COL774 Assignment 3

Aniruddha Deb
2020CS10869

October 2022

1 Decision Trees, Random Forests, Gradient Boosted Trees

1.1 Dataset 1

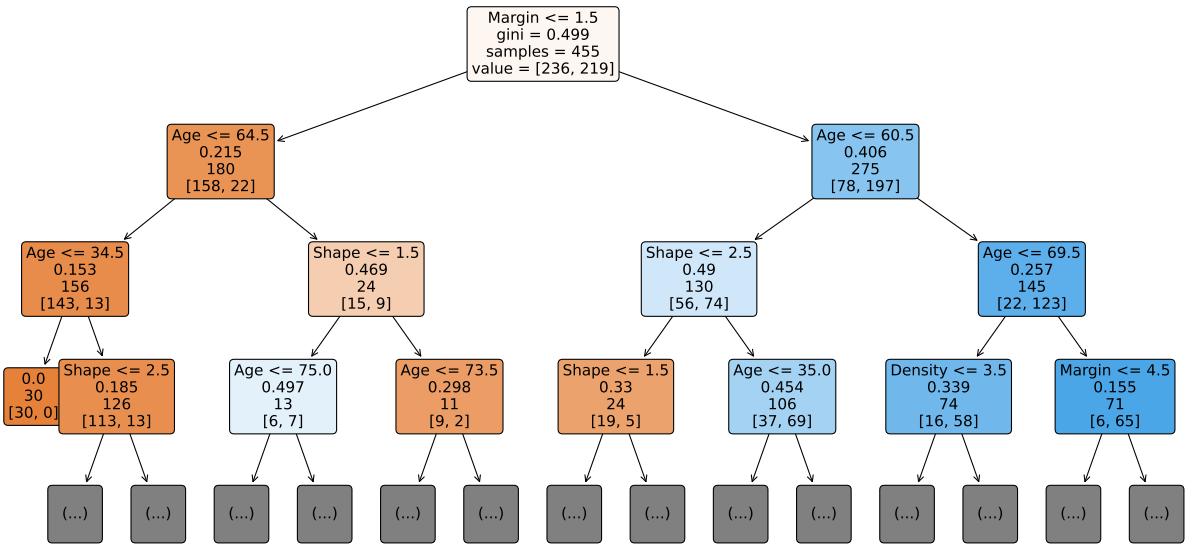
(a) After removing the missing values, we obtain the following accuracies:

(i) **Training set:** 92.53 %

(ii) **Validation set:** 76.03 %

(iii) **Test set:** 69.17 %

The decision tree we obtain is as follows (drawn upto a depth of 3):



(b) With GridSearchCV and the grid of parameters max_depth $= \{4, 6, 8, 10\}$, min_samples_split $= \{2, 3, 4, 5\}$ and min_samples_leaf $= \{1, 2, 3, 4, 5\}$, the best parameters (obtained via five-fold cross validation) are:

(i) max_depth $= 4$

(ii) min_samples_leaf $= 5$

(iii) min_samples_split $= 2$

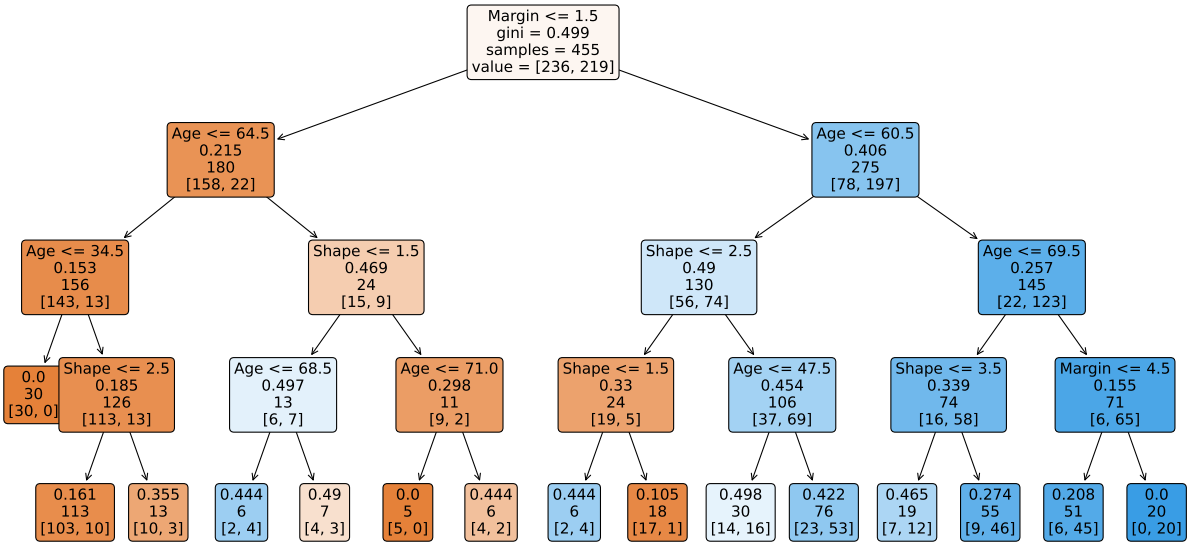
With them, we obtain the following accuracies:

(i) **Training set:** 81.98 %

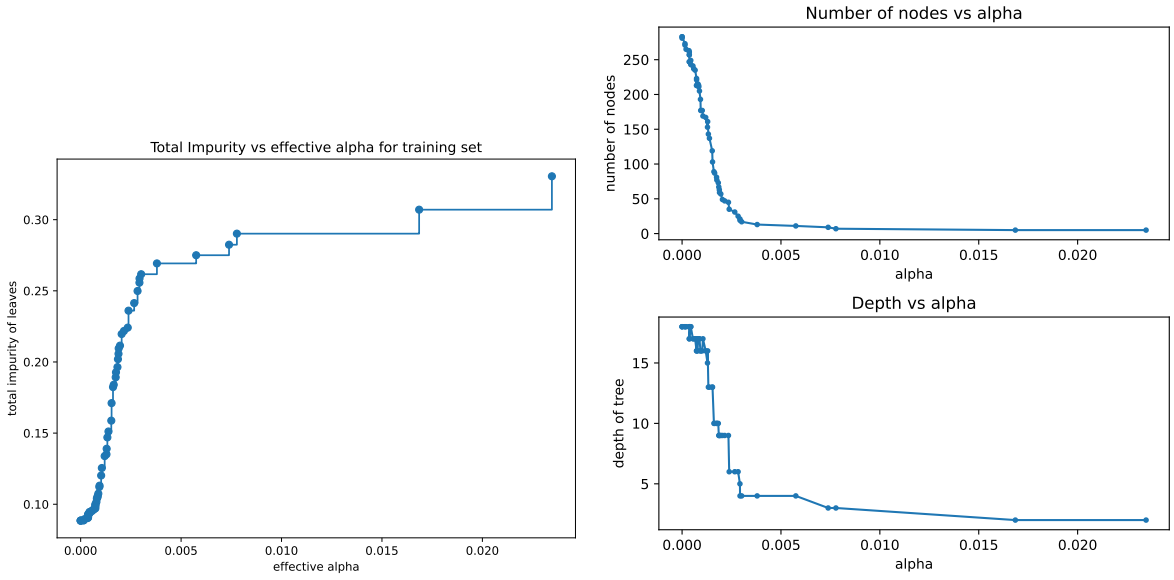
(ii) **Validation set:** 87.6 %

(iii) **Test set:** 75.1 %

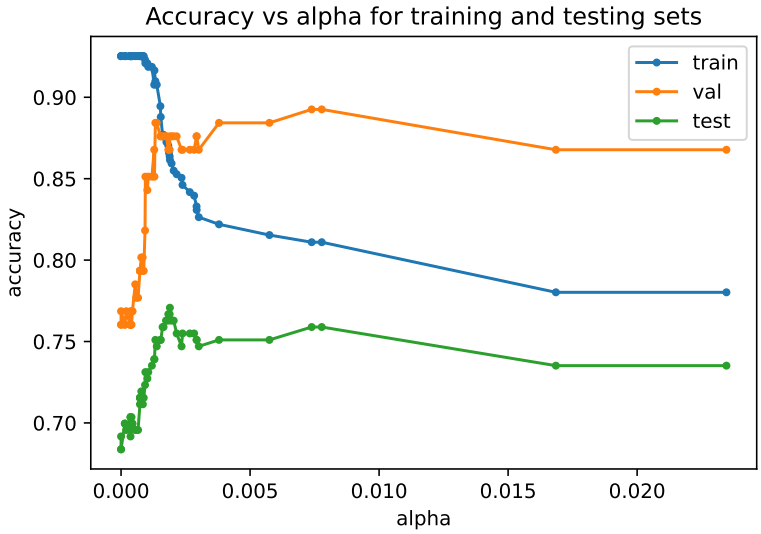
The decision tree has a much shallower depth than the previous tree, and is also simpler



