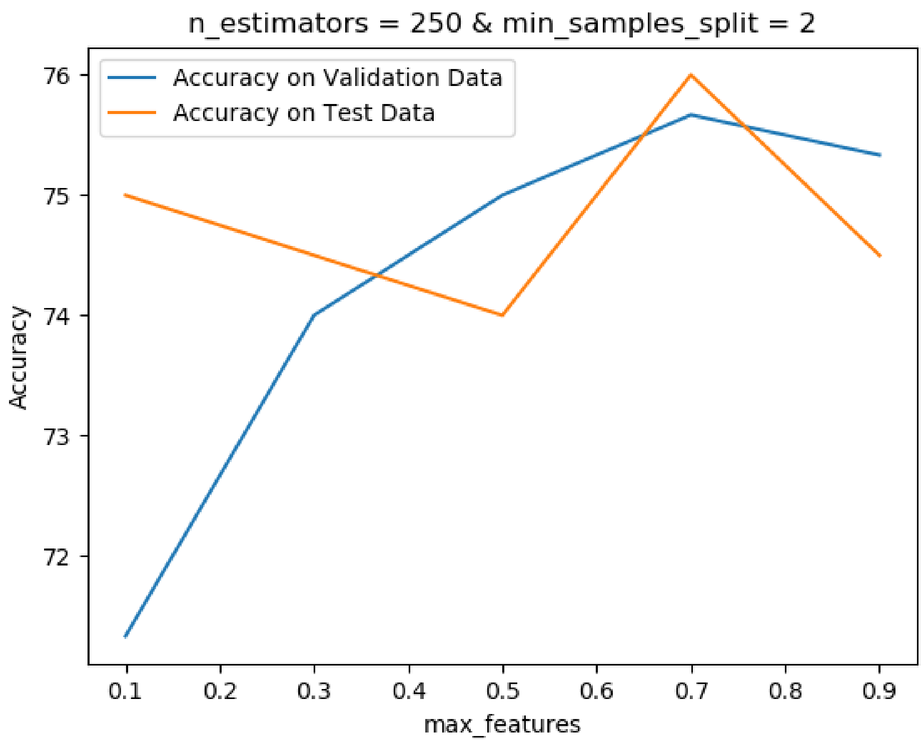


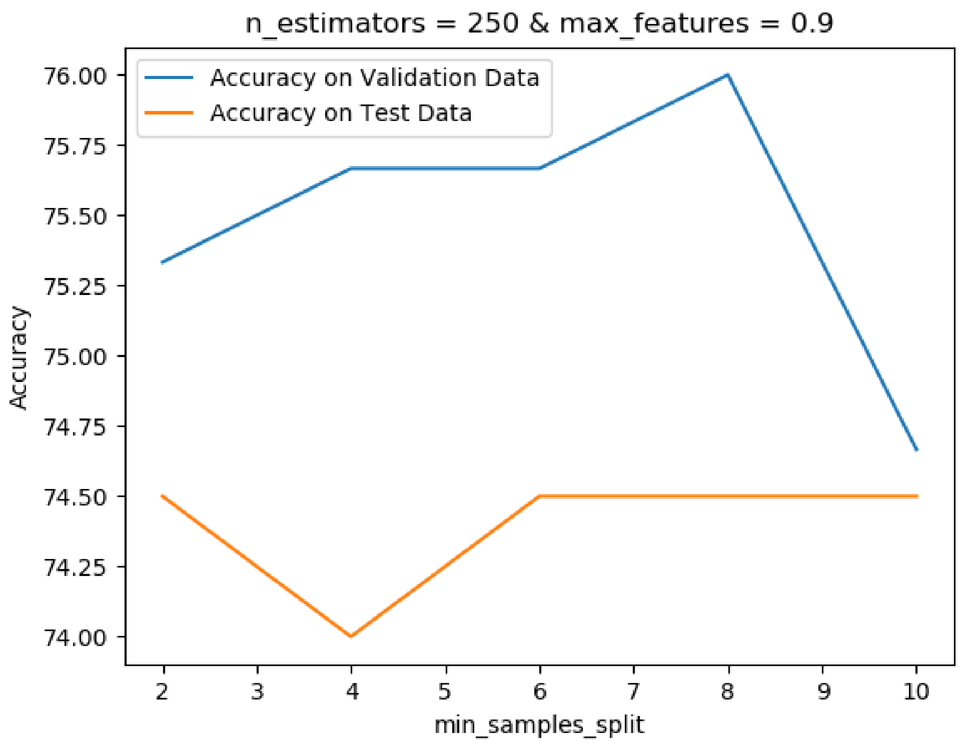
The total number of nodes in the tree went drastically down by almost a factor of 2 and the accuracy on the validation set increased which cost the decrease in the accuracy on the training set which was expected of the algorithm.

