COL774 Assignment - 1

Vidushi Maheshwari (2021CS10083)

April 13, 2025

Contents

§1 Decision Trees (and Random Forests)

§1.1 Decision Tree Construction

Tree Depth	Train Accuracy	Test Accuracy
1	0.7522	0.7543
5	0.8669	0.8059
10	0.9382	0.7731
15	0.9840	0.7615
20	0.9915	0.7607

Table 1: Train and Test Accuracy for Different Tree Depths

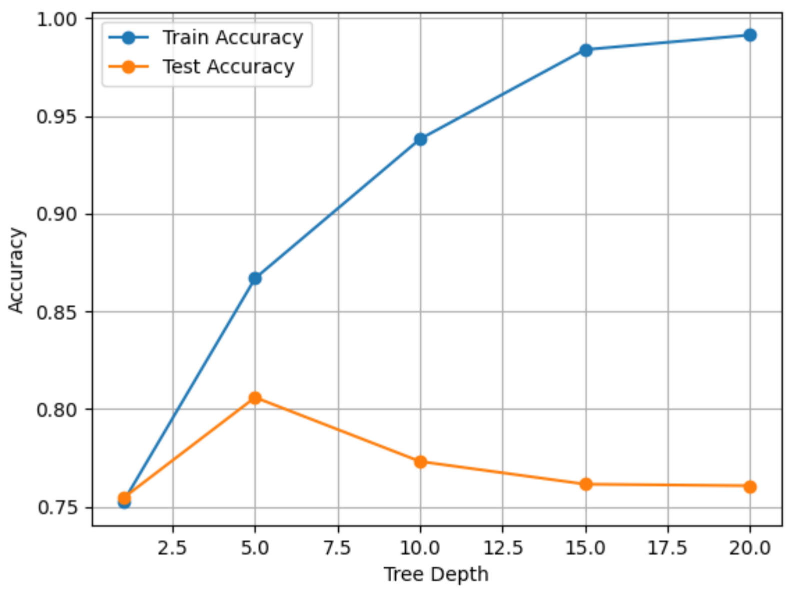

Figure 1: Train And Test Accuracy vs Tree Depth

- At a tree depth of 1, both training and test accuracies are low (approximately 75%), indicating the model is too simple to capture the underlying patterns in the data.
- A significant increase in both training (86.7%) and test (80.6%) accuracies is observed at depth 5, suggesting this depth offers a good balance between bias and variance.
- As the depth increases beyond 5, the training accuracy continues to rise (reaching over 99% at depth 20), while test accuracy declines. This indicates overfitting, where the model learns noise in the training data.
- A maximum depth of 5 gives the best test performance. Beyond this, deeper trees lead to reduced generalization. Pruning or using ensemble methods like Random Forests could help address overfitting.

§1.2 Decision Tree One Hot Encoding

Max Depth	Train Accuracy	Test Accuracy
25	0.8955	0.8392
35	0.9222	0.8344
45	0.9428	0.8303
55	0.9591	0.8288

