CS640 PA1 Report

Name: Alex Wong
Team Members: Alex Wong, Rahul Suresh, Wei-Hsiang Lin
Date: 10/09/2019

Problem Definition

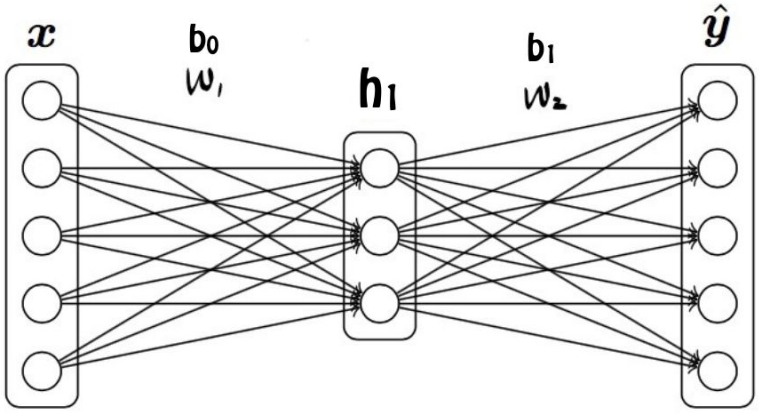
In Part 1, we are given two different sets of data; both with 2 classes. The 2 classes are distributed linearly or non linearly. In Part 2, we are given a set of data with 10 classes. In both parts, our goal is to implement and train a neural network to accurately classify a given input into the correct class.

In this assignment, we held the assumption that a shallow neural network (with 1 layer in the hidden layer) is enough to learn the decision boundary for both datasets. We also anticipated the 2 difficulties. The first is backward propagation could result in error when calculating the gradients at each step and the second one is updating the weight matrices can be tricky.

Method and Implementation

In this assignment, we implemented the forward step and backward propagation for a single hidden layer neural network from scratch. Then, we fitted our network on three different datasets, and tested if our network can correctly label testing set input. In order to assess how 'correct' our neural network was performing, we used ROC curve and performance metrics, notably accuracy, precision, recall, F-1 scores and confusion matrix.

Figure 1 : Neural Network Architecture



Artificial Neural Network

The concept of an Artificial Neural Network (ANN) is inspired by the neurons in the human brain. Similar to some degree, an Artificial Neural Network is made up of layers of nodes. Our neural network consist of an input layer, one hidden layer, and an output layer.

The first stage is **defining our Neural Network**. This includes defining the number of nodes in each layer, the activation function for each node and loss function in our output layer. All weights and thresholds are initially set to a random value between -1 to 1.

In terms of dataset preprocessing, we adopted **K-fold validation**, where K is set to 5 in Dataset_1 and 2 in Dataset_2. We also performed one-hot encoding to the labels.

Forward Propagation

We define the forward propagation step of our network as follows:

Given input batch $x \in R^{B \times N}$, where B is batch size and N is the input feature dimension, we randomly initialize weight matrix $W_1 \in R^{N \times M}$ and bias vectors $B_0 \in R^M$ in the range of $[-1, 1]$, where M denotes the hidden layer size. Then we compute the hidden layer $h_1 \in R^{B \times M}$ using the following equation:

$$h_1 = \sigma_1(XW_1^T + B_0)$$

σ_1 is an activation function that maps any real valued input logistically, and in our work we implemented both sigmoid and ReLU activation functions.

$$sigmoid(x) = \frac{1}{1 + e^{-x}}$$
$$ReLU(x) = max(0, x)$$

B_0 is broadcasted into $R^{B \times M}$ to perform element-wise addition.

Since we only implement single hidden layer in our experiments, we then compute the output label probabilities $\hat{y} \in R^{B \times Z}$ where Z is the count of possible labels (all of the y is $1 \; of \; Z$ encoded) as follow:

$$\hat{y} = softmax(h_1 W_2 + B_1)$$
$$softmax(x) = \frac{e^x}{\Sigma e^{x_i}}$$

In this formula, $W_2 \in R^{M \times Z}$, $B_1 \in R^Z$ are also parameters randomly initialized between $[-1, 1]$. *(Please refer*

https://github.com/shawn3298317/CS640_Artificial_Intelligence/blob/master/pa1/nn.py#L104 for detail implementation of the forward computation step.)

In terms of optimizing the model parameters, we try to maximize the likelihood of observations being correctly classified. The cost function is defined as follows:

$$J(\theta) = -\frac{1}{N}\sum_{k=1}^{K} y\ log(\hat{y}) + \frac{\lambda}{N}||\theta||^2$$

This is the sum of the cross entropy loss and L2 regularization term. In our experiments, we tested with different values of λ to see how different values of λ affects our performance.

Backward Propagation

We implemented both **stochastic gradient descent (min-batch)** and **vanilla gradient descent** in our work. We also implemented back propagation algorithm so that we can calculate the gradients of the cost function with respect to model parameters much faster. During backward propagation, we utilized chain rule and kept track of the gradient of each intermediate parameters to avoid computing same derivative multiple times. *(Please refer https://github.com/shawn3298317/CS640_Artificial_Intelligence/blob/master/pa1/nn.py#L132 for detail implementation of the backward computation step.)*

The pseudo code of the back propagation algorithm is shown below:

Figure 2 : Backpropagation Algorithm

After the forward computation, compute the gradient on the output layer:
$\boldsymbol{g} \leftarrow \nabla_{\hat{\boldsymbol{y}}} J = \nabla_{\hat{\boldsymbol{y}}} L(\hat{\boldsymbol{y}}, \boldsymbol{y})$
for $k = l, l-1, \ldots, 1$ **do**
  Convert the gradient on the layer's output into a gradient into the pre-nonlinearity activation (element-wise multiplication if f is element-wise):
  $\boldsymbol{g} \leftarrow \nabla_{\boldsymbol{a}^{(k)}} J = \boldsymbol{g} \odot f'(\boldsymbol{a}^{(k)})$
  Compute gradients on weights and biases (including the regularization term, where needed):
  $\nabla_{\boldsymbol{b}^{(k)}} J = \boldsymbol{g} + \lambda \nabla_{\boldsymbol{b}^{(k)}} \Omega(\theta)$
  $\nabla_{\boldsymbol{W}^{(k)}} J = \boldsymbol{g}\, \boldsymbol{h}^{(k-1)\top} + \lambda \nabla_{\boldsymbol{W}^{(k)}} \Omega(\theta)$
  Propagate the gradients w.r.t. the next lower-level hidden layer's activations:
  $\boldsymbol{g} \leftarrow \nabla_{\boldsymbol{h}^{(k-1)}} J = \boldsymbol{W}^{(k)\top} \boldsymbol{g}$
end for

Experiments

The neural networks were fitted on the linear, non linear and multiclass classification datasets. A logging system was built to observe the loss, epoch number and other metrics. Different settings were designed for the logger to output varying levels of information. Tests performed include hyperparameter optimization and boundary case checks. The hyper parameters modified are as given below.

Table 1 : Hyperparameters used

NNodes	The number of nodes in the hidden layer
activate	Activation function for the hidden layer
deltaActivate	The derivative of the activation function
learningRate	The length in the direction of gradient update
epochs	The total iterations
regLambda	The L2 regularizer
batchSize	The size of input batch
task	The type of the learning task

A test environment is as given below.

Table 2 : Development/Test environment description

OS	OSX, Windows10
Python	3.6+
Libraries used	Numpy, Pandas

The metrics used are as given below, where TP: True Positive, TN: True Negatives, FP: False Positives, FN: False Negatives,