3. Different forests were growed by playing around with various parameter values given in question. Used the out-of-bag accuracy to tune to the optimal values for these parameters. Performed a grid search over the space of parameters which was taking huge time time estimated was 10 hrs. I did tried to do on google colab but failed due to maximum RAM limit used. Hence I trimmed train, val, and test data to find optimum values of parameters on first 400 train sample, first 300 val sample, and first 200 test sample. Optimal set of parameters obtained was -
n-estimators = 250
max-features = 0.9
min-samples-split = 2
Accuracy reported was -
Out-Of-Bag Accuracy = 84.25%
Accuracy on Train Data = 100.0%
Accuracy on Validation Data = 75.33%
Accuracy on Test Data = 74.5%
These numbers of Train Data accuracy was more as compared to part (b) obtained after pruning. Here it is 100% on trimmed examples where as there is was 93.41%. Accuracy on Validation Data and Test was was reported low. Here it is around 75% where as there is was 93.13% and 89.96% respectively.

4. Plot with varying parameters to check their sensitivity is plotted below -


max_features = 0.9 & min_samples_split = 2
Accuracy
n_estimators
Accuracy on Validation Data
Accuracy on Test Data
76.0
75.5
75.0
74.5
74.0
73.5
73.0
50
100
150
200
250
300
350
400
450


n_estimators = 250 & min_samples_split = 2
Accuracy
max_features
Accuracy on Validation Data
Accuracy on Test Data
76
75
74
73
72
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9



Increasing n-estimators increase the accuracy but also increases the training time as we grow more trees. we should keep it high as the core processor allow computation fast. Increasing max-features will increase accuracy since each tree learn more using more features but increase it too high will make all trees homogeneous and hence the property of random forest is not used. min-sample-split should be more to avoid over-fitting and less to under-fitting. Hence we have to find optimum values for these using grid search.

