COL774: Assignment 3 Report

Aniruddha Chatterjee

2021MT10896

Q1. Decision Trees and Random Forests

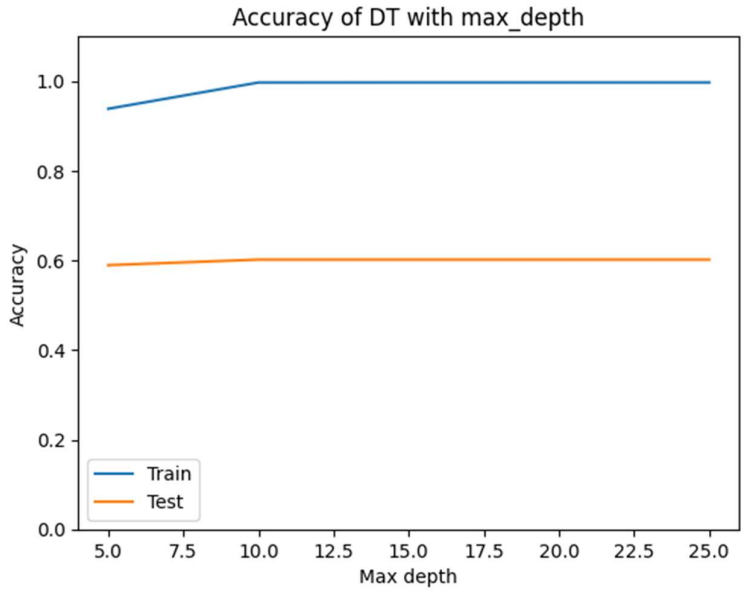
(a) The initial Decision Tree splits our data into k-nodes for categorical features with 'k' unique values, and 2 ranges for continuous/numerical data (split about the median).

On a trial tree with depth=10, we obtain accuracy:
Train = 99.70614539414846% Test = 60.18614270941055%

When making simple guesses we obtain these scores:

	Only-win prediction	Only-loss prediction
Train	50.3385 %	49.6380 %
Test	49.6614 %	50.3619 %

Upon varying the max depth of leaf nodes in the tree, the accuracies obtained are as:



Max Depth	5	10	15	20	25
Test Acc.	0.589451	0.601861	0.601861	0.601861	0.601861

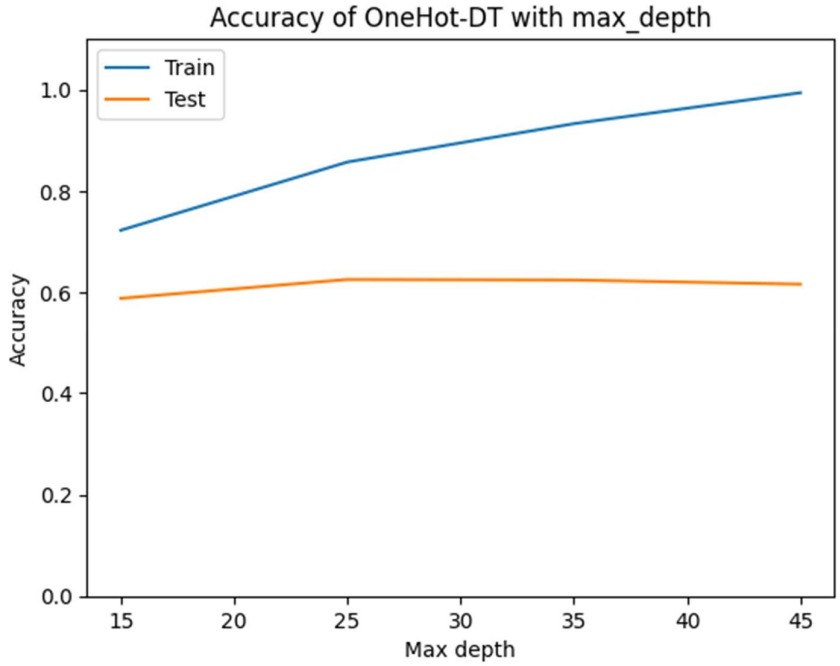
It seems that our current trees overfit to the training data, given the large train-test gap. Although, the test accuracy stays unaffected probably because our model didn't pick up actual trends to a fuller extent, and only memorized the training set.

But there are still improvements over the above naïve predictions, i.e. test accuracy being above the halfway ballpark.

(b) Here, a custom-defined function is used for one-hot encoding, performs the same task as the scikit function (assuming we had to use our own implementation).

Now, all the categorical labels with 'k' values have been separated into k different categorical columns with boolean 0, 1 values.

With this, we obtain the following results for depths = [15,25,35,45]:
(The depths are higher than in (a), maybe since one-hot reduced nodes at one depth)

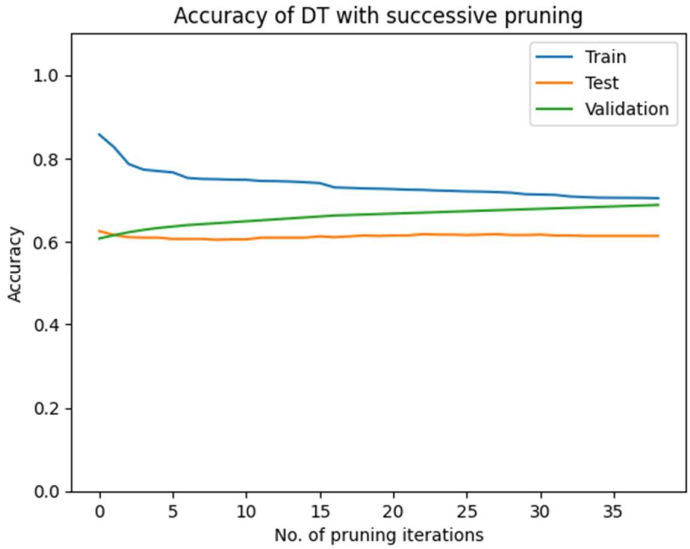


Max Depth	15	25	35	45
Test Acc.	0.588417	0.625646	0.624612	0.616339

From the graph, there seems to be a small improvement on test accuracy upon using one-hot encoded data. The train curve also seems to max out gradually.
There is a slight degree of overfitting beyond max depth of 25 as the test score falls off.