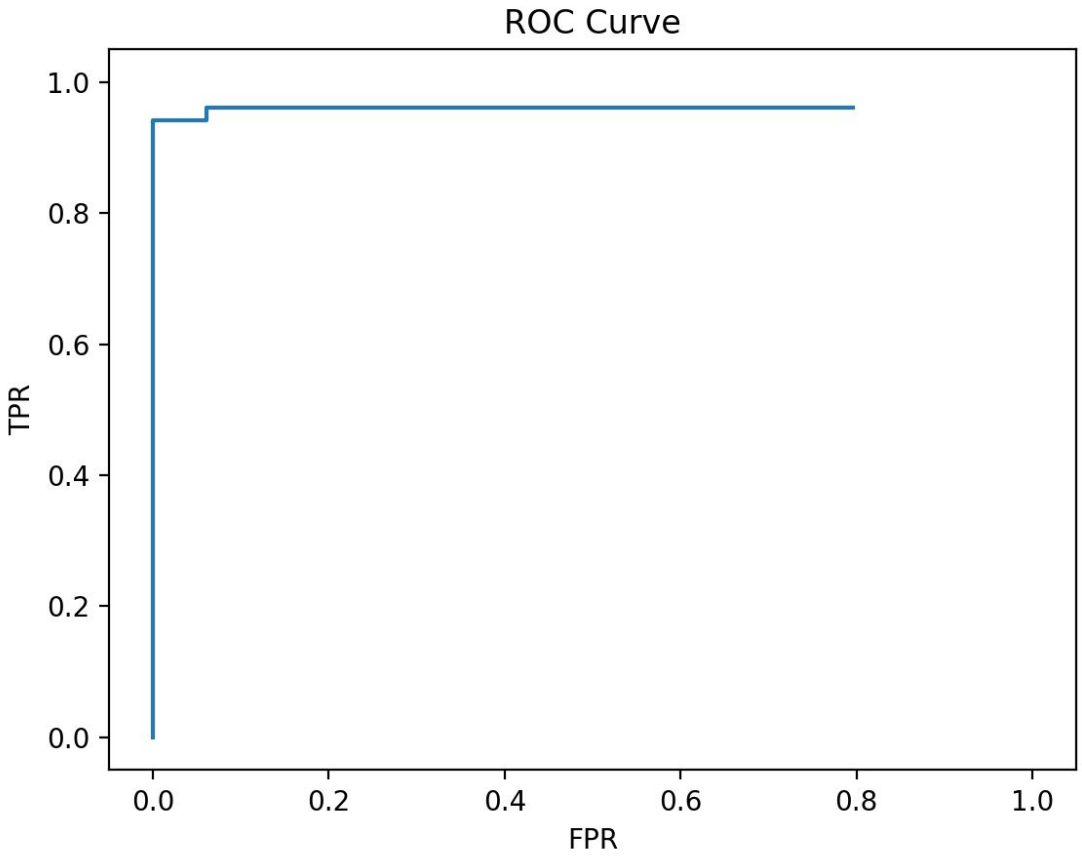
Table 3: Evaluation metrics

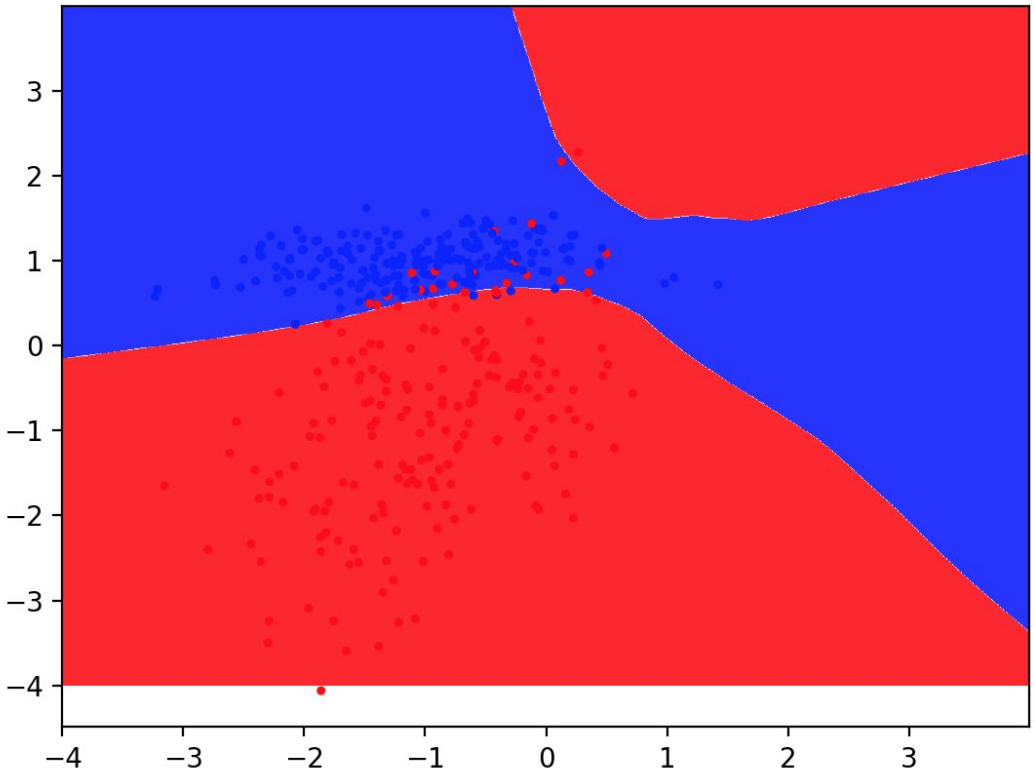
Metric	Mathematical Equation	Explanation
Accuracy	$\frac{(TP + TN)}{\#\text{ number of samples}}$	Fraction of correct predictions
Precision	TP / (TP + FP)	Fraction of positive predictions which were correct
Recall	TP / (TP + FN)	Number of positive samples predicted as positive
F1	$\frac{2 \times \text{Precision} \times \text{Recall}}{(\text{Precision} + \text{Recall})}$	The harmonic mean of precision and recall expressing the overall classification capability of the model

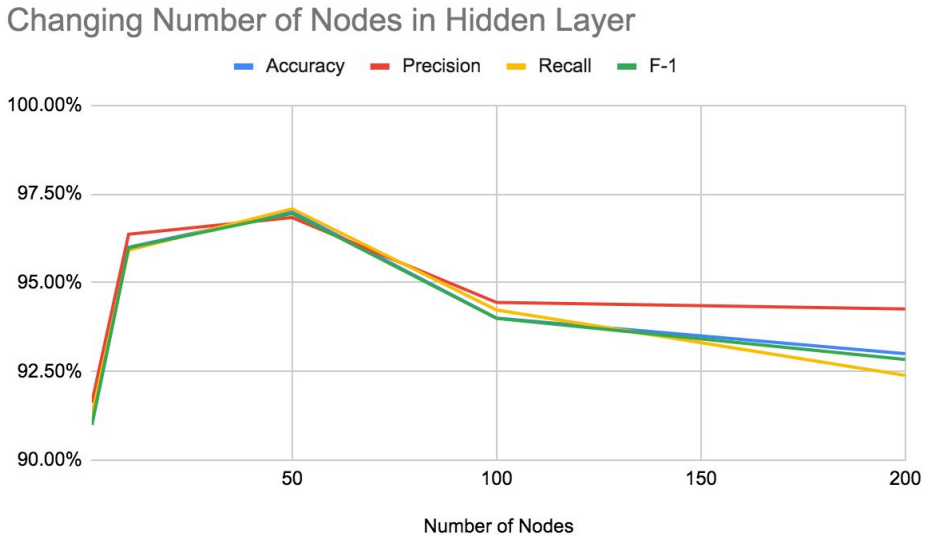
Additionally the confusion matrix, decision boundaries and ROC curve were also plotted. We implemented grid search on the hyperparameters to find the optimal values. We ran approximately 5 different values for each of the 4 hyperparameters: number of nodes in the hidden layer, regularization, learning rate and epochs.

Results

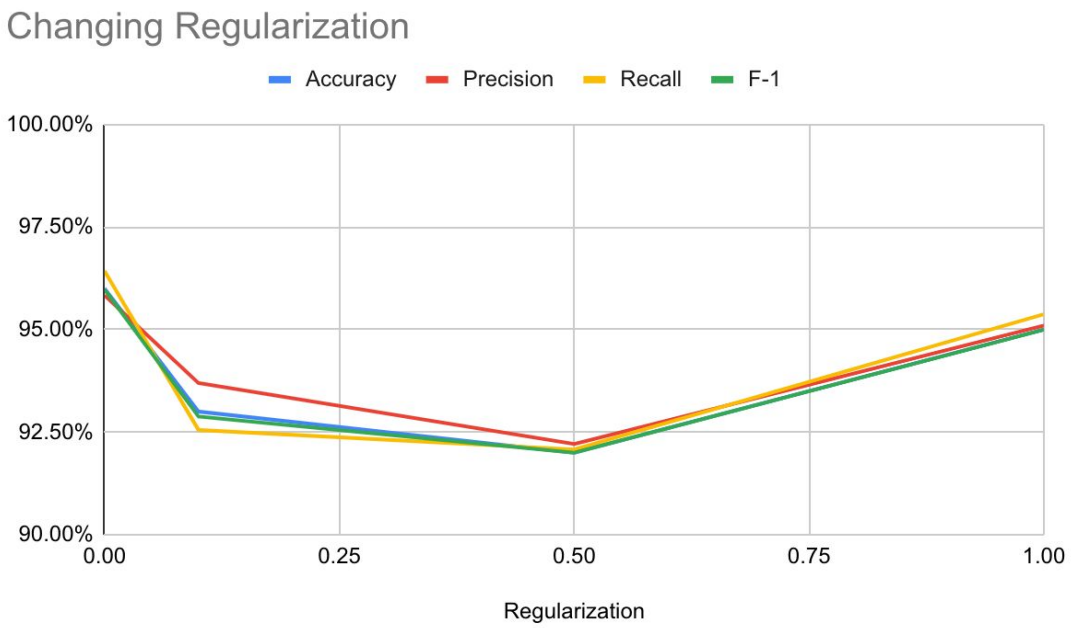
Part 1a: Dataset 1 (Linear Data)


ROC Curve
TPR
FPR
0.0
0.2
0.4
0.6
0.8
1.0



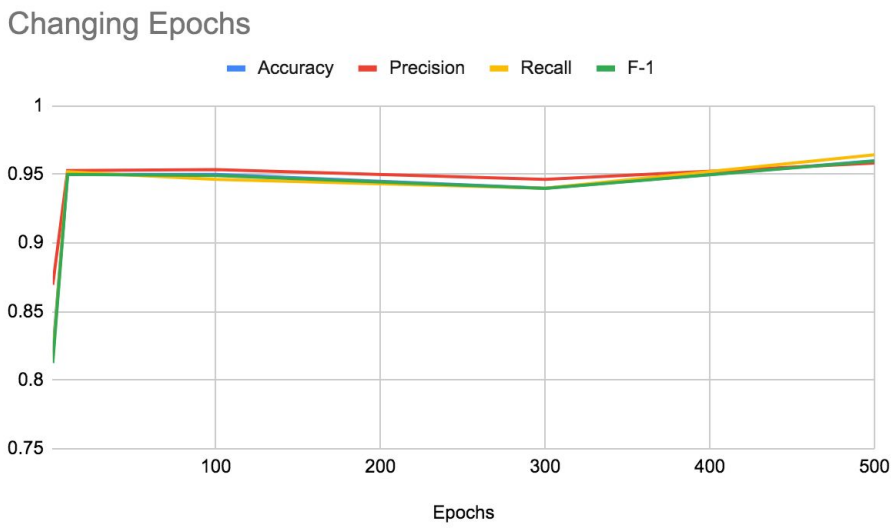


As shown above, the optimal number of nodes is 50. A reduced performance when less than 50 could be a result of underfitting. A reduced performance when more than 50 could be a result of overfitting.