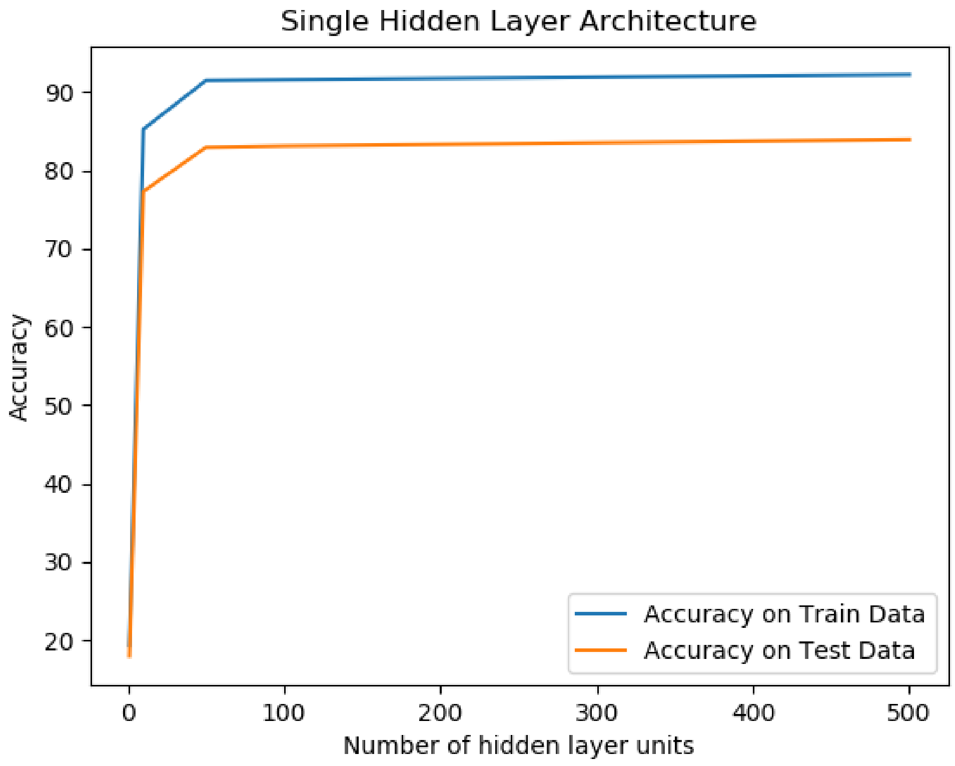
## 2. Neural Network

1. A generic neural network architecture to learn a model for multi-class classification using one-hot encoding was implemented.It used back propagation and mini batch stochastic Gradient Descent to train the network. It used Mean Squared Error as loss function. It uses "sigmoid" as the activation function for the units in output layer and for the hidden layers it uses "ReLU" or "sigmoid" depending on the parameter given to the model. It was implemented from scratch and parameters allowed were- mini batch size, number of features, hidden layer architecture, number of target class, and activation type. It is fully connected architecture.

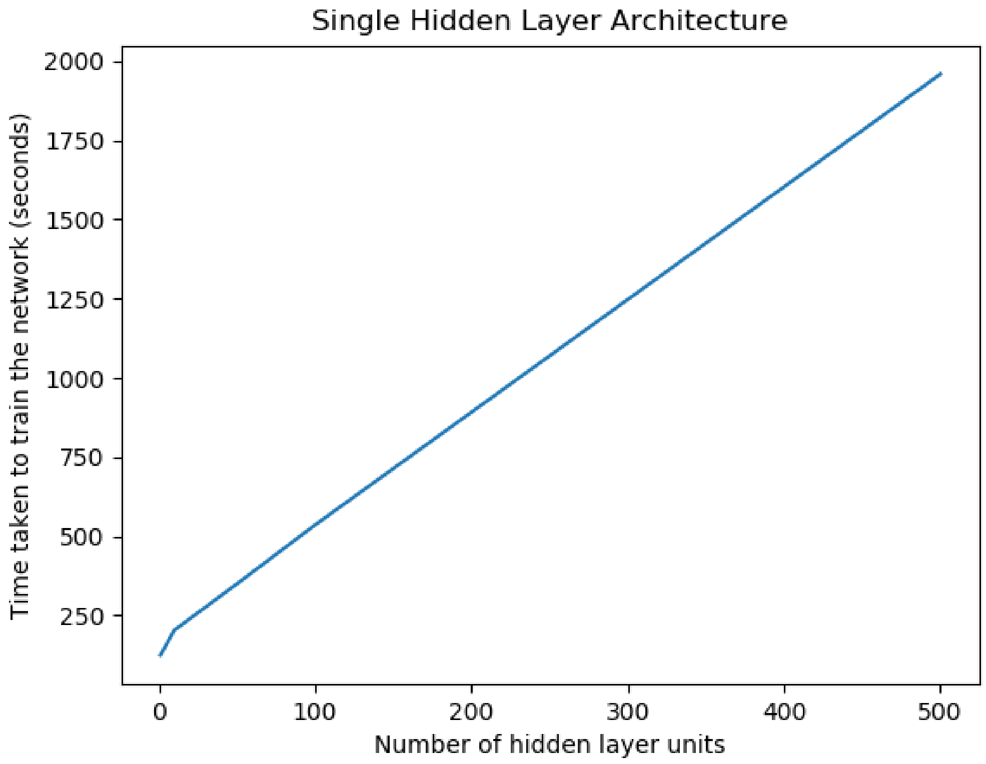
2. Neural Network having a single hidden layer was experimented with varying number of hidden layer units from the set $\{1, 10, 50, 100, 500\}$. Learning Rate $= 0.001$. Mini Batch $= 100$. The training stops when it finds less than 0.0001 change in loss function between two consecutive epochs for 10 consecutive times. It also stops when maximum allowed epochs of 500 is reached.

| Number of hidden layer units | Accuracy on Train Data | Accuracy on Test Data | Time taken to train the network |
|---|---|---|---|
| 1 | 19.32 % | 17.93 % | 2 min 5 sec |
| 10 | 85.27 % | 77.31 % | 3 min 24 sec |
| 50 | 91.49 % | 82.95 % | 5 min 50 sec |
| 100 | 91.59 % | 83.12 % | 8 min 56 sec |
| 500 | 92.23 % | 83.94 % | 32 min 39 sec |

Accuracy on train data set is always found high than train because train data set was used to train the network and may have over-fitted it whereas test data was new unseen data to test.