Table 2: Decision Tree Accuracy with One-Hot Encoding

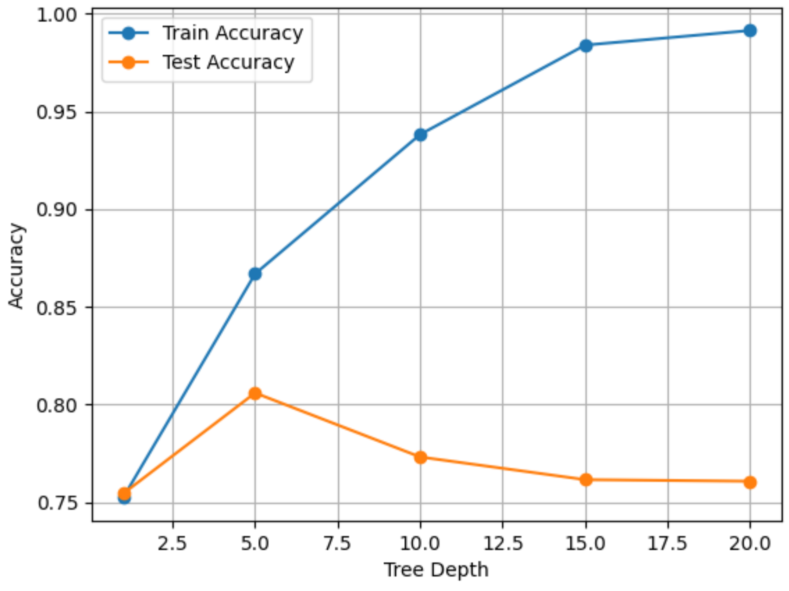

Figure 2: Train And Test Accuracy vs Tree Depth

- One-hot encoding generally improves model expressiveness by allowing categorical attributes to be used more flexibly during splits.
- After applying one-hot encoding, we observe that the training accuracy continues to increase with tree depth, indicating that the model fits the training data more closely.
- The test accuracy, however, starts to decline slightly as depth increases, suggesting potential overfitting at higher depths.
- The test accuracy after applying one hot encoding in part(b) is higher for even greater depths, than the accuracy obtained in part(a).

§1.3 Decision Tree Post Pruning

Depth	Train Accuracy	Val Accuracy	Test Accuracy
25	0.8780	0.8488	0.8411
35	0.8891	0.8487	0.8392
45	0.9011	0.8515	0.8341
55	0.9198	0.8520	0.8297