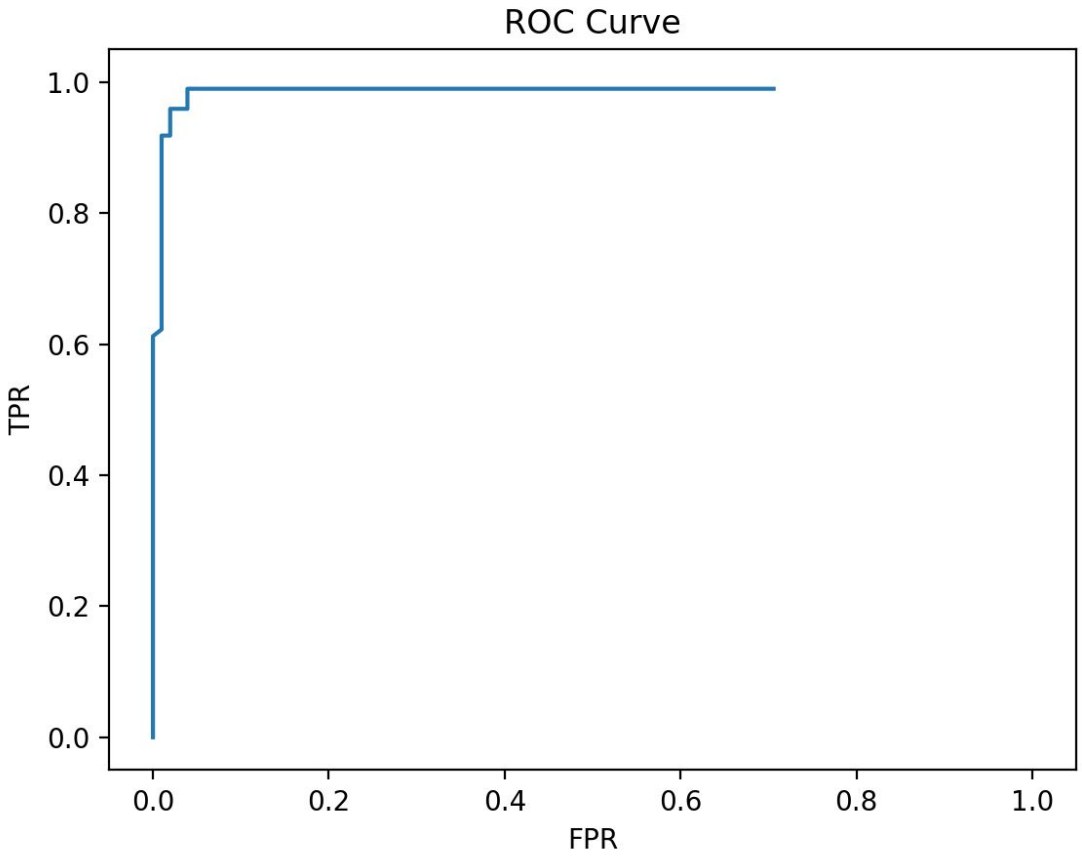
As shown above, proper regularization helps our network generalize better to testing set, but if we increase regularization too much, our performance drops.

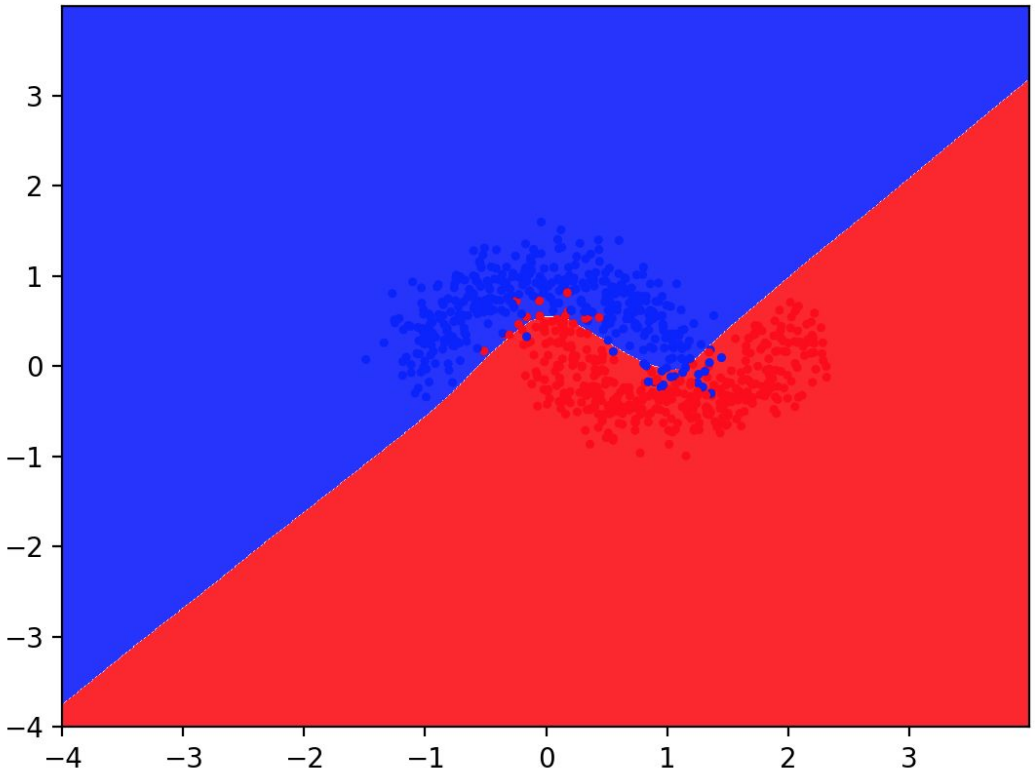
As shown above, the optimal learning rate is around 0.25. Since we're keeping a batch size of 20 during our experiment, if we increase learning rate too much, it's likely that the model misses the local minimum point.

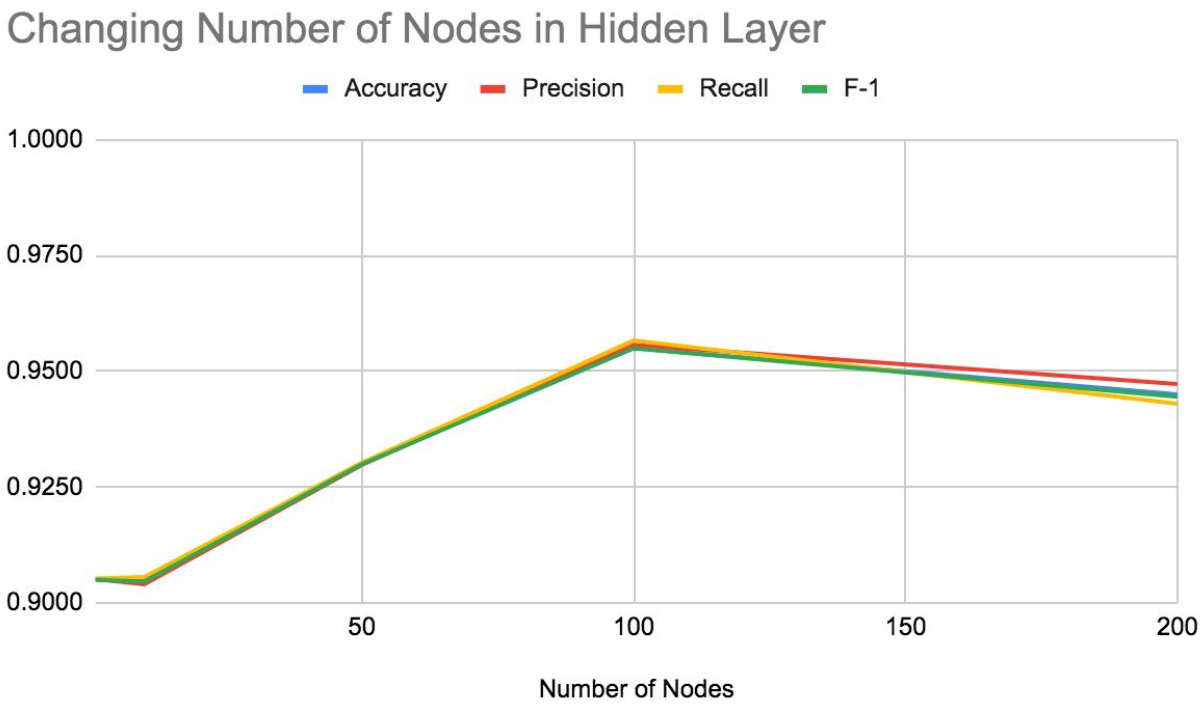


As shown above, we need at least 10 epochs for the accuracy to be close to optimal. But too with too many training epoch, our network tends to overfit.