As we increases the number of hidden layer units, we get better performance in terms of accuracy and saturates because more parameters means more learning and learning has limit of 100 % accuracy.



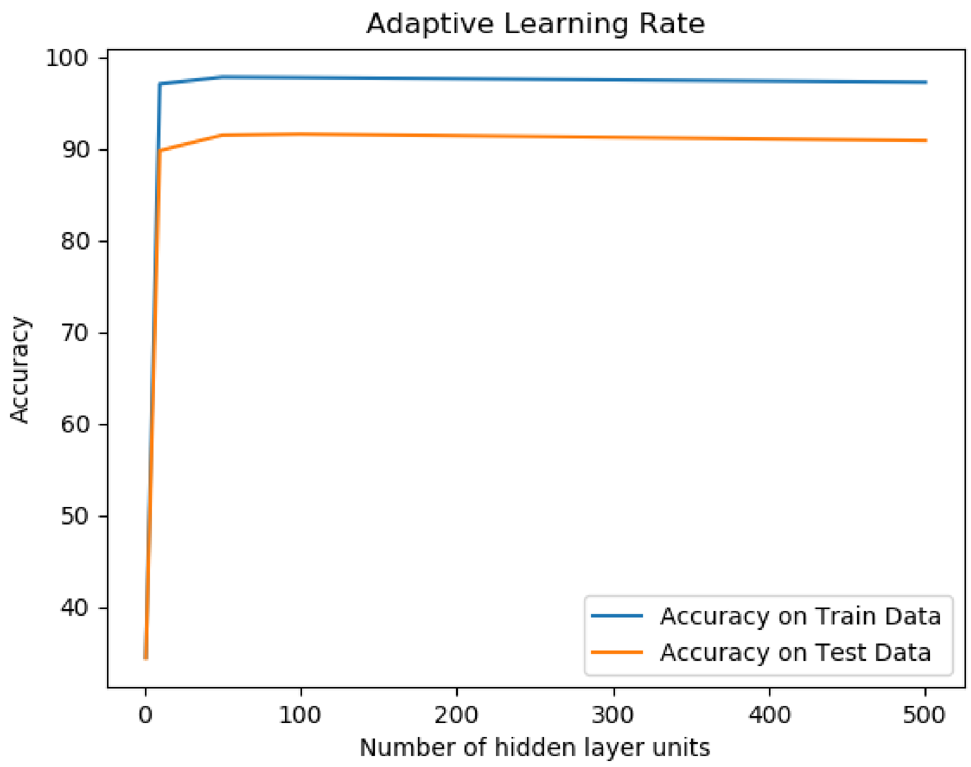
As we increases the number of hidden layer units, time taken by the model to train also increases because parameters increases and computation time for more parameters also increases.

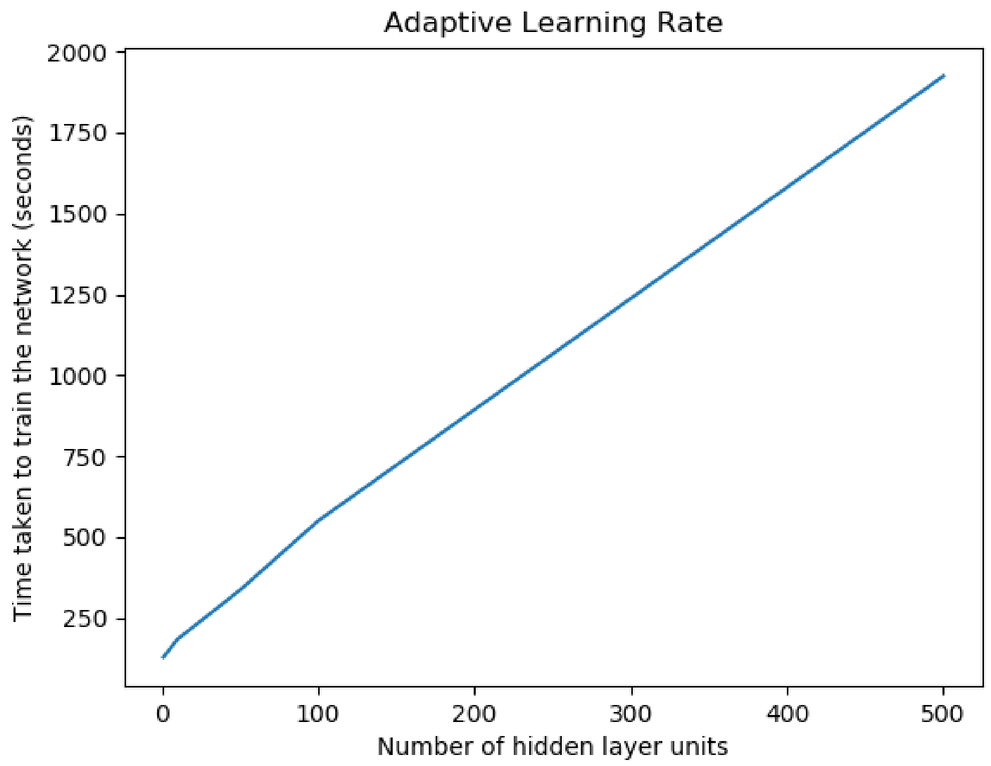
3. In this part learning rate was changed to an adaptive learning rate inversely proportional to number of epochs.