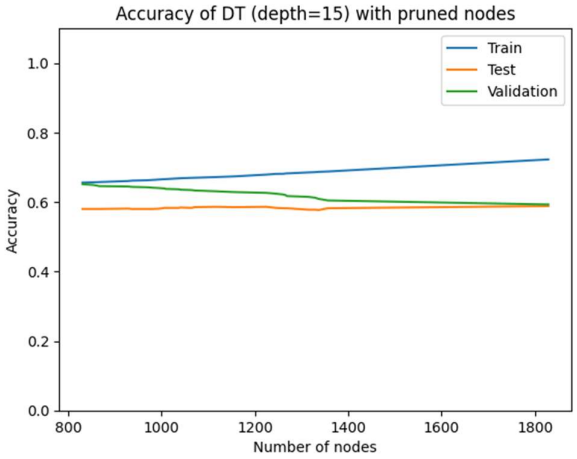
(c) We use the post-pruning approach to tackle the overfitting problem. Our max depth limit is already a sort of pre-prune method deployed.
The approach here is to use a validation set and parse all the nodes in our tree, finding the one which when pruned gives best scores on this X-val, i.e. that node is converted to a leaf(removed its subtree) returning predictions. We repeat this process till there is no further improvement on any of the remaining nodes.

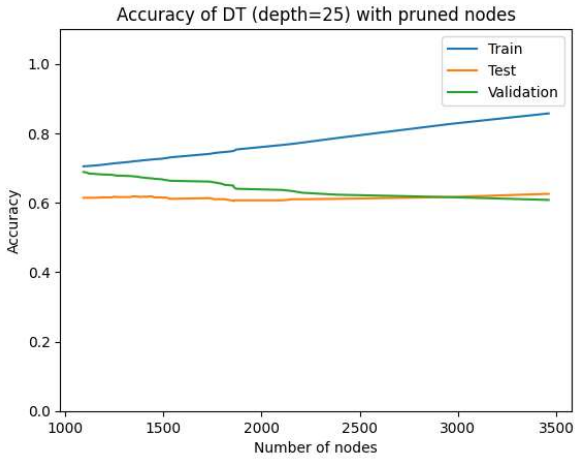
On our trial post-prune tree of depth=25, this is the result of successive pruning:

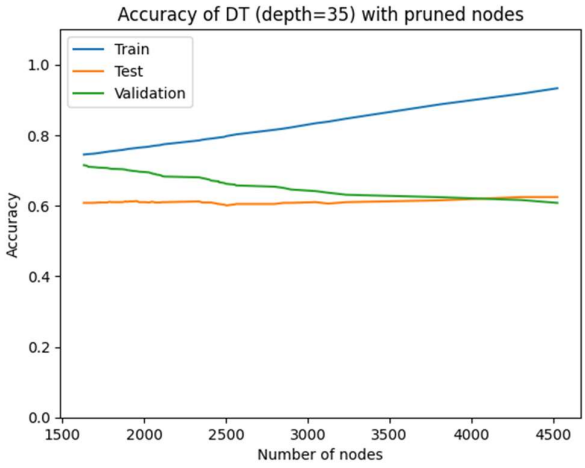


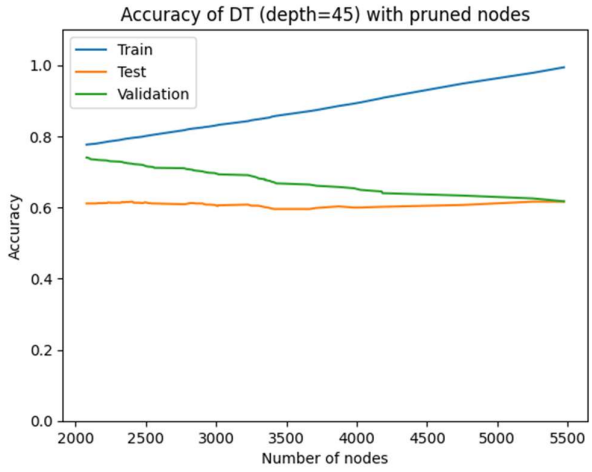
As expected, validation accuracy rises, as well as training accuracy reduces as the model becomes more generalized. But it seems test accuracy is largely unaffected, i.e. the pruned model doesn't generalize as well as one might expect.
The performance of the obtained tree is similar to before, but is more efficient at predictions given its smaller size.