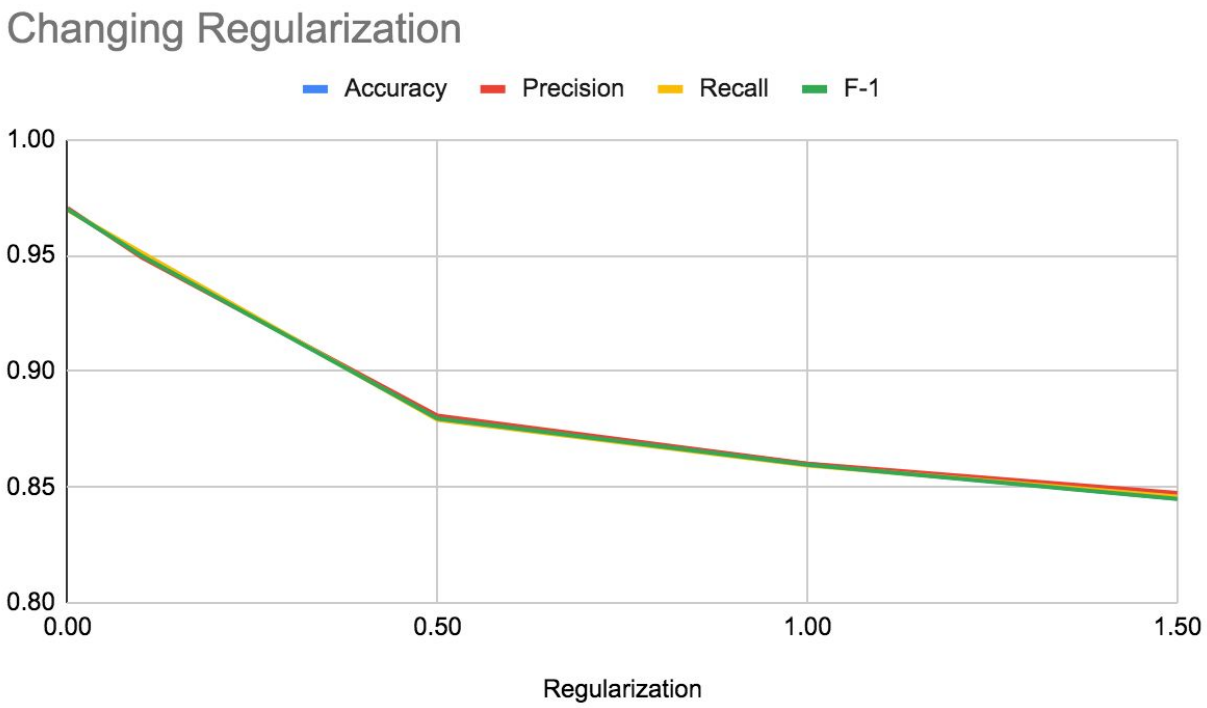
Part 1b: Dataset 1 (Non-Linear Data)



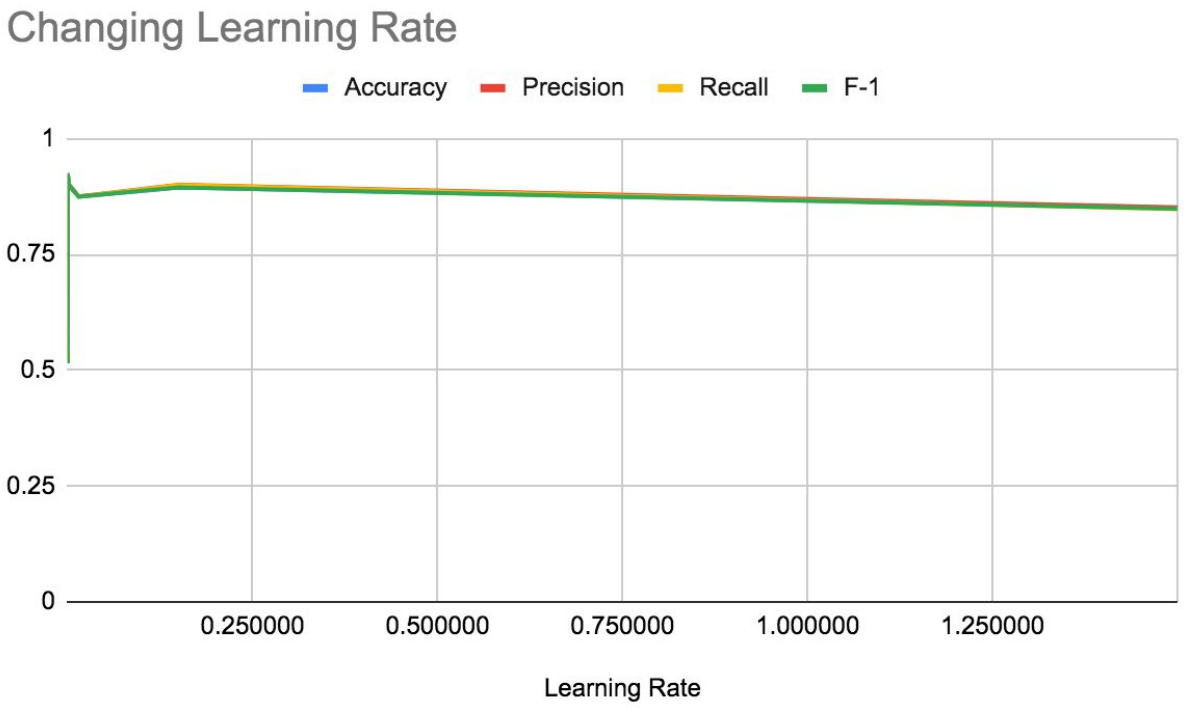

3
2
1
0
−1
−2
−3
−4
−4
−3
−2
−1
0
1
2
3

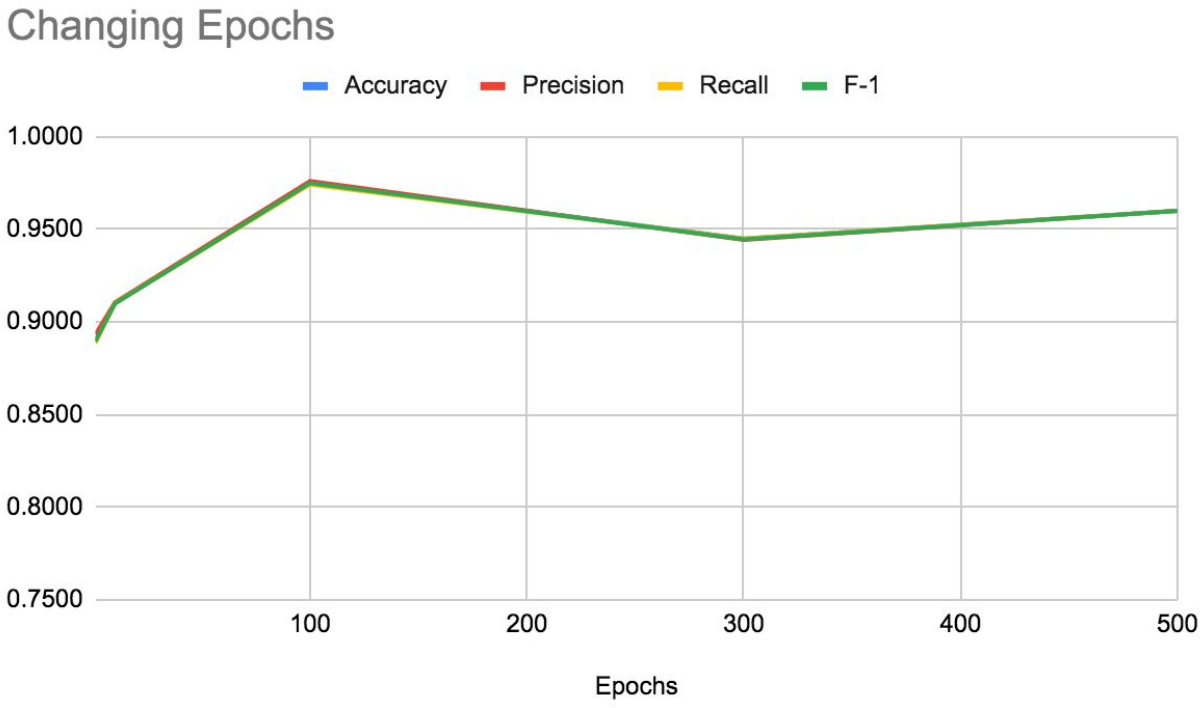


As shown above, the optimal number of nodes is 100. It is possible to underfit when below 100, and overfit when above 100.