Table 3: Post-pruning accuracies of decision trees

Accuracy Plots during Tree Pruning

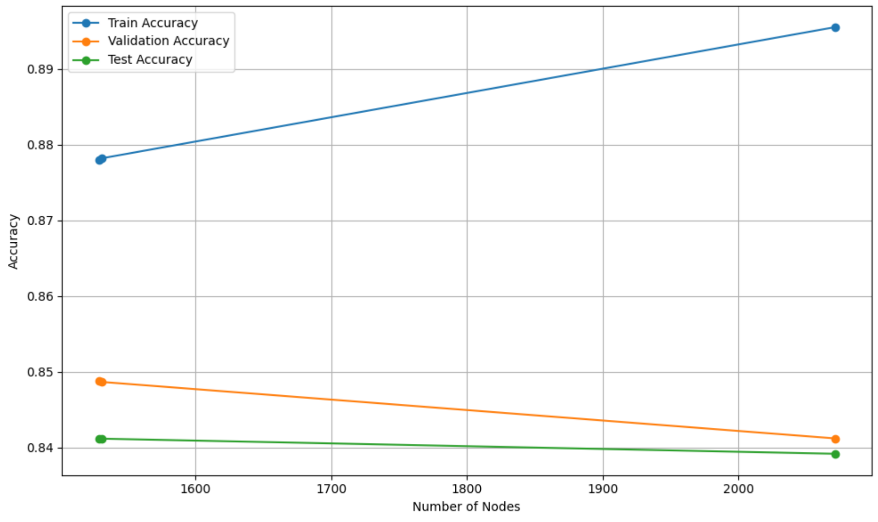

Figure 3: Depth = 25

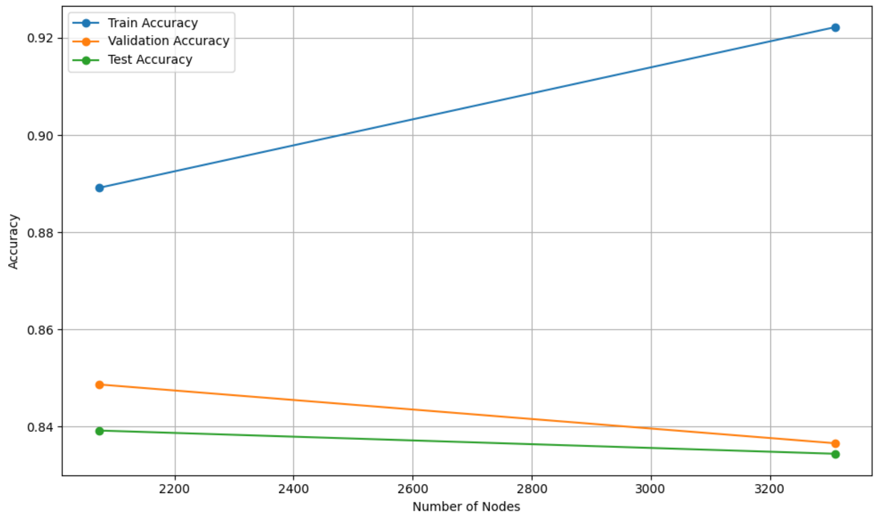

Figure 4: Depth = 35

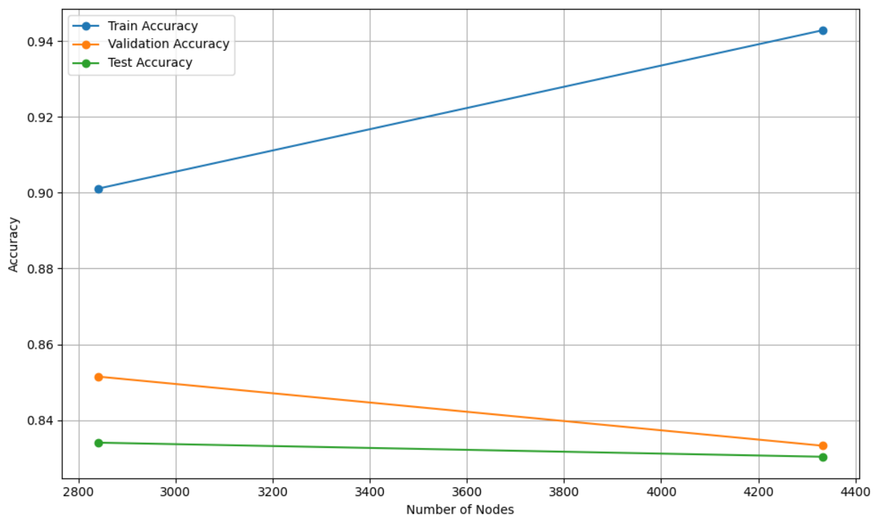

Figure 5: Depth = 45

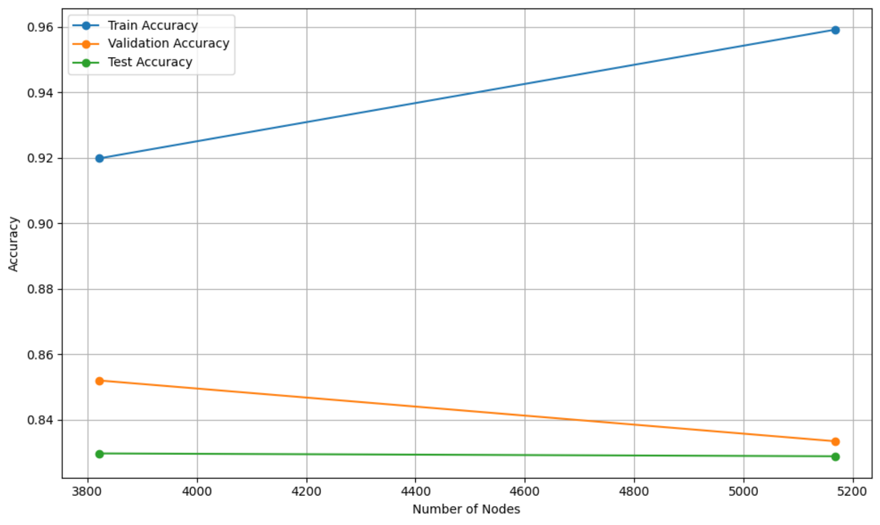

Figure 6: Depth = 55

Observations

- Post-pruning slightly improves validation accuracy, especially for depths 45 and 55, where the validation accuracy reaches 0.8515 and 0.8520, respectively. This shows that pruning helps generalization by reducing overfitting.
- As nodes are pruned, the training accuracy decreases consistently. This is expected since pruning simplifies the tree and thus may sacrifice fit to the training data.
- The test accuracy initially improves or remains stable with pruning, especially at smaller depths. For instance:
  - At depth 25, test accuracy is 0.8411, highest among all.
  - For deeper trees (e.g., depth 55), test accuracy slightly declines to 0.8297, indicating overfitting when pruning isn't aggressive enough.
- In all plots:

  - Train accuracy increases with the number of nodes (before pruning).
  - Validation and test accuracies tend to peak and then slightly decline as trees grow more complex, especially visible for depths 45 and 55.

  This validates the need for post-pruning to avoid overfitting in deeper trees.

- Based on validation accuracy:
  - Depth 55 performs slightly better in validation but worse on the test set than depth 25.
  - Depth 25 provides the best test accuracy (0.8411), suggesting it generalizes better despite being shallower.