Results from post-pruning the trees obtained in (b) are as follows:



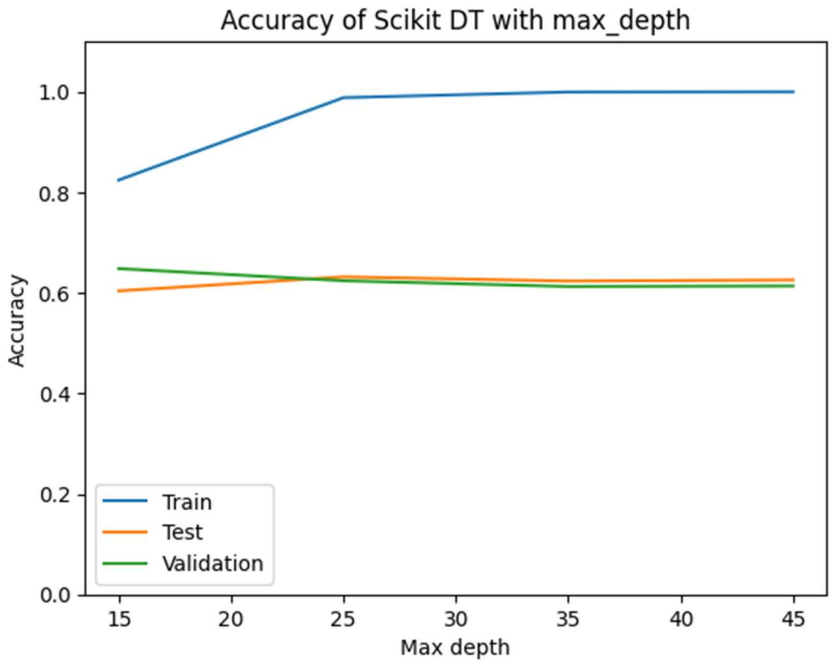






Most of these trees are getting pruned to about a third of their initial sizes. Observing the graphs, we see that the pruning process 'converges' when these train and validation accuracies converge. Weirdly, the test accuracy is almost the same throughout. This suggests maybe the post-pruned trees were selectively overfitting towards validation compared to training as before.

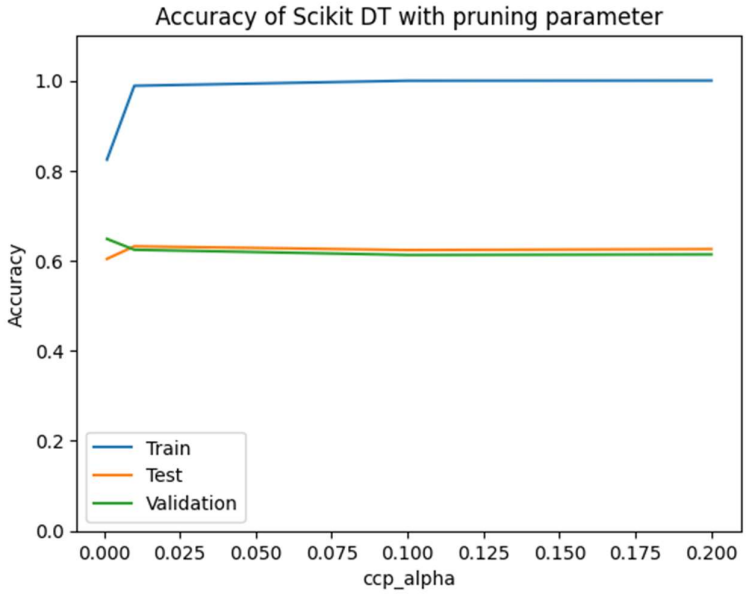
(d) Using the Scikit DecisionTreeClassifier with entropy criterion, on depths = [15,25,35,45]:



Although the train/test curves suggest the best depth to be somewhere around 25, according to our validation set, we obtain:

- Best depth: 15 with accuracy=0.6482758620689655.

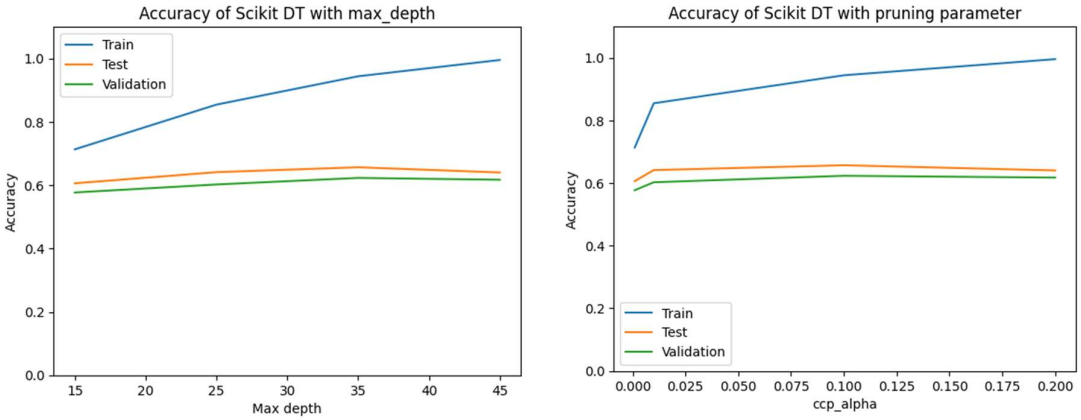
With the default depth parameter, we find the optimal ccp_alpha for our validation set:



We obtain: Best ccp_alpha: 0.001 with accuracy=0.6318510858324715

The best tree with these parameters gives the result:
- Train accuracy = 0.6844257058898684 - Test accuracy = 0.6225439503619442

Although, the above was done with sklearn's internal one-hot encoding. I tried repeating the same over our previously encoded datasets:



Here, we get:
- Best depth: 35 with accuracy=0.6229885057471264
- Best ccp_alpha: 0.001 with accuracy=0.6318510858324715

The best tree with these parameters gives the result:
- Train accuracy = 0.6946467356586176 - Test accuracy = 0.6318510858324715