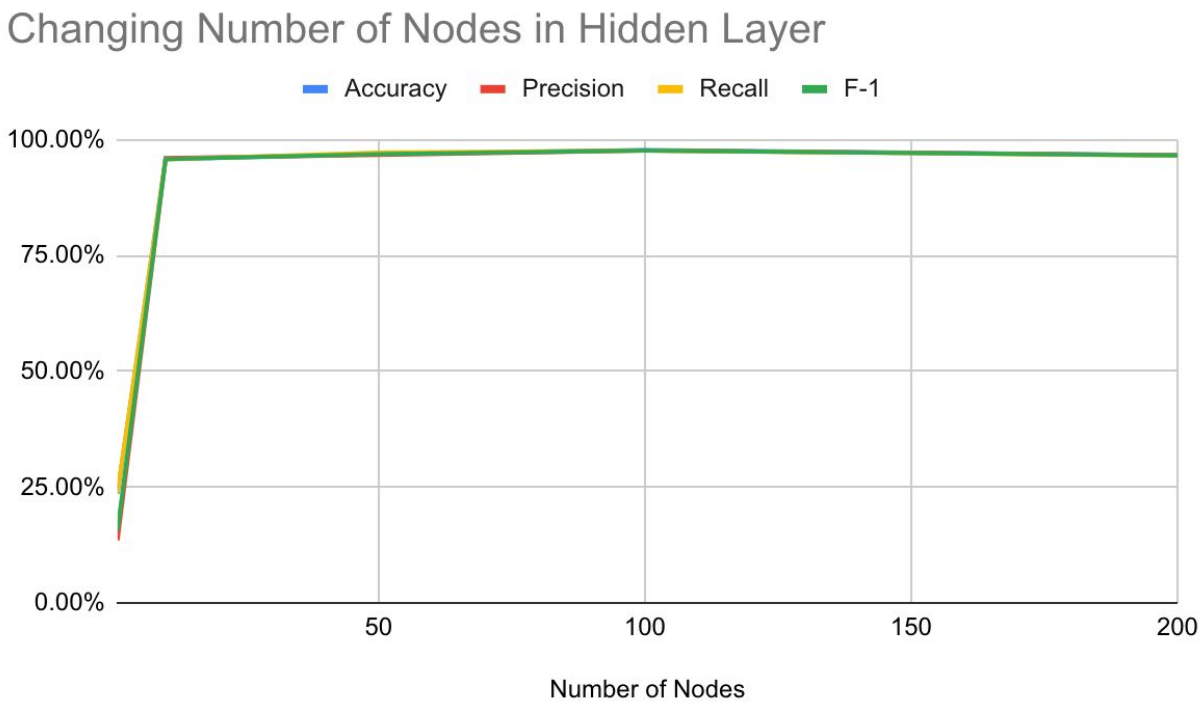


As shown above, the performance is optimal when regularization is set to 0. This indicates that we are probably underfitting or at optimal.