§1.4 Decision Tree sci-kit learn

(i) Variable `max_depth`

Depth	Train Accuracy	Val Accuracy	Test Accuracy
25	0.9175	0.8508	0.8472
35	0.9383	0.8473	0.8440
45	0.9599	0.8480	0.8436
55	0.9726	0.8427	0.8394

Table 4: Accuracies of decision trees for different *max_depth*

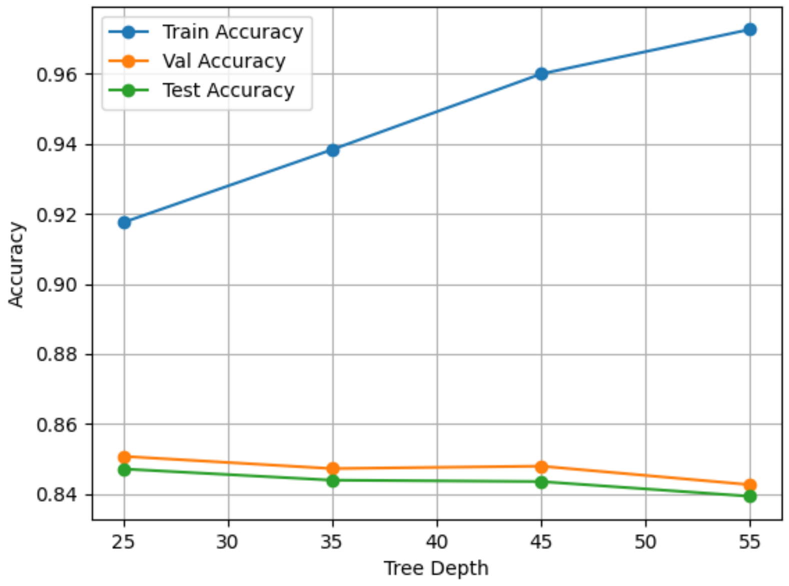

Figure 7: Train, Val and Test Accuracy vs Tree Depth

(ii) Variable `ccp_alpha`

ccp_alpha	Train Accuracy	Validation Accuracy	Test Accuracy
0.001	0.8495	0.8507	0.8480
0.010	0.8134	0.8122	0.8139
0.100	0.7522	0.7464	0.7543
0.200	0.7522	0.7464	0.7543

Table 5: Accuracies of decision trees for different *ccp_alpha*

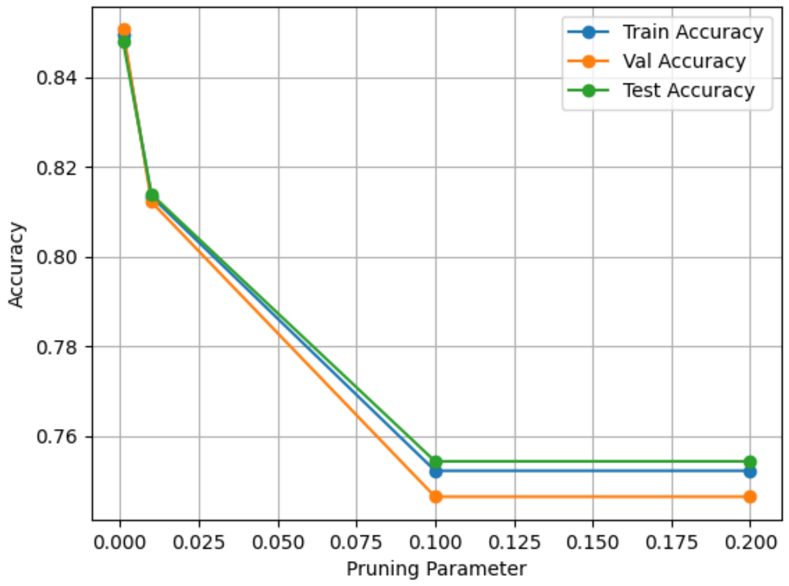

Figure 8: Train, Val and Test Accuracy vs Tree Depth

Comparision and Observations

- Train Accuracy:
  - In part (b), the train accuracy increases significantly as depth increases, indicating that the model overfits as it grows deeper.
  - In part (c), the train accuracy also increases but at a slower rate due to post-pruning, which helps in preventing overfitting.
  - In part (d)(i), using entropy as the criterion leads to higher train accuracy at all depths, as entropy tends to produce better splits compared to Gini.
  - In part (d)(ii), increasing `ccp_alpha` reduces the train accuracy, as the tree becomes simpler with more pruning.
- Validation Accuracy:
  - In part (b), validation accuracy decreases as depth increases, with the best performance at depth 25.
  - In part (c), post-pruning improves the validation accuracy across all depths, showing that pruning helps in generalization.

  - In part (d)(i), validation accuracy improves with depth, especially at deeper trees, and entropy also improves the validation accuracy compared to part (b).
  - In part (d)(ii), validation accuracy is highest at `ccp_alpha = 0.001`, showing the effectiveness of light pruning in improving validation performance.