(e) Searching over the list of parameter values given, based on the out-of-bag accuracy, i.e. oob_score attribute, obtained:
Best parameters:

a. n_estimators: 150, max_features: 0.3, min_samples_split: 10
b. Out-of-bag accuracy: 0.7310591542097866
c. Training accuracy: 0.9445509135045356
d. Validation accuracy: 0.7114942528735633
e. Test accuracy: 0.7311271975180972

Also tried using the GridSearchCV module on 'accuracy' scoring, obtained results:
Best parameters:

a. {'max_features': 0.3, 'min_samples_split': 10, 'n_estimators': 350}
b. Out-of-bag accuracy: 0.7273540309186151
c. Training accuracy: 0.9427622332950045
d. Validation accuracy: 0.7172413793103448
e. Test accuracy: 0.7238883143743536

Q2. Neural Networks

(a) We implement a class of Artificial Neural Networks based on matrix operations on numpy arrays. The primary operations in train method are:

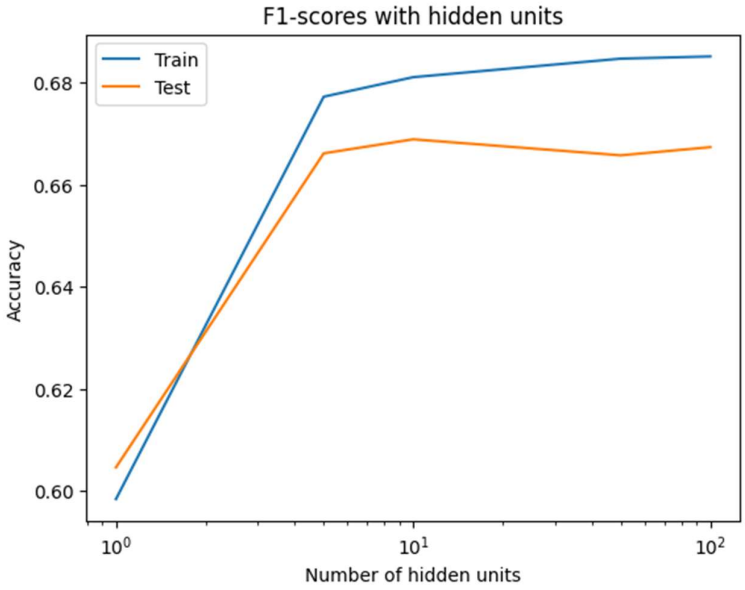
- **Forward propagation (fwd_prop):** Generates the list of layer outputs and their activations input into following layers.
- **Backward propagation (bwd_prop):** Calculates gradients based on the cross-entropy loss on the softmax output layer and passes backwards the previous weights' gradients calculated using the chain rule.

On a trial model with a singular hidden layer = [100], trained within 100 epochs, tolerance= 0.001 and with a learning rate of 0.01, we obtain results:

Test accuracy on hidden layer [100]:
- Epochs=100 : 67% accuracy, 0.7278 loss
- Epochs=500 : 72% accuracy, 0.6146 loss

(Note: The following parts have the model vitals recorded in the output2<part>.txt file.)

(b) Using the above implementation of ANN and the given parameters, for a single hidden-layer model with number of hidden units = [[1],[5],[10],[50],[100]]:



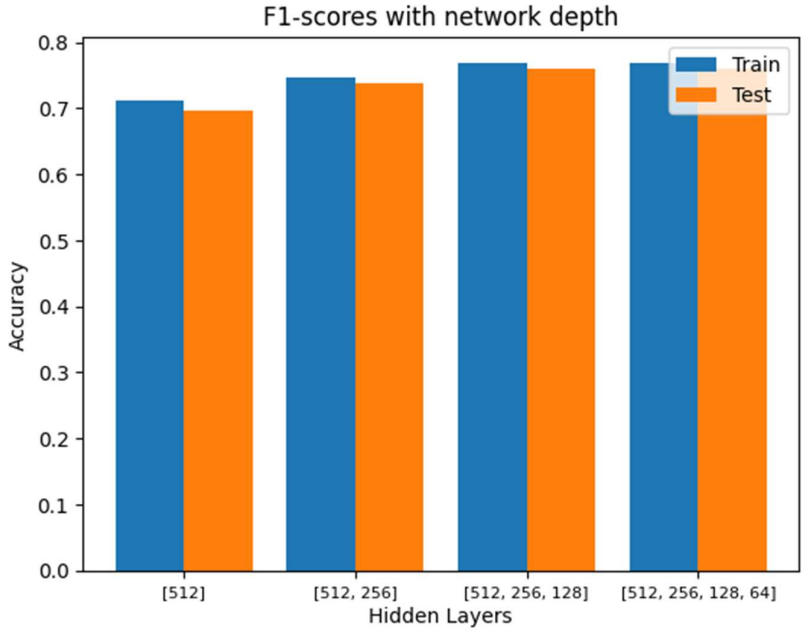
We see the f1-scores improve with the number of hidden units, but the score on test data is best when there are 10 hidden units.

(c) We now test a multi-layered model from the following architectures.

[[512],[512,256],[512,256,128],[512,256,128,64]]

We use the default tolerance and learning rate of 0.01 over 500 epochs. (For the 4th architecture, the error tolerance was reduced to 10^-6, else the training terminated prematurely.)