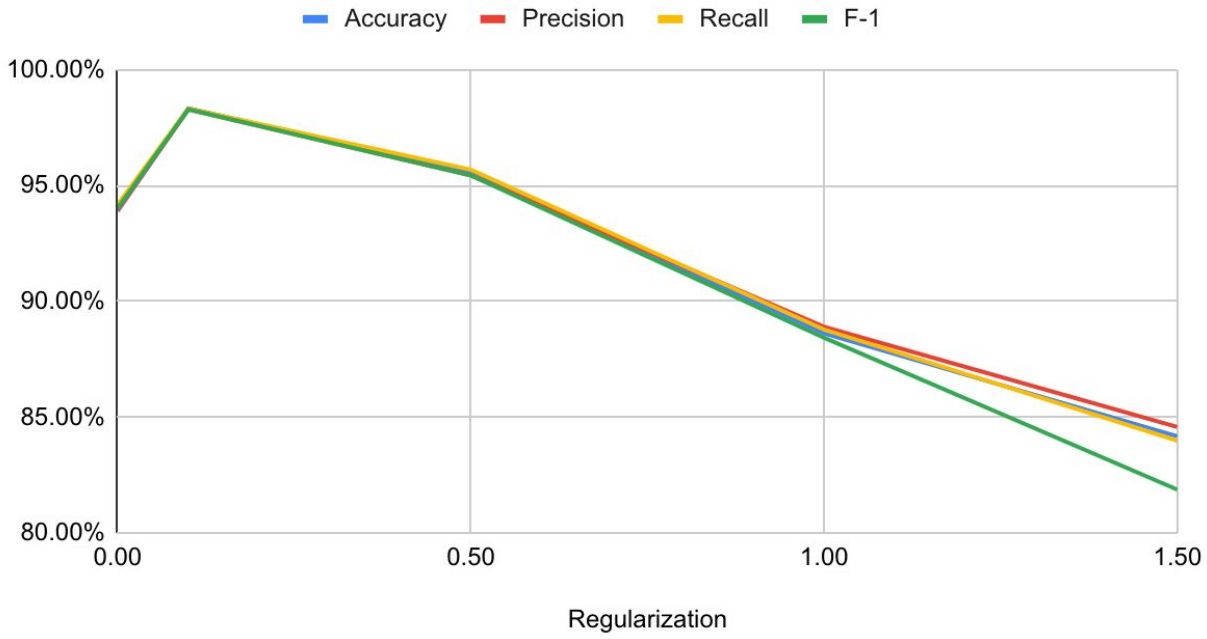




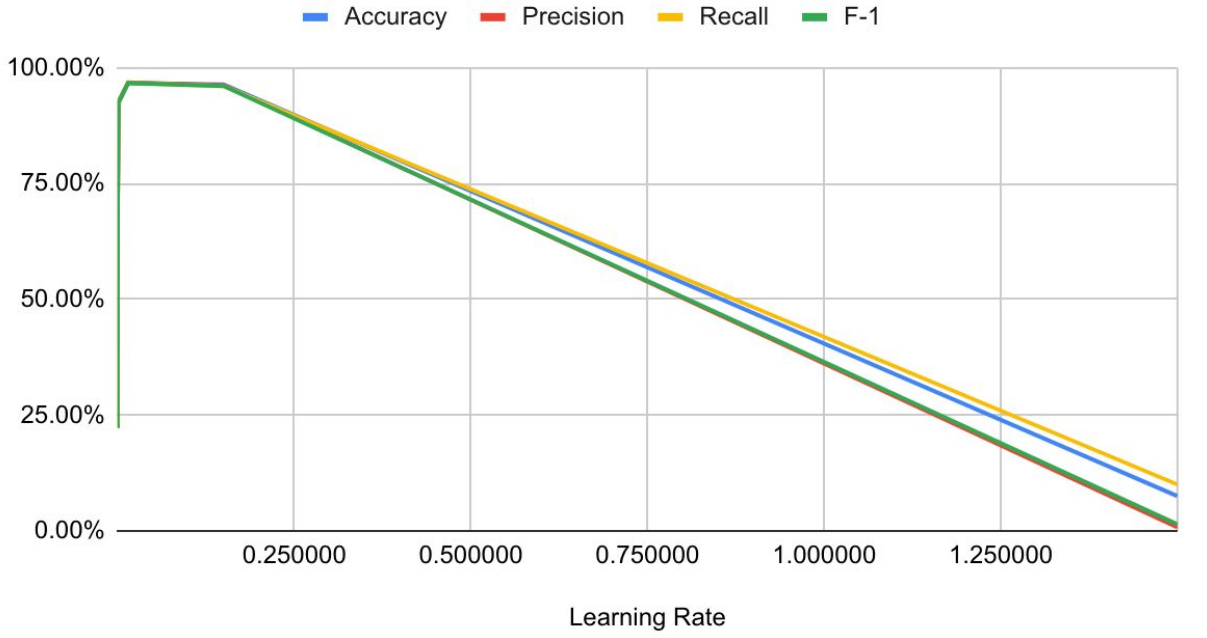
Part 2: Dataset 2

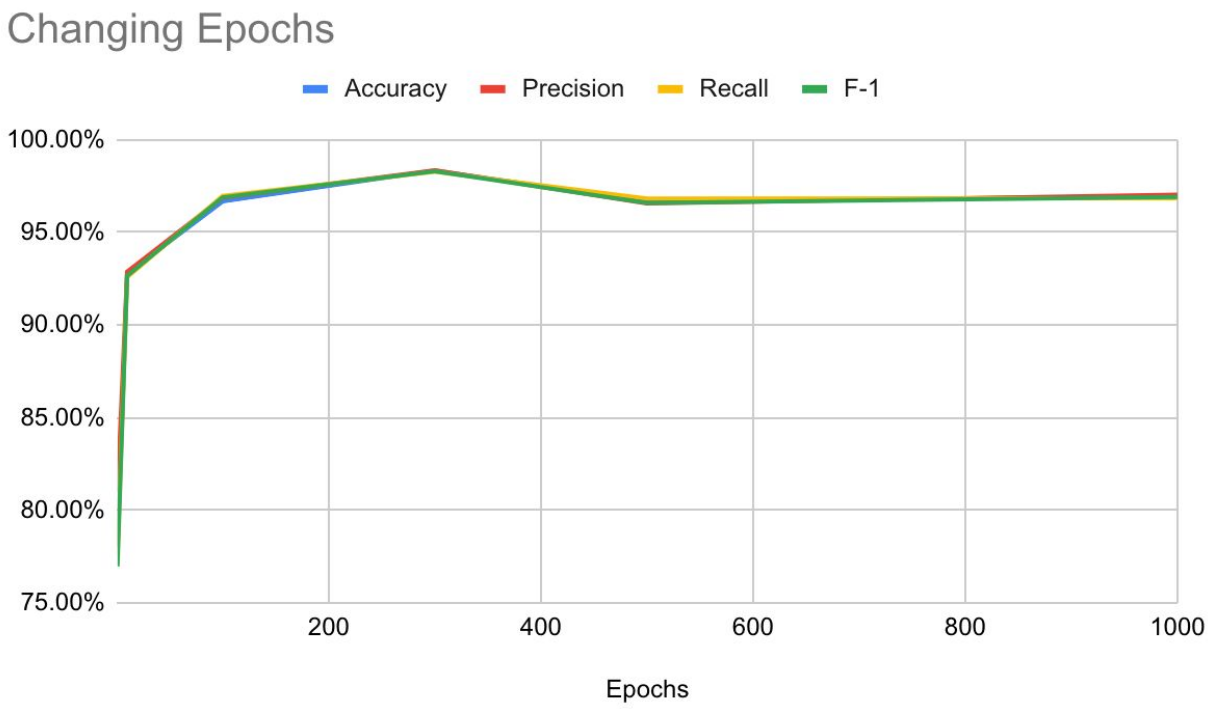


Changing Regularization



Changing Learning Rate





Discussion

The advantage of Neural Network is that it can fit any continuous function. However, the computation power required can be massive. Therefore, without GPU, it can be difficult to train a deep neural network. Based on our performance metrics (obtained from running our test sets), we concluded that our neural network is generally successful in both datasets provided.

During our experiments, we had suspecisions on our expectations and thus, we ran different values of the hyperparameters to see if performance of neural network degrades as a result of overfitting. Our experiment results is evidence that our Neural Network is vulnerable to overfitting when the hyperparameters are not tuned correctly.

There are several things that we could improve in our current training strategy:

1. Implement RMSProp, AdaGrad and other optimizers to achieve a more flexible learning rate, so as to increase our chance of finding the global minimum.
2. Increase the hidden layers, since the deeper the network is, the more efficient it gets in terms of fitting other functions.
3. Normalize dataset before we start training it.

We concluded that the numbers (in theory) can change the performance of our neural network, including time costs, for the following reasons:

1. If we change to different optimizer, the model might be able to converge faster, so we can achieve better results given same training epochs.
2. If we increase hidden layers, it'll take extra time to forward/backward propagate.
3. If we normalize the dataset beforehand, the model can also converge faster.

Conclusions

In conclusion, we built a neural network library which supports building, training and testing neural network. We allow the users to easily tune parameters, such as number of layers, number of neurons in each layer, activation function, learning rate to fit their use cases better.

Based on the experiments, it was found that hyper parameters of the model should be tuned carefully to prevent overfitting. According to our experiments, a model with 1,000 neurons is likely to overfit, but a model with around 100 neurons worked pretty good. In general, it was observed that hyperparameter optimization is an important task to maximize the performance of neural networks.

Credits and Bibliography

- Figure 1 - https://medium.com/@aerinykim/derive-the-gradients-w-r-t-the-inputs-to-an-one-hidden-layer-neural-network-fb24ed1ed05f
- Figure 2 - Course material from CS542 Machine Learning.

The code, the math derivation behind the code, visualization and metric calculation was performed by all 3 members.

Appendix

Codes are available on github:
https://github.com/shawn3298317/CS640_Artificial_Intelligence/tree/master/pa1

Part 1a: Dataset 1 (Linear Data)

Number of Nodes	Accuracy	Precision	Recall	F-1	Confusion Matrix
1	91.00%	91.62%	91.27%	90.99%	[[47. 1.][8. 44.]]

10	96.00%	96.36%	95.92%	95.99%	[[51. 0.][4. 45.]]
50	97.00%	96.83%	97.08%	96.95%	[[55. 2.][1. 42.]]
100	94.00%	94.44%	94.23%	94.00%	[[48. 0.][6. 46.]]
200	93.00%	94.26%	92.39%	92.84%	[[54. 0.][7. 39.]]
Regularization	Accuracy	Precision	Recall	F-1	Confusion Matrix
0.00	96.00%	95.83%	96.43%	95.97%	[[44. 0.][4. 52.]]
0.10	93.00%	93.70%	92.55%	92.88%	[[53. 1.][6. 40.]]
0.50	92.00%	92.21%	92.08%	92.00%	[[47. 2.][6. 45.]]
1.00	95.00%	95.10%	95.37%	95.00%	[[46. 0.][5. 49.]]
1.50	95.00%	95.10%	95.37%	95.00%	[[46. 0.][5. 49.]]
Learning Rate	Accuracy	Precision	Recall	F-1	Confusion Matrix
0.000015	82.00%	82.53%	82.53%	82.00%	[[41. 5.][13. 41.]]
0.000150	93.00%	93.75%	92.42%	92.84%	[[54. 1.] [6. 39.]]
0.001500	92.00%	91.85%	92.12%	91.95%	[[42. 3.][5. 50.]]
0.015000	93.00%	94.17%	92.55%	92.88%	[[53. 0.][7. 40.]]
0.150000	95.00%	95.21%	94.94%	94.99%	[[50. 1.][4. 45.]]
1.5000	92.00%	93.22%	91.84%	91.92%	[[51. 0.][8. 41.]]
15.0000	51.00%	25.50%	50.00%	33.77%	[[51. 0.][49. 0.]]
Epochs	Accuracy	Precision	Recall	F-1	Confusion Matrix
1	0.82	0.8695652174	0.8163265306	0.8125	[[51. 0.][18. 31.]]
10	0.95	0.9528301887	0.9519230769	0.9499949995	[[48. 0.][5. 47.]]
100	0.95	0.9536124795	0.9464646465	0.949140474	[[54. 1.][4. 41.]]
300	0.94	0.9464285714	0.94	0.9397832196	[[50. 0.][6. 44.]]
500	0.96	0.9583333333	0.9642857143	0.959742351	[[44. 0.][4. 52.]]

Part 1b: Dataset 1 (Non-Linear Data)

Number of Nodes	Accuracy	Precision	Recall	F-1	Confusion Matrix
1	0.9050	0.9051	0.9053	0.9050	[[90. 8.][11. 91.]]
10	0.905	0.9040100251	0.9055958132	0.904596922	[[97. 11.][8. 84.]]
50	0.93	0.9297719088	0.9302884615	0.9299369432	[[96. 8.][6. 90.]]
100	0.955	0.9556822729	0.9566	0.9550	[[97. 8.][1. 94.]]
200	0.945	0.9472402597	0.9429705557	0.9444991044	[[85. 8.][3. 104.]]
Regularization	Accuracy	Precision	Recall	F-1	Confusion Matrix
0.00	0.97	0.9704937776	0.9696	0.9699	[[102. 2.][4. 92.]]
0.10	0.95	0.9491185897	0.9512882448	0.9498193497	[[89. 3.][7. 101.]]
0.50	0.8800	0.8808	0.8790064103	0.8795664392	[[82. 14.][10. 94.]]
1.00	0.86	0.8600963468	0.859375	0.8596491228	[[91. 13.][15. 81.]]
1.50	0.845	0.8474022711	0.8458383353	0.8449030644	[[87. 11.][20. 82.]]
Learning Rate	Accuracy	Precision	Recall	F-1	Confusion Matrix
0.000015	0.515	0.5165596146	0.5165413534	0.5149878747	[[52. 43.][54. 51.]]
0.000150	0.925	0.9250626566	0.9246794872	0.9248478168	[[97. 7.][8. 88.]]
0.001500	0.9	0.901905434	0.8990091082	0.8996386993	[[96. 7.][13. 84.]]
0.015000	0.875	0.875400641	0.8748374837	0.8749218261	[[85. 14.][11. 90.]]
0.150000	0.895	0.9003623188	0.8997588182	0.8949973749	[[90. 3.][18. 89.]]

1.5000	0.85	0.851461039	0.8476515456	0.8487750781	[[76. 18.][12. 94.]]
15.0000	0.53	0.265	0.5	0.3464052288	[[106. 0.][94. 0.]]
Epochs	Accuracy	Precision	Recall	F-1	Confusion Matrix
1	0.8900	0.8931	0.8887	0.8895	[[96. 7.][15. 82.]]
10	0.9100	0.9106570513	0.91	0.9099639856	[[93. 7.][11. 89.]]
100	0.975	0.9760466989	0.9741854637	0.9748939268	[[91. 4.][1. 104.]]
300	0.945	0.9442431562	0.9450045367	0.9445997331	[[86. 5.][6. 103.]]
500	0.96	0.9599959996	0.9599959996	0.9600	[[95. 4.][4. 97.]]