- Test Accuracy:
  - In part (b), test accuracy declines with increasing depth, suggesting overfitting as the tree becomes more complex.
  - In part (c), test accuracy shows a smaller decline due to post-pruning, with depth 25 giving the best performance.
  - In part (d)(i), test accuracy improves at depth 25 compared to part (b) (from 0.8392 to 0.8472), and it stabilizes for deeper trees, with entropy helping reduce overfitting.
  - In part (d)(ii), test accuracy is highest at `ccp_alpha = 0.001`, showing that pruning with a small `ccp_alpha` value helps improve generalization.

§1.5 Random Forests

Best Parameters

- `max_features = 0.5`
- `min_samples_split = 8`
- `n_estimators = 150`

Metric	Accuracy (%)
Out-of-Bag Accuracy	85.97
Train Accuracy	99.88
Validation Accuracy	86.41
Test Accuracy	86.10

Table 6: Model Accuracy across Different Evaluation Metrics

§2 Neural Networks

§2.1 Generic Neural Network Architecture

For Part (a), we implemented a fully connected feedforward neural network for multi-class classification on the GTSRB dataset. The network is modular, allowing for flexible hidden layer configurations, mini-batch sizes, and input/output dimensions. It uses mini-batch Stochastic Gradient Descent (SGD) for training and the cross-entropy loss function with softmax activation at the output layer to generate class probabilities.

Loss Function

The cross-entropy loss is given by:

$$J(\theta) = -\sum_{k=1}^{r} 1\{k = \tilde{k}\} \log(o_k)$$

where $\tilde{k}$ is the true label, and o_k is the predicted probability.

Backpropagation

We use the chain rule to compute the gradients for weights and biases. The gradient for the output layer is:

$$\frac{\partial J(\theta)}{\partial z_k^{[L]}} = \begin{cases} o_k - 1, & \text{if } k = \tilde{k} \\ o_k, & \text{otherwise} \end{cases}$$

Training

The network is trained using mini-batch SGD. For each mini-batch, we:

1. Perform a forward pass to compute the output.
2. Calculate the loss using cross-entropy.
3. Compute gradients via backpropagation.
4. Update parameters using the gradients.

Input Parameters

The network accepts the following parameters:

- **Input Size (n):** Number of features (e.g., $28 \times 28 \times 3 = 2352$ for RGB images).
- **Hidden Layer Architecture:** List of layer sizes (e.g., `[512, 256]`).
- **Number of Target Classes (r):** 43 for GTSRB.
- **Mini-Batch Size (M):** Defines the number of samples per batch.

§2.2 Single Hidden Layer Neural Network

Stopping Criteria: Fixed number of epochs or Convergence of training loss

Hidden Units	Train			Test		
	F1	Precision	Recall	F1	Precision	Recall
1	0.0027	0.0015	0.0230	0.0028	0.0015	0.0233
5	0.2440	0.2721	0.2915	0.2353	0.2445	0.2744
10	0.6984	0.7486	0.7056	0.6018	0.6228	0.5985
50	0.9469	0.9584	0.9380	0.8165	0.8629	0.7992
100	0.9557	0.9664	0.9473	0.8172	0.8651	0.8004