We obtain the following trends for the f1-scores:

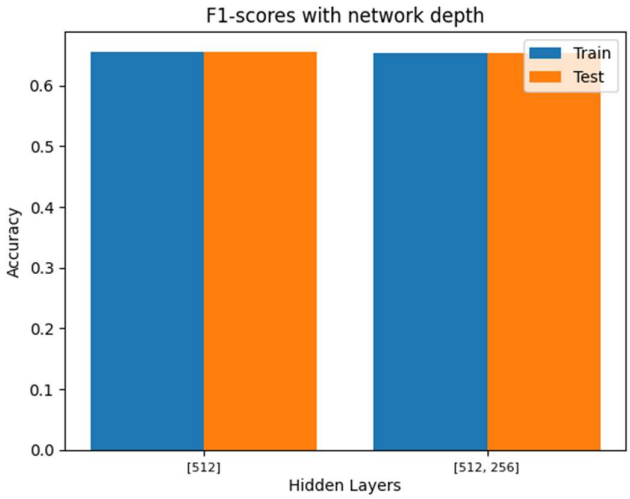


If we had used the same stopping criterion for the last model, it obtains trivial accuracies of around 20% on test and train data.

Overall, we notice that as the network grows deeper, the better it fits the dataset.
The maximum obtained scores are (all scores listed in file):
- Train(4-depth) = 0.7682536033570446 - Test(3-depth) = 0.7593450543277083

There is a miniscule fall off for test scores in the model, indicating that going further deep may lead to overfitting in the model.
Although, increasing the learning rate seems to be a better idea here.

(d) Used the initial seed = 0.01 for the adaptive learning rate model, we obtain:



For hidden_size: [512]:
Train scores:-

	precision	recall	f1-score	support
1	0.90	0.86	0.88	2074
2	0.68	0.67	0.67	2004
3	0.52	0.54	0.53	1892
4	0.44	0.50	0.47	1750
5	0.76	0.69	0.72	2280
accuracy			0.66	10000
macro avg	0.66	0.65	0.66	10000
weighted avg	0.67	0.66	0.67	10000

Test scores:-

	precision	recall	f1-score	support
1	0.90	0.90	0.90	228
2	0.67	0.65	0.66	202
3	0.51	0.56	0.53	183
4	0.51	0.49	0.50	195
5	0.70	0.68	0.69	192
accuracy			0.66	1000
macro avg	0.66	0.65	0.65	1000
weighted avg	0.67	0.66	0.66	1000

For hidden_size: [512, 256]:
Train scores:-

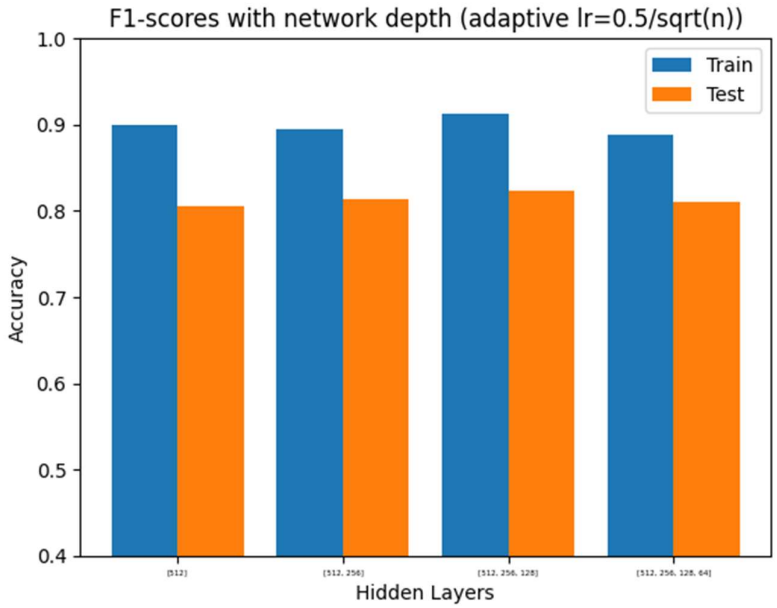
	precision	recall	f1-score	support
1	0.90	0.86	0.88	2073
2	0.68	0.68	0.68	1981
3	0.52	0.54	0.53	1858
4	0.44	0.49	0.46	1795
5	0.75	0.69	0.72	2293
accuracy			0.66	10000
macro avg	0.66	0.65	0.65	10000
weighted avg	0.67	0.66	0.66	10000

Test scores:-

	precision	recall	f1-score	support
1	0.91	0.89	0.90	236
2	0.65	0.66	0.66	195
3	0.50	0.57	0.53	175
4	0.51	0.48	0.50	198
5	0.70	0.67	0.68	196
accuracy			0.66	1000
macro avg	0.66	0.65	0.65	1000
weighted avg	0.67	0.66	0.67	1000

According to observations, the given seed rate was quite small and gave the same scores on the other models, i.e. [512,256,128] and [512,256,128,64] as well regardless of network depth.