Part 2: Dataset 2

Number of Nodes	Accuracy	Precision	Recall	F-1	Confusion Matrix
1	23.61%	13.54%	23.79%	15.58%	[[26., 0., 3., 0., 3., 0., 0., 0., 1., 0.], [4., 0., 13., 1., 3., 0., 0., 0., 8., 6.], [3., 0., 11., 0., 5., 0., 0., 0., 7., 8.], [0., 0., 1., 9., 0., 0., 0., 21., 2., 2.], [23., 0., 2., 0., 2., 0., 0., 0., 1., 2.], [0., 0., 2., 9., 0., 0., 0., 18., 4., 5.], [42., 0., 1., 0., 1., 0., 0., 0., 0.,

					0.], [0., 0., 0., 6., 0., 0., 0., 30., 0., 1.], [2., 0., 13., 9., 1., 0., 0., 0., 6., 3.], [0., 0., 2., 12., 0., 1., 0., 1., 3., 21.]]
10	95.83%	96.02%	95.75%	95.84%	[[35., 0., 0., 0., 0., 1., 1., 0., 0., 0.], [0., 44., 0., 0., 0., 0., 0., 0., 1., 0.], [0., 0., 34., 0., 0., 0., 0., 0., 0., 0.], [1., 0., 0., 26., 0., 0., 0., 0., 3., 1.], [0., 1., 0., 0., 30., 0., 0., 0., 1., 0.], [0., 0., 0., 0., 0., 29., 0., 0., 0., 0.], [0., 0., 0., 0., 0., 0., 44., 0., 0., 0.], [0., 0., 0., 0., 0., 0., 0., 43., 0., 0.], [0., 1., 0., 0., 0., 0., 0., 0., 31., 0.], [0., 0., 0., 0., 0., 0., 0., 0., 1., 32.]]

50	96.94%	96.73%	97.19%	96.89%	[[39., 0., 0., 0., 0., 0., 0., 0., 0., 0.], [0., 33., 0., 0., 0., 0., 0., 0., 0., 1.], [0., 0., 36., 0., 0., 0., 0., 0., 0., 0.], [0., 0., 0., 35., 0., 0., 0., 0., 1., 0.], [0., 0., 0., 0., 34., 0., 0., 0., 1., 0.], [0., 0., 0., 0., 0., 36., 0., 0., 0., 0.], [0., 1., 0., 0., 0., 0., 39., 0., 0., 0.], [0., 0., 0., 0., 0., 0., 0., 43., 0., 0.], [0., 1., 0., 0., 0., 1., 0., 0., 29., 1.], [0., 0., 0., 1., 0., 1., 0., 0., 0., 27.]]
100	97.78%	97.69%	97.67%	97.65%	[[43., 0., 0., 0., 0., 1., 0., 0., 0., 0.], [0., 37., 0., 0., 0., 0., 0., 0., 0., 0.], [0., 0., 24., 0., 0., 0., 0., 0., 0., 0.], [0., 0., 0., 33., 0., 1., 0., 0., 0., 0.], [0., 0., 0., 0., 40., 0., 0., 1., 0., 0.], [0., 0., 0., 0., 0., 33., 1., 0.,

					0., 0.], [0., 2., 0., 0., 0., 0., 36., 0., 0., 0.], [0., 0., 0., 0., 0., 0., 0., 37., 1., 1.], [0., 1., 0., 0., 0., 0., 0., 0., 34., 0.], [0., 1., 0., 0., 0., 0., 0., 0., 0., 33.]]
200	96.67%	96.67%	96.56%	96.60%	[[40., 0., 0., 0., 0., 0., 0., 0., 0., 0.], [0., 38., 0., 0., 0., 1., 0., 0., 0., 0.], [0., 1., 27., 0., 0., 0., 0., 0., 0., 0.], [0., 0., 0., 28., 0., 0., 0., 0., 0., 0.], [0., 0., 0., 0., 38., 0., 0., 0., 0., 0.], [0., 0., 0., 0., 0., 33., 1., 0., 0., 1.], [0., 0., 0., 0., 0., 0., 44., 0., 0., 0.], [0., 0., 0., 0., 0., 0., 0., 34., 0., 1.], [0., 0., 0., 0., 0., 1., 0., 0., 34., 0.], [0., 0., 0., 0., 0., 0., 0., 0., 1., 37.]]
Regularization	Accuracy	Precision	Recall	F-1	Confusion Matrix

0.00	93.89%	93.94%	94.17%	94.01%	[[39., 0., 0., 0., 0., 0., 0., 0., 0., 0.], [0., 38., 0., 0., 0., 0., 0., 0., 0., 0.], [0., 0., 28., 0., 0., 0., 0., 0., 0., 0.], [0., 0., 0., 35., 0., 0., 0., 1., 0., 0.], [0., 0., 0., 0., 36., 0., 0., 0., 1., 0.], [0., 0., 0., 0., 0., 33., 0., 0., 0., 1.], [0., 0., 0., 0., 0., 0., 35., 0., 1., 0.], [0., 0., 0., 0., 0., 0., 0., 39., 0., 0.], [0., 0., 0., 0., 0., 0., 0., 0., 34., 0.], [0., 0., 0., 0., 0., 1., 0., 1., 0., 37.]]
0.10	98.33%	98.29%	98.33%	98.28%	[[34., 0., 0., 0., 0., 0., 0., 0., 0., 0.], [0., 39., 0., 0., 0., 0., 0., 0., 0., 0.], [0., 0., 39., 0., 0., 0., 0., 0., 0., 0.], [0., 0., 0., 41., 0., 0., 0., 0., 1., 0.], [0., 0., 0., 0., 31., 0., 0., 0., 0., 0.], [0., 0., 0., 0., 0., 38., 0., 0.,

					0., 0.], [0., 0., 0., 0., 0., 0., 39., 0., 0., 0.], [0., 0., 0., 0., 0., 0., 0., 27., 1., 1.], [0., 2., 0., 0., 0., 0., 1., 0., 27., 0.], [0., 0., 0., 0., 0., 0., 0., 0., 0., 39.]]
0.50	95.56%	95.44%	95.69%	95.44%	[[41., 0., 0., 0., 0., 2., 0., 0., 0., 0.], [0., 29., 0., 0., 0., 0., 0., 0., 0., 3.], [0., 0., 35., 0., 0., 0., 0., 0., 0., 0.], [0., 0., 0., 27., 0., 1., 0., 1., 0., 1.], [0., 1., 0., 0., 29., 0., 0., 0., 0., 0.], [0., 1., 0., 0., 1., 34., 0., 0., 0., 0.], [1., 0., 0., 0., 0., 0., 43., 0., 0., 0.], [0., 0., 0., 0., 0., 0., 0., 38., 0., 0.], [0., 1., 0., 0., 0., 0., 0., 0., 39., 2.], [0., 0., 0., 0., 1., 1., 0., 1., 1., 26.]]

1.00	88.61%	88.89%	88.78%	88.42%	[[34., 0., 0., 0., 0., 0., 0., 0., 0., 0.], [0., 39., 0., 0., 0., 0., 0., 0., 0., 2.], [0., 2., 28., 0., 0., 0., 0., 0., 0., 0.], [0., 0., 0., 30., 0., 0., 0., 0., 0., 0.], [0., 3., 0., 0., 33., 0., 0., 0., 1., 0.], [0., 0., 0., 0., 0., 45., 0., 0., 0., 0.], [0., 0., 0., 0., 0., 0., 29., 0., 0., 0.], [0., 0., 0., 0., 1., 0., 0., 30., 1., 0.], [0., 7., 0., 0., 0., 1., 0., 0., 29., 1.], [0., 1., 0., 2., 1., 1., 0., 1., 0., 38.]]
1.50	84.17%	84.57%	83.98%	81.87%	[[35., 0., 0., 0., 0., 0., 0., 0., 0., 0.], [0., 27., 4., 0., 0., 0., 1., 0., 0., 1.], [0., 3., 29., 0., 0., 0., 0., 0., 0., 0.], [0., 0., 3., 34., 0., 1., 0., 2., 1., 1.], [0., 2., 0., 0., 28., 0., 0., 0., 0., 0.], [0., 0., 0., 0., 0., 35., 1., 0.,

					0., 0.], [0., 0., 0., 0., 1., 0., 43., 0., 0., 0.], [0., 0., 0., 0., 1., 0., 0., 46., 0., 0.], [0., 9., 0., 0., 0., 1., 0., 1., 18., 4.], [0., 1., 0., 0., 1., 0., 0., 1., 0., 25.]]
Learning Rate	Accuracy	Precision	Recall	F-1	Confusion Matrix
0.000015	22.50%	22.36%	22.42%	22.18%	[[26., 0., 1., 0., 5., 0., 2., 0., 1., 1.], [2., 2., 0., 0., 9., 13., 3., 9., 3., 0.], [2., 12., 0., 0., 1., 0., 3., 2., 14., 1.], [1., 6., 0., 0., 0., 2., 3., 10., 4., 3.], [4., 1., 10., 4., 9., 1., 1., 2., 4., 0.], [0., 4., 10., 0., 6., 14., 0., 2., 0., 2.], [4., 0., 5., 2., 6., 1., 0., 2., 21., 0.], [0., 1., 10., 0., 1., 0., 24., 4., 2., 0.], [5., 0., 1., 0., 2., 9., 4., 7., 0., 0.], [2., 9., 1., 0., 5., 6., 2., 8., 0., 1.]]