Table 7: F1 score, Precision and Recall for different number of Hidden Units

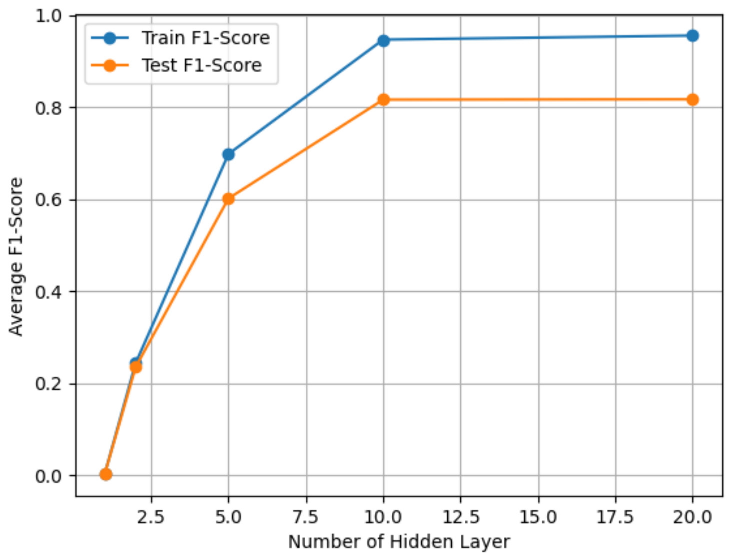

Figure 9: Average F1-Score vs Number of Hidden Units

Observations

- The model performs poorly with only 1 hidden unit, showing very low F1 scores, precision, and recall on both the training and test sets. This indicates underfitting, where the model cannot capture the complexity of the data.
- As the number of hidden units increases to 5 and 10, there is a noticeable improvement in model performance, especially in the training F1 score and generalization to the test set. However, precision and recall remain moderate, showing some room for improvement.
- The performance of the model improves significantly with 50 and 100 hidden units, showing high F1 scores, precision, and recall on both training and test sets. These configurations help the model generalize well to unseen data without substantial overfitting.
- Despite high performance with larger hidden layers, there is a slight difference between training and test F1 scores, indicating that overfitting may be a concern. However, the gap is not substantial, suggesting the model is generally robust.

§2.3 Varying Depth of Hidden Layers

Stopping Criteria: Fixed number of epochs or Convergence of training loss

Hidden Units	Train			Test		
	F1	**Precision**	**Recall**	**F1**	**Precision**	**Recall**
[512]	0.9456	0.9594	0.9346	0.8051	0.8561	0.7835
[512, 256]	0.9114	0.9277	0.8995	0.7517	0.7821	0.7438
[512, 256, 128]	0.7127	0.7723	0.7002	0.5812	0.6258	0.5721
[512, 256, 128, 64]	0.1878	0.1778	0.2277	0.1718	0.1736	0.2096

Table 8: F1 score, Precision and Recall for different hidden layer depths (fixed learning rate)

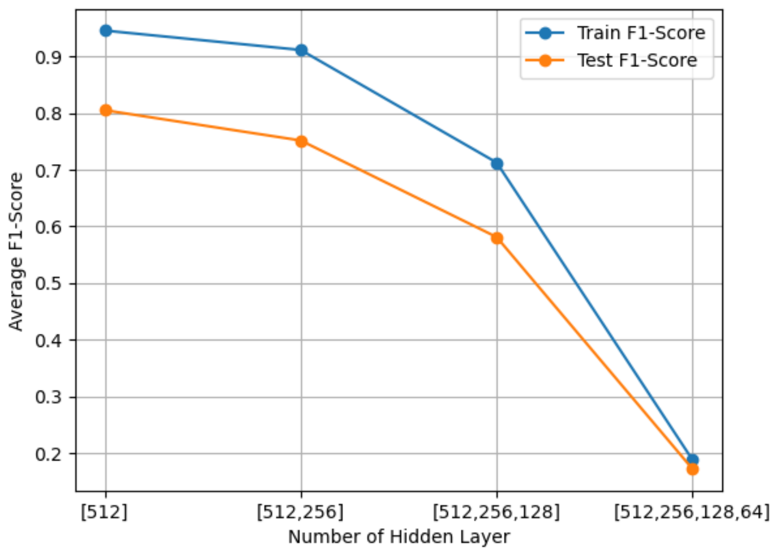

Figure 10: Average F1-Score vs Depth of Hidden Units

Observations

- The network with a single hidden layer `[512]` performed the best, achieving the highest test F1 score of 0.8051, indicating that a moderately deep architecture can effectively model the data without overfitting.
- As the network depth increased to `[512, 256]` and `[512, 256, 128]`, both training and test performance gradually declined, suggesting that deeper architectures with fixed learning rate and sigmoid activation may face training difficulties such as vanishing gradients.
- The architecture `[512, 256, 128]` still maintained reasonable performance (test F1 = 0.5812), but the decline from shallower networks indicates diminishing returns with added depth under current training conditions.
- The deepest network `[512, 256, 128, 64]` performed very poorly (test F1 = 0.1718), likely due to significant training instability or poor convergence, reinforcing the challenge of training deep networks with sigmoid activations and fixed learning rate.