(c) The total impurity vs the alphas is plotted below, along with the depth and the number of nodes v/s alpha



We see that the training accuracy is very high for low values of alpha, at the expense of the validation and test accuracy i.e. the model overfits for low values of alpha. The best trees are the ones with fewer nodes



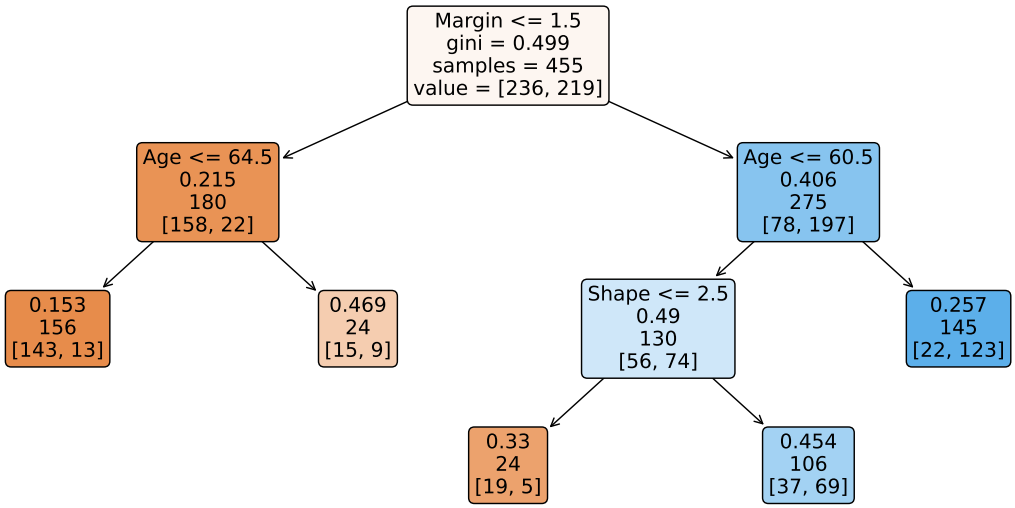
The accuracies we obtain for the various datasets, using the best pruned tree are:

(a) **Training set:** 81.1 %

(b) **Validation set:** 89.26 %

(c) **Test set:** 75.89 %

The best pruned tree, as discussed before, has only 9 nodes compared to the trees we obtained in part (a) and part (b).



(d) With GridSearchCV and the grid of parameters n_estimators $= \{50, 100, 150, 200\}$, max_features $= \{1, 2, 3, 4\}$ and min_samples_split $= \{2, 3, 4, 5\}$, the best parameters (obtained using out of bag score as the metric) are:

(i) n_estimators $= 200$

(ii) min_samples_split $= 5$

(iii) max_features $= 3$

With them, we obtain the following accuracies:

(i) **Training set:** 90.11 %

(ii) **Out of Bag set:** 76.26 %

(iii) **Validation set:** 86.78 %

(iv) **Test set:** 76.68 %

(e) The results are summarized in the following table:

Default decision tree:	Median	Mode
Train	91.81 %	90.69 %
Validation	74.07 %	75.56 %
Test	74.31 %	71.18 %

Grid Searched Decision tree:	Median	Mode
max_depth	4	4
min_samples_split	2	2
min_samples_leaf	3	2
Train	81.56 %	81.38 %
Validation	87.41 %	86.67 %
Test	80.90 %	77.43 %

ccp_alpha optimized Decision tree:	Median	Mode
n_nodes	11	137
ccp_alpha	0.0055	0.0012
Train	80.26 %	88.83 %
Validation	87.41 %	88.15 %
Test	79.17 %	76.04 %

Random Forest classifier:	Median	Mode
n_estimators	200	100
min_samples_split	5	5
max_features	3	2
Train	88.45 %	88.08 %
Out of Bag	75.23 %	73.37 %
Validation	83.70 %	83.70 %
Test	77.78 %	77.43 %

The imputed tree is marginally better (2-3 %) compared to the tree which ignores missing values. Note that the comparision is not exact, as the size of the validation and test sets are also changing as a consequence (while ignoring missing values, we also ignore the ones in the test and validation set). As a result, the actual increase in accuracy post imputation will be much more than that without imputation.