Since the effective learning rate would be small at farther epochs, I tried experimenting with a larger seed = 0.5, and the results were better:



There is a significant increase in both train and test scores as follows:

For hidden_size: [512]:
Train scores:-

	precision	recall	f1-score	support
1	0.99	0.99	0.99	1969
2	0.95	0.94	0.95	1996
3	0.81	0.91	0.86	1753
4	0.81	0.80	0.81	2029
5	0.93	0.87	0.90	2253
accuracy			0.90	10000
macro avg	0.90	0.90	0.90	10000
weighted avg	0.90	0.90	0.90	10000

Test scores:-

	precision	recall	f1-score	support
1	0.98	0.99	0.99	227
2	0.87	0.86	0.86	200
3	0.65	0.76	0.70	171
4	0.71	0.63	0.66	210
5	0.82	0.80	0.81	192
accuracy			0.81	1000
macro avg	0.81	0.81	0.81	1000
weighted avg	0.81	0.81	0.81	1000

For hidden_size: [512, 256]:
Train scores:-

	precision	recall	f1-score	support
1	0.99	1.00	0.99	1954
2	0.93	0.98	0.95	1869
3	0.84	0.90	0.87	1824
4	0.73	0.81	0.77	1797
5	0.98	0.80	0.88	2556
accuracy			0.89	10000
macro avg	0.89	0.90	0.89	10000
weighted avg	0.90	0.89	0.90	10000

Test scores:-

	precision	recall	f1-score	support
1	0.99	0.99	0.99	229
2	0.85	0.91	0.88	185
3	0.68	0.79	0.73	173
4	0.64	0.64	0.64	185
5	0.91	0.75	0.82	228
accuracy			0.82	1000
macro avg	0.81	0.82	0.81	1000
weighted avg	0.83	0.82	0.82	1000

For hidden_size: [512, 256, 128]:
Train scores:-

	precision	recall	f1-score	support
1	0.99	1.00	0.99	1953
2	0.95	0.98	0.96	1915
3	0.90	0.92	0.91	1912
4	0.77	0.84	0.81	1843
5	0.95	0.83	0.89	2377
accuracy			0.91	10000
macro avg	0.91	0.91	0.91	10000
weighted avg	0.91	0.91	0.91	10000

Test scores:-

	precision	recall	f1-score	support
1	0.97	1.00	0.98	224
2	0.90	0.89	0.89	201
3	0.74	0.81	0.77	183
4	0.66	0.66	0.66	186
5	0.84	0.77	0.80	206
accuracy			0.83	1000
macro avg	0.82	0.82	0.82	1000
weighted avg	0.83	0.83	0.83	1000

For hidden_size: [512, 256, 128, 64]:
Train scores:-

	precision	recall	f1-score	support
1	0.97	1.00	0.99	1924
2	0.91	0.96	0.93	1866
3	0.86	0.87	0.87	1933
4	0.73	0.82	0.77	1775
5	0.97	0.81	0.88	2502
accuracy			0.89	10000
macro avg	0.89	0.89	0.89	10000
weighted avg	0.89	0.89	0.89	10000

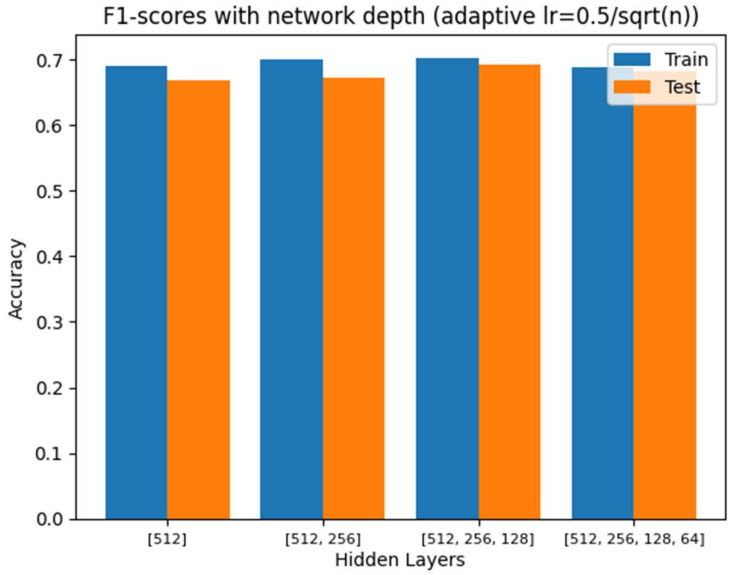
Test scores:-

	precision	recall	f1-score	support
1	0.97	0.99	0.98	224
2	0.83	0.90	0.86	182
3	0.70	0.77	0.73	180
4	0.67	0.64	0.66	197
5	0.89	0.76	0.82	217
accuracy			0.82	1000
macro avg	0.81	0.81	0.81	1000
weighted avg	0.82	0.82	0.82	1000

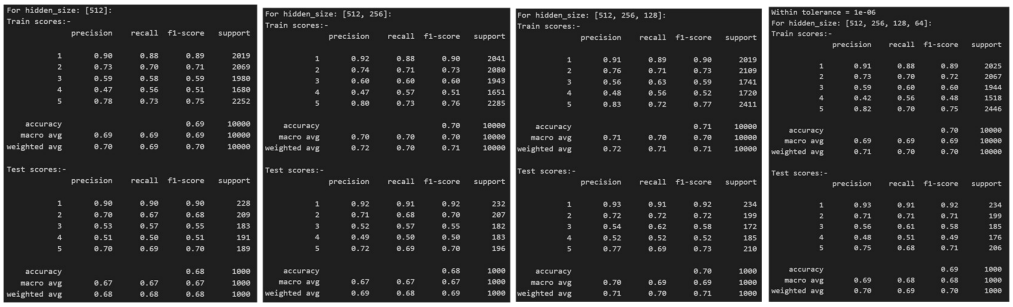
We can see an increasing trend with network depth, as the f1 scores go from below 80 to 90+. But it falls off slightly at [512,256,128,64], most likely because the learning rate we chose for this model was too large (can be inferred from the losses increasing near the epoch ends), that can be controlled by stopping our model when loss function increases. (Infact, the 3rd model also showed min loss = 0.1964). Nevertheless, running for more than 500 epochs was useful since test results improve, suggesting an underfit earlier. The adaptive rate also helps deal with overfitting here.