§2.4 Adaptive Learning Rate

Hidden Units	Train			Test		
	F1	Precision	Recall	F1	Precision	Recall
[512]	0.7658	0.8738	0.7388	0.6581	0.7555	0.6342
[512, 256]	0.4059	0.5096	0.4109	0.3588	0.4363	0.3692
[512, 256, 128]	0.0474	0.0470	0.0794	0.0425	0.0366	0.0704
[512, 256, 128, 64]	0.0043	0.0024	0.0237	0.0060	0.0034	0.0300

Table 9: F1 score, Precision and Recall for adaptive learning rate

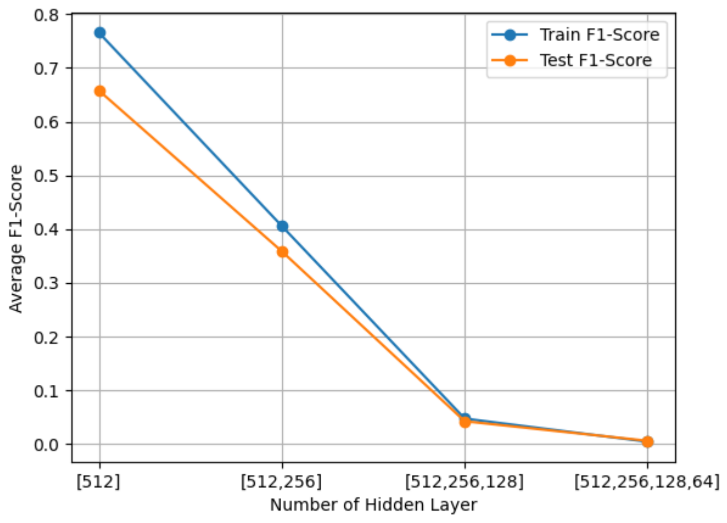

Figure 11: Average F1-Score vs Depth of Hidden Units

Observations

- In Part (c), the network with a single hidden layer `[512]` achieved the highest test F1 score of 0.8051, whereas in Part (d), the same architecture achieved a lower test F1 score of 0.6581, indicating that the adaptive learning rate did not significantly improve performance for shallow networks.
- The deeper networks with adaptive learning rate (`[512, 256, 128]` and `[512, 256, 128, 64]`) showed significantly worse results in Part (d) (F1 = 0.0425 and 0.0060) compared to Part (c), where deeper networks were already facing performance degradation. The adaptive learning rate did not solve issues like vanishing gradients or overfitting in these networks.
- The adaptive learning rate led to slower convergence for deeper networks, as the learning rate decreased over time. This resulted in poorer performance for deeper models, as the model didn't have enough momentum to converge effectively. The slower convergence may have prevented the model from reaching optimal performance.
- A more suitable criterion for using an adaptive learning rate would have been a larger number of epochs compared to Part (c). Since the adaptive learning rate reduces the learning rate over time, convergence is slower, and it would require more training epochs to reach optimal performance.

§2.5 ReLU activation

Hidden Units	Train			Test		
	F1	**Precision**	**Recall**	**F1**	**Precision**	**Recall**
[512]	0.9416	0.9574	0.9293	0.8096	0.8592	0.7855
[512, 256]	0.9568	0.9642	0.9510	0.8199	0.8483	0.8098
[512, 256, 128]	0.8391	0.8866	0.8156	0.6881	0.7341	0.6759
[512, 256, 128, 64]	0.8418	0.8809	0.8252	0.6848	0.7430	0.6733