(f) After implementing an XGBoost classifier and searching for parameters using GridSearchCV in the given parameter space, we obtain the following best parameters:

(i) max_depth = 10

(ii) n_estimators = 10

(iii) subsample = 0.5

With them, we obtain the following accuracies:

(i) **Training set:** 83.61 %

(ii) **Validation set:** 84.44 %

(iii) **Test set:** 77.08 %

1.2 Dataset 2

NOTE: This dataset was not completely evaluated due to the large running time of grid search on the dataset

(a) Fitting a decision tree on the entire dataset gave us the following accuracies:

(i) **Training set:** 100 %

(ii) **Validation set:** 58.49 %

(iii) **Test set:** 57.74 %

(b) Grid search was run on the following parameter space:

(i) `max_depth`: $\{10, 15, 20, 25, 30\}$

(ii) `min_samples_split`: $\{2, 3, 4\}$

(iii) `min_samples_leaf`: $\{1, 2, 3\}$

The best parameters were:

(i) `max_depth`: 30

(ii) `min_samples_split`: 2

(iii) `min_samples_leaf`: 1

The accuracies for these parameters are:

(i) **Training set:** 63.17 %

(ii) **Validation set:** 44.30 %

(iii) **Test set:** 44.01 %

The kernel timed out after this point, which is why no more statistics were obtained.

2 Neural Network

(a) The neural network was implemented in Numpy, in a purely functional manner (so it can support any activation, loss and learning rate function passed to it). Forward propagation is given by the following equations:

$$z_i = l_{i-1} \times w_i + b_i$$
$$l_i = \text{act}(z_i)$$

where l_i is the output of the ith layer, z_i is the linear output of the perceptrons and act is the activation function. Note that we use $\times$ to denote matrix multiplication and $\odot$ to denote Hadamard (elementwise) multiplication.

Backward propagation is given by the following equations:

$$\frac{\partial J}{\partial l_L} = \text{loss'}(\theta)$$

$$\begin{aligned}\frac{\partial J}{\partial w_i} &= \frac{\partial J}{\partial l_i}\frac{\partial l_i}{\partial z_i}\frac{\partial z_i}{\partial w_i} \\ &= l_{i-1}^T \times \left(\frac{\partial J}{\partial l_i} \odot \text{act'}(z_i)\right)\end{aligned}$$

$$\begin{aligned}\frac{\partial J}{\partial b_i} &= \frac{\partial J}{\partial l_i}\frac{\partial l_i}{\partial z_i}\frac{\partial z_i}{\partial b_i} \\ &= \frac{\partial J}{\partial l_i} \odot \text{act'}(z_i)\end{aligned}$$

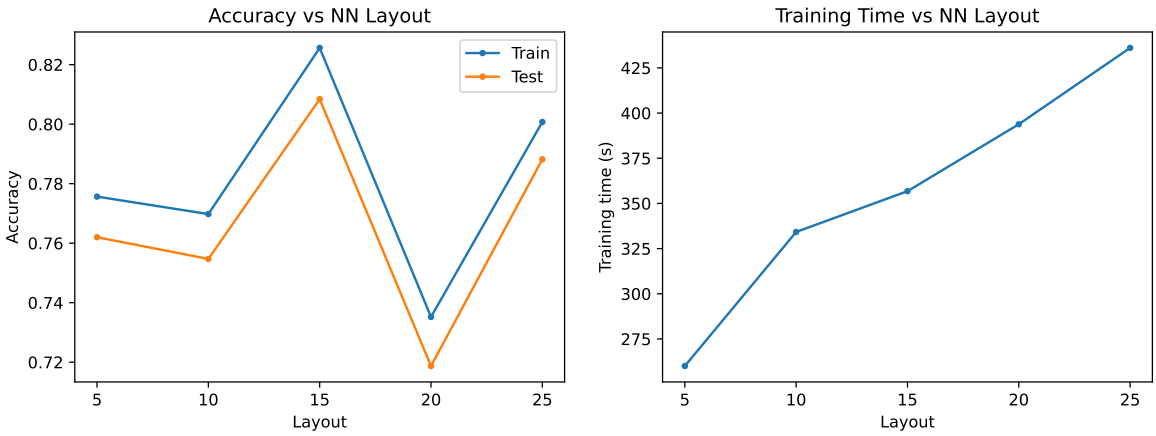
$$\begin{aligned}\frac{\partial J}{\partial l_{i-1}} &= \frac{\partial J}{\partial l_i}\frac{\partial l_i}{\partial z_i}\frac{\partial z_i}{\partial l_i} \\ &= \left(\frac{\partial J}{\partial l_i} \odot \text{act'}(z_i)\right) \times w_i^T\end{aligned}$$

The matrix initialization was done according to He et al., wherein the matrices w_i and b_i were initialized using a zero-mean normal distribution with variance $\sqrt{2/n_i}$, where n_l is the number of neurons in the ith layer.