(e) Using ReLU activation, the initial learning rate seed of 0.01 had to be used, since the gradients were blowing up (presumably because ReLU isn't bound like sigmoid). Also, reduced the epoch limit back to 500 since ReLU should ideally take lesser time to converge.

With these changes reverted, we obtain the following:



We see f1-scores around the 70% mark, improving faintly with network depth. The last model converges before 500 epochs as the error falls into the tolerance limit, making it run for almost half the time of the previous one, while giving almost the same results.



For hidden_size: [512]:
Train scores:-

	precision	recall	f1-score	support
1	0.90	0.88	0.89	2019
2	0.73	0.70	0.71	2069
3	0.59	0.58	0.59	1988
4	0.47	0.56	0.51	1680
5	0.78	0.73	0.75	2252
accuracy			0.69	10000
macro avg	0.69	0.69	0.69	10000
weighted avg	0.70	0.69	0.70	10000

Test scores:-

	precision	recall	f1-score	support
1	0.90	0.90	0.90	228
2	0.70	0.67	0.68	209
3	0.53	0.57	0.55	183
4	0.51	0.50	0.51	191
5	0.70	0.69	0.70	189
accuracy			0.68	1000
macro avg	0.67	0.67	0.67	1000
weighted avg	0.68	0.68	0.68	1000

For hidden_size: [512, 256]:
Train scores:-

	precision	recall	f1-score	support
1	0.92	0.88	0.90	2041
2	0.74	0.71	0.73	2080
3	0.60	0.60	0.60	1941
4	0.47	0.57	0.51	1651
5	0.80	0.73	0.76	2285
accuracy			0.70	10000
macro avg	0.70	0.70	0.70	10000
weighted avg	0.72	0.70	0.71	10000

Test scores:-

	precision	recall	f1-score	support
1	0.92	0.91	0.92	232
2	0.71	0.68	0.70	207
3	0.52	0.57	0.55	182
4	0.49	0.50	0.50	183
5	0.72	0.69	0.70	196
accuracy			0.68	1000
macro avg	0.67	0.67	0.67	1000
weighted avg	0.69	0.68	0.69	1000

For hidden_size: [512, 256, 128]:
Train scores:-

	precision	recall	f1-score	support
1	0.91	0.89	0.90	2019
2	0.76	0.71	0.73	2109
3	0.56	0.63	0.59	1741
4	0.48	0.56	0.52	1720
5	0.83	0.72	0.77	2411
accuracy			0.71	10000
macro avg	0.71	0.70	0.70	10000
weighted avg	0.72	0.71	0.71	10000

Test scores:-

	precision	recall	f1-score	support
1	0.93	0.91	0.92	234
2	0.72	0.72	0.72	199
3	0.54	0.62	0.58	172
4	0.52	0.52	0.52	185
5	0.77	0.69	0.73	210
accuracy			0.70	1000
macro avg	0.70	0.69	0.69	1000
weighted avg	0.71	0.70	0.71	1000

Within tolerance = 1e-06
For hidden_size: [512, 256, 128, 64]:
Train scores:-

	precision	recall	f1-score	support
1	0.91	0.88	0.89	2025
2	0.73	0.70	0.72	2067
3	0.59	0.60	0.60	1944
4	0.42	0.56	0.48	1518
5	0.82	0.70	0.75	2446
accuracy			0.70	10000
macro avg	0.69	0.69	0.69	10000
weighted avg	0.71	0.70	0.70	10000

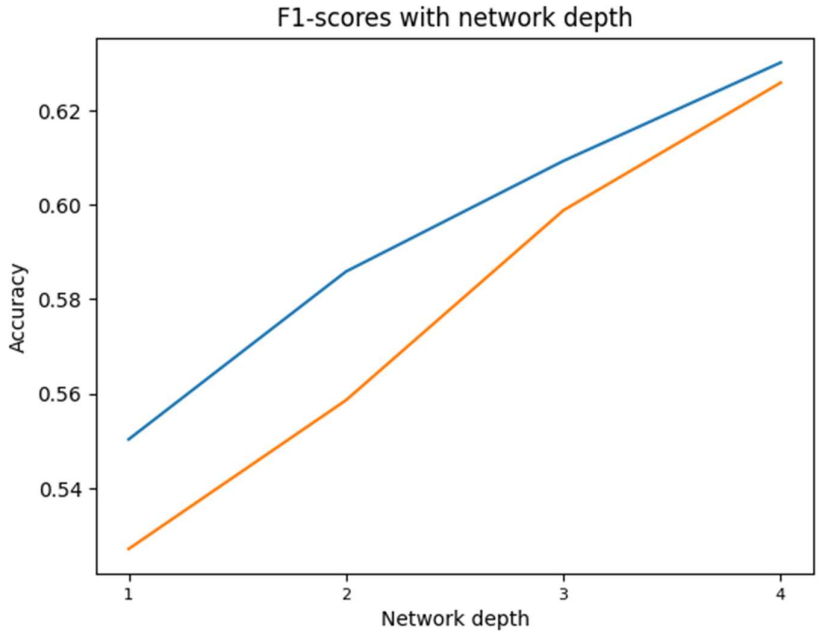
Test scores:-

	precision	recall	f1-score	support
1	0.93	0.91	0.92	234
2	0.71	0.71	0.71	199
3	0.56	0.61	0.58	185
4	0.48	0.51	0.49	176
5	0.75	0.68	0.71	206
accuracy			0.69	1000
macro avg	0.69	0.68	0.68	1000
weighted avg	0.70	0.69	0.70	1000