Table 10: F1 score, Precision, and Recall for ReLU activation

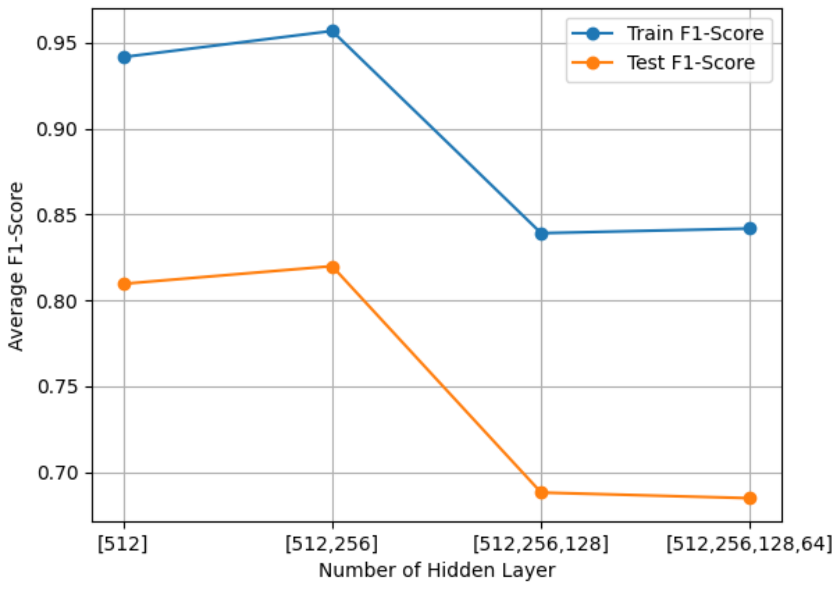

Figure 12: Average F1-Score vs Depth of Hidden Units

Observations

- Part (e) with ReLU activation significantly outperforms Part (d) with adaptive learning rate, particularly on the test set. For example, the test F1 score for [512] is 0.8096 in Part (e) compared to 0.6581 in Part (d), showing a clear advantage for ReLU in improving convergence.
- Deeper models in Part (e), such as [512, 256, 128] (F1 = 0.6881), maintain decent performance on the test set, whereas the same models in Part (d) ([512, 256, 128, 64]) show poor performance (F1 = 0.1718), indicating that the adaptive learning rate struggles to effectively train deeper networks.
- The adaptive learning rate in Part (d) leads to slower convergence and worse performance, especially in deeper architectures, where the learning rate becomes too small for effective learning. This results in poor test performance across deeper networks.
- ReLU activation in Part (e) facilitates faster convergence and better performance, particularly for deeper networks, making it a more suitable choice for training multi-layered neural networks compared to the adaptive learning rate in Part (d).

§2.6 MLPClassifier by Scikit-Learn

Hidden Units	Train			Test		
	F1	**Precision**	**Recall**	**F1**	**Precision**	**Recall**
[512]	0.9060	0.9329	0.8868	0.7721	0.8365	0.7466
[512, 256]	0.9201	0.9357	0.9086	0.7930	0.8269	0.7792
[512, 256, 128]	0.9098	0.9224	0.9007	0.7754	0.8029	0.7682
[512, 256, 128, 64]	0.8955	0.9093	0.8862	0.7644	0.7865	0.7591

Table 11: F1 score, Precision and Recall for different hidden layer depths

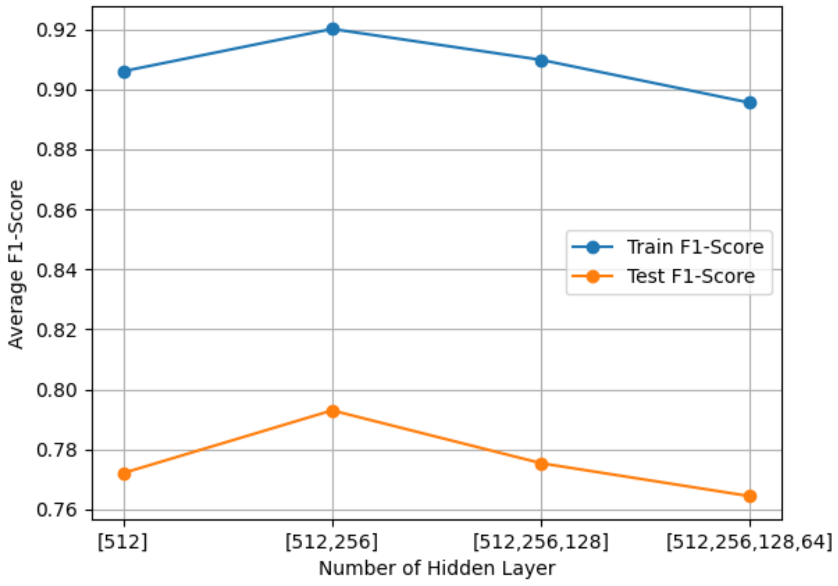

Figure 13: Average F1-Score vs Depth of Hidden Units

Observations

- In Part (e), the test F1 score for the architecture with [512] was 0.8096, while with MLPClassifier, it is 0.7721, indicating that the custom ReLU network performs slightly better on simpler architectures.
- MLPClassifier shows a higher precision across all architectures, for example, 0.8365 for [512] compared to 0.8096 in Part (e). This suggests that MLPClassifier provides a more precise classification, but with slightly lower recall values than in Part (e).
- For deeper architectures, MLPClassifier shows better stability and performance. In Part (e), the performance drops significantly for [512, 256, 128, 64] (F1 = 0.1718), while in Part (f), it is much higher at 0.7644, indicating that MLPClassifier handles deeper networks more effectively.
- Overall, MLPClassifier performs better with deeper architectures due to its built-in optimization techniques and regularization. While Part (e) (ReLU activation) gives slightly better performance on simpler networks, MLPClassifier seems more robust and efficient for complex models.