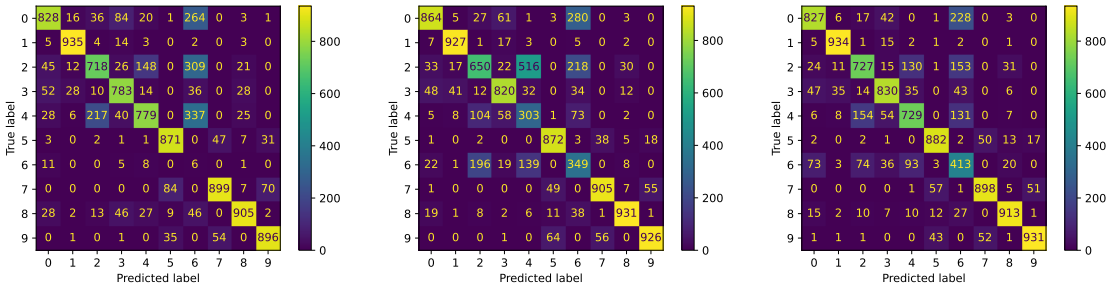
(b) The network was trained with the stopping criteria that the % change in loss between two consecutive epochs, after training for atleast 50 epochs, is less than 10^{-4}. The network also has an upper bound of 300 epochs for training.

The accuracies the trained model obtains on the test dataset, with varying hidden layer sizes, are:

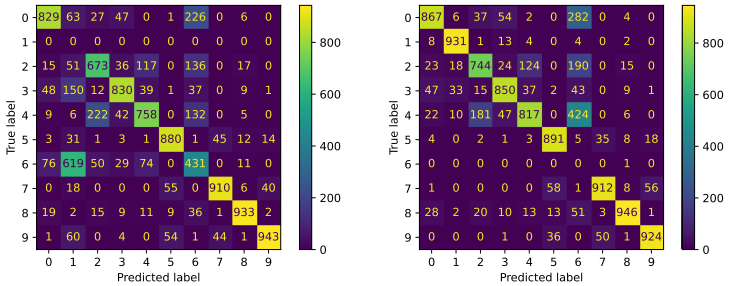
Neurons	Test Accuracy	Train Accuracy
5	76.1 %	77.7 %
10	75.8 %	77.1 %
15	80.2 %	82.3 %
20	71.9 %	73.9 %
25	78.7 %	79.9 %



The confusion matrices are:



5 10 15



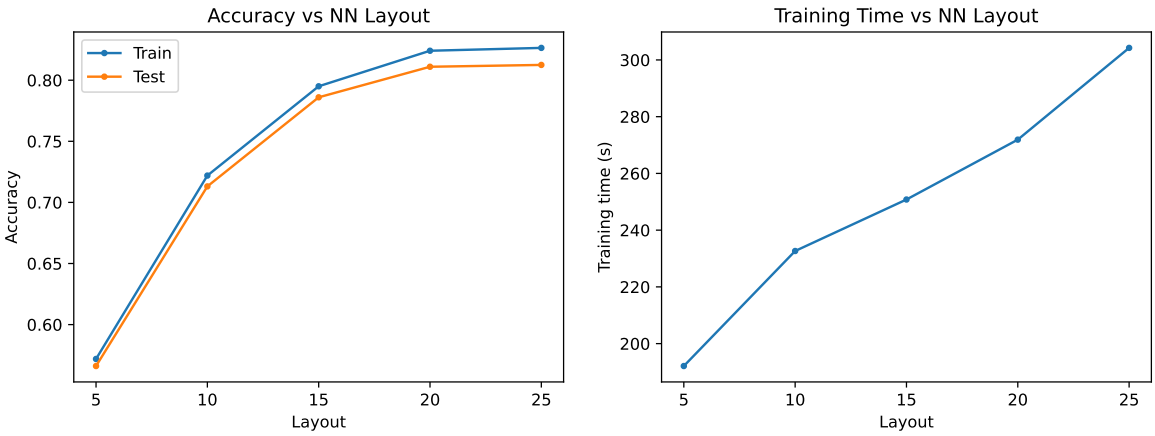
20 25

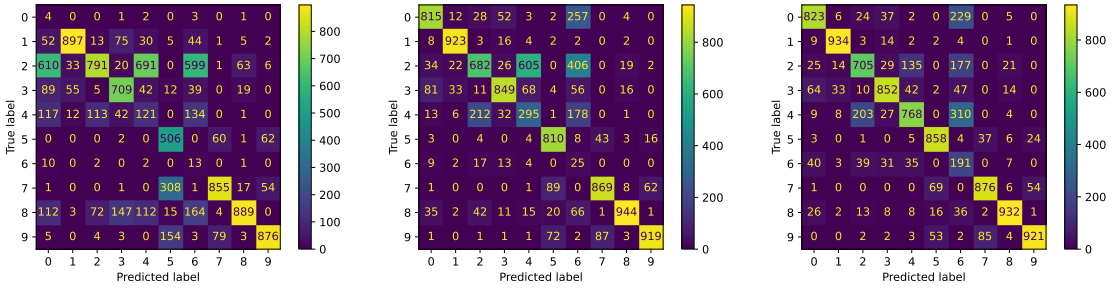
We observe that models with fewer number of neurons tend to marginally underfit, and that there is a dip for models with an even number of neurons in the hidden layer (a zigzag pattern is observed). The training time, as expected, increases with an increase in the number of neurons.

Also, note that 20 and 25 may not have completely learnt the model: they do not classify a label correctly (class 1 in the 20-neuron network and class 6 in the 25-neuron network). They may give better results if we run the network for a longer period of time with a different convergence criterion.

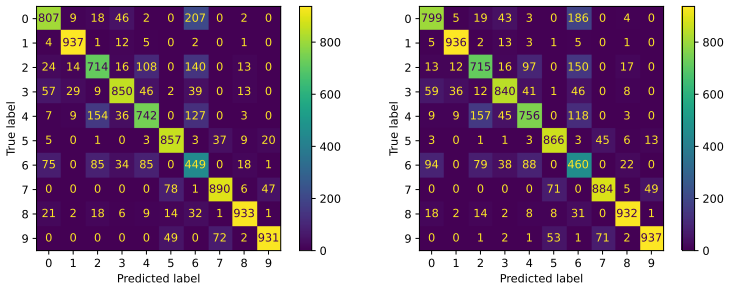
(c) The stopping condition would now change, as our step size (and consequently change in loss) would decrease with increasing epochs. Changing the stopping condition to $10^{-4}/\sqrt{e}$, where e is the current epoch, gave the following results:

Neurons	Test Accuracy	Train Accuracy
5	56.2 %	56.4 %
10	71.1 %	71.9 %
15	78.5 %	79.6 %
20	80.3 %	82.3 %
25	80.4 %	82.5 %