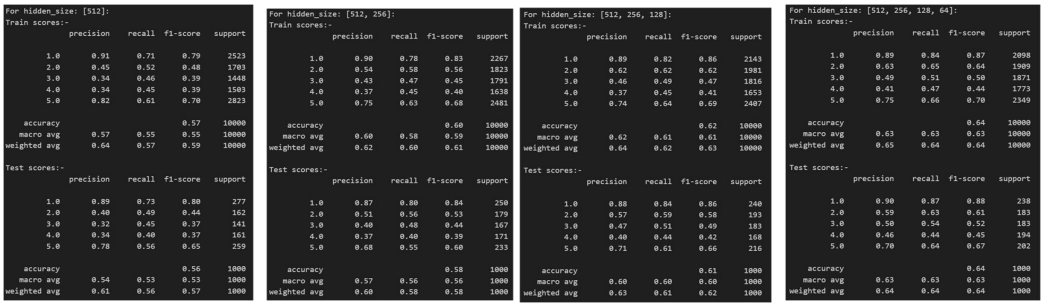
Compared to the sigmoid-based models, the ReLU-based models do in fact seem to perform better If we compare results with same learning seed = 0.01, there is an increase of around 4% over their sigmoid counterparts, i.e the original results in (d) with 0.01 adaptive learning seed.

It would be interesting to see the performance of relu model with higher learning rates. I have tried a few countermeasures such as: 1. subtracting maximum in softmax layer to prevent overflow; 2. Downscaling ReLU function by a factor, but they need further work, and were not very effective.

(f) Using the Scikit MLPClassifier, we see a clear trend of increasing f1-score (and accuracy) with network depth (number of hidden layers):



We used the SGD optimizer with 'invscaling' learning rate. The activation function used in the hidden layer was ReLU and a batch size of 32, similar to our models in (e).



For hidden_size: [512]:
Train scores:-

	precision	recall	f1-score	support
1.0	0.91	0.71	0.79	2523
2.0	0.45	0.52	0.48	1703
3.0	0.34	0.46	0.39	1448
4.0	0.34	0.45	0.39	1503
5.0	0.82	0.61	0.70	2823
accuracy			0.57	10000
macro avg	0.57	0.55	0.55	10000
weighted avg	0.64	0.57	0.59	10000

Test scores:-

	precision	recall	f1-score	support
1.0	0.89	0.73	0.80	277
2.0	0.40	0.49	0.44	162
3.0	0.32	0.45	0.37	141
4.0	0.34	0.40	0.37	161
5.0	0.78	0.56	0.65	259
accuracy			0.56	1000
macro avg	0.54	0.53	0.53	1000
weighted avg	0.61	0.56	0.57	1000

For hidden_size: [512, 256]:
Train scores:-

	precision	recall	f1-score	support
1.0	0.90	0.78	0.83	2267
2.0	0.54	0.58	0.56	1823
3.0	0.43	0.47	0.45	1791
4.0	0.37	0.45	0.40	1638
5.0	0.75	0.63	0.68	2481
accuracy			0.60	10000
macro avg	0.60	0.58	0.59	10000
weighted avg	0.62	0.60	0.61	10000

Test scores:-

	precision	recall	f1-score	support
1.0	0.87	0.80	0.84	250
2.0	0.51	0.56	0.53	179
3.0	0.40	0.48	0.44	167
4.0	0.37	0.40	0.39	171
5.0	0.68	0.55	0.60	233
accuracy			0.58	1000
macro avg	0.57	0.56	0.56	1000
weighted avg	0.60	0.58	0.58	1000

For hidden_size: [512, 256, 128]:
Train scores:-

	precision	recall	f1-score	support
1.0	0.89	0.82	0.86	2143
2.0	0.62	0.62	0.62	1981
3.0	0.46	0.49	0.47	1816
4.0	0.37	0.45	0.41	1653
5.0	0.74	0.64	0.69	2407
accuracy			0.62	10000
macro avg	0.62	0.61	0.61	10000
weighted avg	0.64	0.62	0.63	10000

Test scores:-

	precision	recall	f1-score	support
1.0	0.88	0.84	0.86	240
2.0	0.57	0.59	0.58	193
3.0	0.47	0.51	0.49	183
4.0	0.40	0.44	0.42	168
5.0	0.71	0.61	0.66	216
accuracy			0.61	1000
macro avg	0.60	0.60	0.60	1000
weighted avg	0.63	0.61	0.62	1000

For hidden_size: [512, 256, 128, 64]:
Train scores:-

	precision	recall	f1-score	support
1.0	0.89	0.84	0.87	2098
2.0	0.63	0.65	0.64	1909
3.0	0.49	0.51	0.50	1871
4.0	0.41	0.47	0.44	1773
5.0	0.75	0.66	0.70	2349
accuracy			0.64	10000
macro avg	0.63	0.63	0.63	10000
weighted avg	0.65	0.64	0.64	10000

Test scores:-

	precision	recall	f1-score	support
1.0	0.90	0.87	0.88	238
2.0	0.59	0.63	0.61	183
3.0	0.50	0.54	0.52	183
4.0	0.46	0.44	0.45	194
5.0	0.70	0.64	0.67	202
accuracy			0.64	1000
macro avg	0.63	0.63	0.63	1000
weighted avg	0.64	0.64	0.64	1000

As the model depth increases, both test and train accuracies improve. All the architectures ran for the default maximum iterations = 200 without convergence, indicates that the accuracy may improve further.