$$\eta = \frac{\eta_0}{\sqrt{e}}$$

where $\eta_0 = 0.5$ is the seed value and $e$ is the current epoch number. Mini batch and stopping criterion was kept. Mini batch size = 100. The training stops when it finds less than 0.0001 change in loss function between two consecutive epochs for 10 consecutive times. It also stops when maximum allowed epochs of 500 is reached. Number of hidden layer units for single hidden layer architecture was varied and the train/test set accuracy, as well as training time was measured and plotted.

| Number of hidden layer units | Accuracy on Train Data | Accuracy on Test Data | Time taken to train the network |
|---|---|---|---|
| 1 | 34.44 % | 34.37 % | 2 min 9 sec |
| 10 | 97.09 % | 89.81 % | 3 min 4 sec |
| 50 | 97.83 % | 91.5 % | 5 min 36 sec |
| 100 | 97.78 % | 91.62 % | 9 min 10 sec |
| 500 | 97.28 % | 90.9 % | 32 min 4 sec |





The performance in terms of accuracy was increased with respect to part (b) because since we started fast we were able to reach very close to convergence and to the optimum parameters with in 500 epochs where as in part (b) it has not converged and optimum parameter was yet to determined but program stopped as maximum epochs was reached. The adaptive learning rate made the training faster as we started fast and as we reached the optimum parameters we went slow to learn more accurate. Training time taken by part (b) and this part was close when it was not converged and stopped by maximum epochs because time taken by each epoch is same as is determined by size of the training data but this part was learning faster because in same time adaptive learning rate found more optimum parameters and hence accuracy was increased.

4. Implemented a network with 2 hidden layers with 100 units each. Experimented with both "ReLU" and "sigmoid" activation units. Used the adaptive learning rate.

| Activation units | Accuracy on Train Data | Accuracy on Test Data | Time taken to train the network |
|---|---|---|---|
| "sigmoid" | 97.89 % | 91.54 % | 12 min 8 sec |
| "ReLU" | 99.73 % | 94.33 % | 11 min 57 sec |

Train and Test set accuracy obtained using "ReLU" was better than "sigmoid". "ReLU" was also faster than sigmoid. Hence "ReLU" performed better. This results of "sigmoid" with two hidden layer each with 100 units when compared with results in part (b) using a single hidden layer with "sigmoid" was better and was comparable to part (c) which used adaptive learning rate. "ReLU" performed better than "sigmoid" from all parts- (b), (c), and this part.

5. Training using existing library and modified loss function of MLPClassifier from scikit-learn library was taking approx same time compared to my own implementation. It took 11 min 44 sec to train where as my implementation using same architecture- [100 100], using same activation unit- "ReLU" and using same solver- Stochastic Gradient Descent took 11 min 57 sec. Its accuracy on Train Data was 99.84% and Test Data was 91.58% where as I have achieved as high as 99.73% on Train Data and 94.33% on Test Data. Both of us used same maximum epoch allowed = 500. The different thing was loss function. MLPClassifier only allows for Cross Entropy Loss (log-loss function) over the final network output. I used Mean Squared Error as log function.