5 10 15

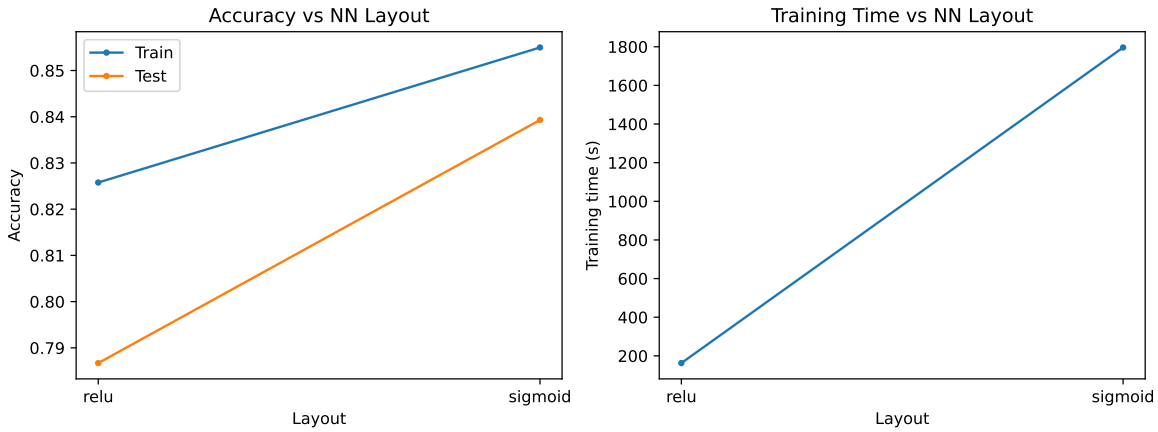


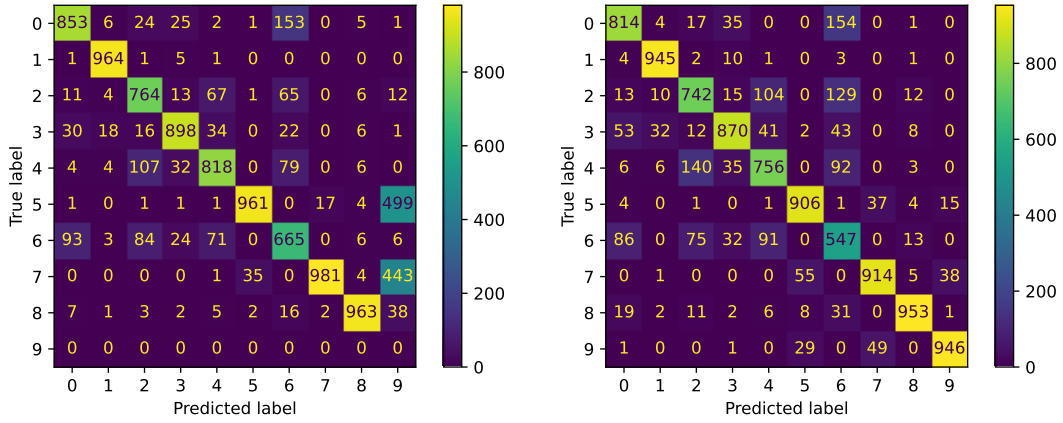
20 25

The Epoch LR is a slow learner, and in none of the cases above did it converge before the epoch limit (200) was hit. In terms of training time, it trained faster than the previous network, even though the epoch limit was hit.

(d) With a 2x100 network as described in the assignment, we obtain the following results:

Network	Test Accuracy	Train Accuracy
relu	78.8 %	82.5 %
sigmoid	83.4 %	85.3 %





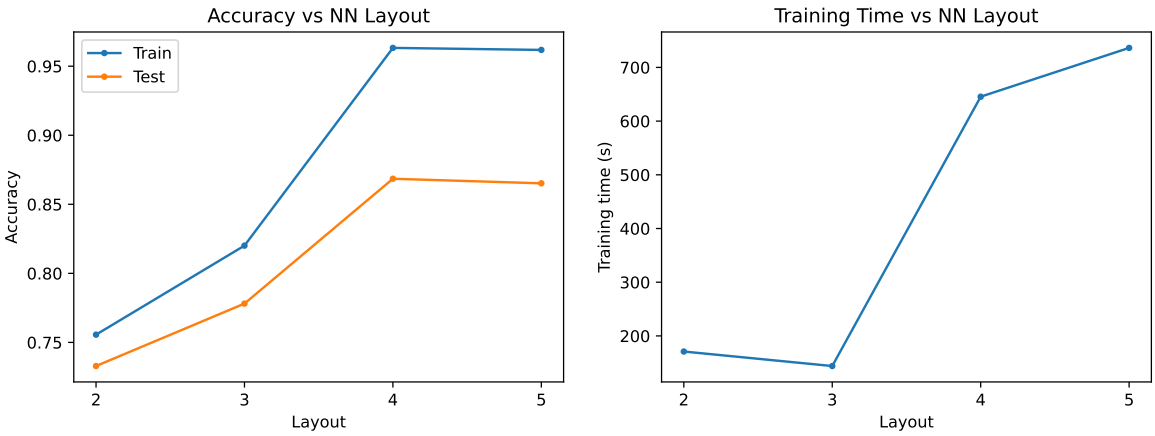
relu sigmoid

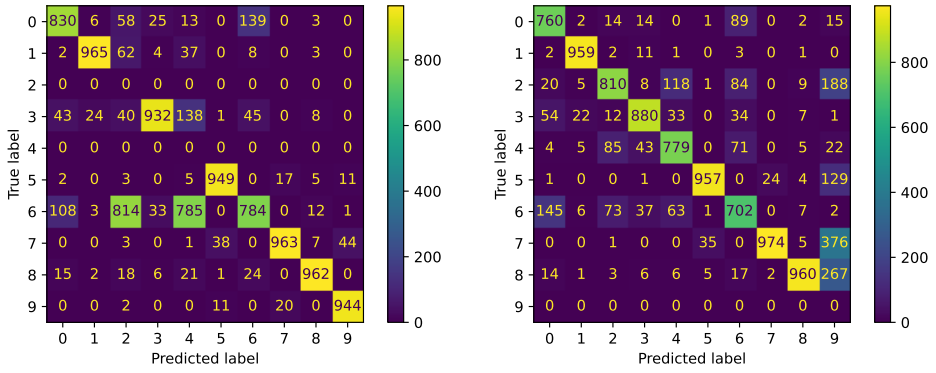
When both were trained with a fixed number of epochs, the ReLU network performed better. This implies that like before, the variable learning rate is not converging to the local minima quickly. The sigmoid performs better as it is 'easier' to learn than the relu: the derivatives do not go immediately to zero, and as a result, the net gradient would be larger than the one obtained from ReLU and this would allow it to take larger steps with the adaptive learning rate.

The results are better than the ones in 2(b): This is to be expected, as more neurons allow the network to learn a larger number of hypotheses.

(e) With increasing the number of layers, the accuracy increases until it saturates at 86% for the test set. The training time graph also indicates that the networks with 2 and 3 hidden layers were inadequately trained and hence underfit the data.

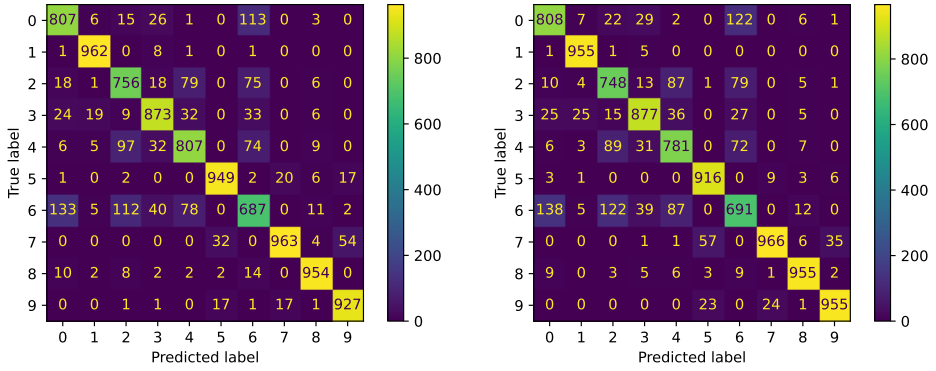
Layers	Test Accuracy	Train Accuracy
2	73.5 %	75.2 %
3	77.2 %	82.3 %
4	86.8 %	96.1 %
5	86.7 %	96.0 %





2

3



4

5

The *best* network was found to be a 1x100 network with ReLU activations, trained using a constant learning rate. This gave an accuracy of **88.29 %** on the test dataset and **93.61 %** on the train dataset.

(f) Cross-Entropy loss for a batch is given by

$$\mathcal{J}^b(\theta) = -\frac{1}{M}\sum_{i=1}^{M}\sum_{j=1}^{10} y_j^{(i)} \log o_j^{(i)} + (1 - y_j^{(i)}) \log\left(1 - o_j^{(i)}\right)$$

The derivative of this, with respect to o is

$$\frac{\partial \mathcal{J}^b(\theta)}{\partial o}\Big|_{ij} = -\frac{1}{M}\left(\frac{y_j^{(i)}}{o_j^{(i)}} - \frac{1 - y_j^{(i)}}{1 - o_j^{(i)}}\right)$$

After training the best network (1x100) for 397 s using cross-entropy loss, we obtained a training set accuracy of **95.74 %** and a test set accuracy of **87.07 %**.

(g) Using the Scikit-Learn MLPClassifier to train a network with the same hyperparameters as the best network we had, we obtain a training set accuracy of **90.36 %** and a test set accuracy of **87.28 %**. The training occurs in 370.87 seconds and takes 200 iterations (the same as our classifier).

The library implementation does better than the model in 2(f), both of which use Cross Entropy loss as the minimization metric. It also trains slightly faster (27 seconds faster).

3 Appendix

The libraries used for this assignment were:

1. numpy
2. pandas
3. matplotlib
4. scikit-learn
5. nltk
6. scipy
7. XGBoost