0.000150	71.67%	72.24%	71.64%	71.64%	[[38., 1., 0., 0., 0., 0., 1., 0., 0., 1.], [0., 25., 2., 3., 0., 1., 0., 2., 0., 0.], [1., 1., 19., 2., 0., 2., 2., 2., 1., 2.], [0., 1., 0., 30., 0., 4., 0., 2., 1., 2.], [1., 2., 0., 0., 22., 5., 1., 2., 1., 2.], [0., 1., 1., 2., 1., 23., 1., 0., 3., 3.], [1., 0., 0., 0., 1., 1., 35., 0., 0., 1.], [1., 2., 0., 1., 3., 1., 2., 22., 5., 1.], [0., 2., 1., 4., 0., 4., 0., 3., 17., 1.], [0., 0., 0., 3., 0., 2., 0., 0., 2., 27.]]
0.001500	92.50%	92.71%	92.55%	92.58%	[[41., 0., 1., 0., 0., 0., 0., 0., 0., 1.], [0., 30., 2., 0., 2., 0., 0., 0., 1., 3.], [0., 0., 32., 0., 0., 0., 0., 0., 0., 0.], [0., 0., 0., 29., 0., 1., 0., 0., 1., 2.], [0., 0., 0., 0., 34., 0., 0., 1., 1., 0.], [0., 0., 0., 0., 0., 32., 0., 0.,

					0., 1.], [0., 1., 0., 0., 1., 0., 32., 0., 1., 0.], [0., 0., 0., 0., 0., 0., 0., 41., 0., 0.], [0., 1., 1., 1., 0., 1., 0., 0., 31., 0.], [1., 0., 1., 1., 0., 1., 0., 1., 2., 27.]]
0.015000	96.67%	96.79%	96.72%	96.68%	[[34., 0., 0., 0., 0., 0., 0., 0., 0., 0.], [1., 39., 0., 0., 0., 0., 0., 0., 0., 0.], [0., 0., 35., 0., 0., 0., 0., 0., 0., 0.], [0., 0., 0., 40., 0., 0., 0., 0., 1., 0.], [0., 0., 0., 0., 30., 0., 0., 0., 0., 0.], [0., 0., 1., 0., 0., 22., 0., 0., 0., 1.], [0., 0., 0., 0., 0., 1., 35., 0., 0., 0.], [0., 0., 0., 0., 0., 0., 0., 39., 1., 0.], [0., 2., 0., 0., 0., 2., 0., 0., 40., 0.], [0., 0., 0., 0., 0., 0., 0., 0., 0., 36.]]

0.150000	96.39%	96.20%	96.04%	96.06%	[[32., 0., 0., 0., 1., 0., 0., 0., 0., 0.], [0., 32., 0., 0., 0., 0., 0., 0., 0., 0.], [0., 0., 43., 0., 0., 0., 0., 0., 0., 0.], [0., 0., 0., 33., 0., 0., 0., 0., 0., 0.], [0., 0., 0., 0., 36., 0., 0., 0., 0., 1.], [0., 0., 0., 0., 0., 40., 1., 0., 0., 0.], [0., 0., 0., 0., 0., 1., 32., 0., 0., 0.], [0., 0., 0., 0., 0., 0., 0., 38., 0., 0.], [0., 1., 0., 0., 0., 0., 1., 0., 34., 0.], [0., 0., 0., 0., 0., 0., 0., 0., 1., 33.]]
1.5000	7.50%	0.75%	10.00%	1.40%	[[0., 0., 34., 0., 0., 0., 0., 0., 0., 0.], [0., 0., 31., 0., 0., 0., 0., 0., 0., 0.], [0., 0., 35., 0., 0., 0., 0., 0., 0., 0.], [0., 0., 31., 0., 0., 0., 0., 0., 0., 0.], [0., 0., 45., 0., 0., 0., 0., 0., 0., 0.], [0., 0., 38., 0., 0., 0., 0., 0., 0.,

					0.], [0., 0., 49., 0., 0., 0., 0., 0., 0., 0.], [0., 0., 37., 0., 0., 0., 0., 0., 0., 0.], [0., 0., 30., 0., 0., 0., 0., 0., 0., 0.], [0., 0., 30., 0., 0., 0., 0., 0., 0., 0.]]
15.0000	8.89%	0.89%	10.00%	1.63%	[[28., 0., 0., 0., 0., 0., 0., 0., 0., 0.], [38., 0., 0., 0., 0., 0., 0., 0., 0., 0.], [40., 0., 0., 0., 0., 0., 0., 0., 0., 0.], [31., 0., 0., 0., 0., 0., 0., 0., 0., 0.], [42., 0., 0., 0., 0., 0., 0., 0., 0., 0.], [39., 0., 0., 0., 0., 0., 0., 0., 0., 0.], [28., 0., 0., 0., 0., 0., 0., 0., 0., 0.], [35., 0., 0., 0., 0., 0., 0., 0., 0., 0.], [37., 0., 0., 0., 0., 0., 0., 0., 0., 0.], [42., 0., 0., 0., 0., 0., 0., 0., 0., 0.]]
Epochs	Accuracy	Precision	Recall	F-1	Confusion Matrix

1	78.06%	80.99%	77.64%	76.97%	[[26., 0., 0., 2., 3., 0., 0., 0., 5., 0.], [7., 8., 5., 4., 8., 5., 0., 0., 1., 1.], [0., 0., 26., 6., 1., 3., 0., 0., 5., 2.], [0., 5., 3., 22., 0., 1., 0., 0., 10., 7.], [0., 3., 0., 2., 10., 8., 8., 2., 2., 1.], [6., 3., 0., 1., 1., 9., 0., 0., 12., 1.], [0., 0., 2., 0., 8., 2., 18., 0., 0., 0.], [1., 2., 4., 2., 1., 0., 0., 18., 6., 0.], [1., 4., 2., 2., 2., 0., 3., 1., 11., 1.], [13., 1., 0., 7., 0., 0., 1., 1., 2., 9.]]
10	92.78%	92.88%	92.58%	92.65%	[[30., 0., 0., 0., 0., 1., 0., 0., 0., 2.], [0., 36., 0., 0., 0., 4., 0., 0., 4., 1.], [0., 0., 34., 2., 0., 0., 0., 1., 0., 0.], [0., 2., 1., 39., 0., 0., 1., 1., 0., 1.], [0., 1., 0., 0., 31., 0., 0., 0., 0., 0.], [1., 0., 1., 0., 0., 34., 0., 0., 1., 3.],

					[0., 0., 0., 0., 0., 0., 32., 0., 1., 0.], [0., 0., 0., 0., 0., 0., 0., 32., 0., 1.], [0., 0., 0., 1., 0., 0., 2., 1., 23., 0.], [1., 1., 0., 5., 1., 2., 0., 1., 3., 21.]]
100	96.67%	96.87%	96.97%	96.89%	[[35., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.], [0., 37., 0., 0., 0., 0., 0., 0., 0., 0., 0.], [0., 0., 33., 0., 0., 0., 0., 0., 1., 0.], [0., 0., 0., 43., 0., 0., 0., 0., 3., 0.], [0., 0., 0., 0., 38., 0., 0., 0., 0., 0.], [0., 0., 0., 0., 1., 34., 0., 0., 0., 1.], [0., 0., 0., 0., 0., 1., 35., 0., 0., 0.], [0., 0., 0., 0., 0., 0., 0., 34., 1., 0.], [0., 0., 0., 0., 0., 0., 0., 0., 31., 0.], [0., 1., 0., 1., 0., 2., 0., 0., 3., 25.]]

300	98.33%	98.34%	98.27%	98.29%	[[45., 0., 0., 0., 0., 0., 1., 0., 0., 0.], [0., 27., 0., 0., 0., 0., 0., 0., 0., 0.], [0., 0., 36., 0., 0., 0., 0., 0., 0., 0.], [0., 0., 0., 40., 0., 0., 0., 0., 3., 0.], [0., 0., 0., 0., 34., 0., 0., 0., 0., 4.], [0., 1., 0., 0., 0., 27., 0., 0., 0., 2.], [0., 0., 0., 0., 0., 0., 35., 0., 0., 0.], [0., 0., 0., 0., 0., 0., 0., 36., 0., 0.], [0., 2., 0., 0., 0., 0., 1., 0., 36., 0.], [0., 0., 0., 0., 0., 0., 0., 0., 0., 30.]]
500	96.67%	96.56%	96.82%	96.59%	[[40., 0., 0., 0., 0., 0., 0., 0., 0., 0.], [0., 45., 0., 0., 0., 0., 0., 0., 0., 0.], [0., 0., 36., 0., 0., 0., 0., 0., 0., 0.], [0., 0., 0., 32., 0., 2., 0., 0., 0., 0.], [0., 0., 0., 0., 28., 0., 0., 0., 3., 0.], [0., 0., 0., 1., 0., 29., 0., 0.,

					0., 2.], [0., 0., 0., 0., 0., 0., 39., 0., 0., 0.], [0., 0., 0., 0., 0., 0., 0., 33., 1., 0.], [0., 0., 0., 0., 0., 0., 0., 0., 33., 0.], [0., 1., 0., 0., 0., 0., 0., 0., 1., 34.]]