Department of Computer Science
COSC 4P80 - Artificial Neural Network

Motor Classification - Multilayer Feed Forward Neural Network

Prepared by: Hridoy Rahman
Date: December 6, 2024

Abstract

A neural network is a method in artificial intelligence that helps computers process data in a way such that it can learn and automatically generate inferences on unseen data. Neural networks inspired by observing the way the human brain works, except the human brain always works in forward direction, whereas neural networks works in forward and backward directions. Neural network uses interconnected nodes in a layered manner to adjust weights and predict outputs. This assignment is related to implementing a neural network from scratch to train a model on provided or combined datasets such that the network is able to perform binary classification. The dataset contains good or motors. The goal is to successfully classify good and bad motors given an unseen dataset to the model. In part(A), this segment discusses the architecture, values for alpha, transfer functions, and iterations used to build the neural network model. Following that, part(B) breaks the input dataset into 3 sets proportionately keeping good and bad motors, and merges first two sets for training dataset and keeps the third portion of the dataset for testing the model. This section also, creates a loss graph which depicts the error of the training set and test for each of the 3 combinations created. Furthermore, discusses the way generalization effects with varying number of nodes in the hidden layer. Part(C) is built on top of part(B) adding momentum to show if the model's accuracy increases or not. A comparison is also shown between part(B) and part(C). Following that, in part(D) a report of varying dataset combination is shown arguing the best results for a dataset. In part(E), the initial neural network model is extended to 5 layer model to see neural network's behavior. In the last part(F), Softmax function is implemented at the output layer and a comparison is shown between the part(E) and part(F). Finally, this report concludes with the learning of the experiments done and model's accuracy and shortcomings.

Contents

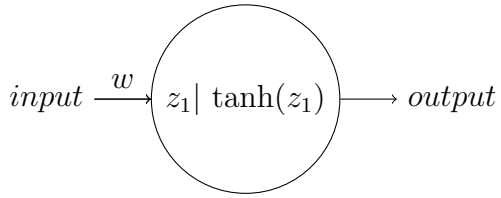
1 Introduction

1.1 Intro - Neural Network

A neural network is a machine learning model algorithm that is inspired from the way human brain function. A neural network is able to train on a set of data and is able to predict results on unseen data. A neural network consists of input layers, hidden layers, output layers. Each unit in these layers called a node or neuron or processing element (P.E), where each node is interconnected with adjacent layer's neurons and weights are associated with these connections. These weights adjust during training to help the model learn patterns in the given data. A model may consists of various methods or techniques such as forward propagation, backward propagation, transfer functions, loss functions, regularization, momentum, static or dynamic learning rates, weight initializations, etc.

1.2 Neuron

Diagram 1: Single Neuron

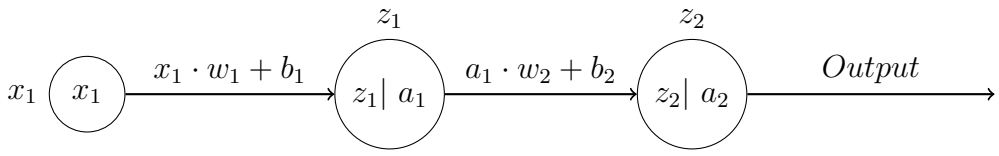


Each neuron contains an input, a transfer function, and an output associated with it. In the above example, given input = 2, $w = 1.5$, b = 1, $z1 = \{(input * w) + b\}$, $\tanh(z_1) = \frac{e^{z_1} - e^{-z_1}}{e^{z_1} + e^{-z_1}}$, after calculating z1 = 2 * 1.5 + 1 = 4, we get $\tanh(4) = \frac{e^4 - e^{-4}}{e^4 + e^{-4}} \approx 0.9993$ is neuron's output with tanh as the transfer function. Any output of a neuron is the composition of the weighted inputs run through an activation function. It is not necessary to have activation function in all layers, it's possible to have one type of activation function for the hidden layers and another type for the output layer. Discussed later in this section.

1.2.1 Forward Propagation

Forward propagation is simply feeding input data in a forward direction calculating through the network to the output layer. During this forward pass, the network calculates intermediate values at each layer, applying weights biases and activation functions and forwards the values to next layer and continuing this process until reached to the final layer meaning the output layer.

Diagram 2: Forward Propagation in Neural Network



In the following diagram 2, shows the forward propagation works. First neuron contains

the x_1 input, following that, the next layer's input $z1$ is denoted by the first layer's input times the weights plus the bias $x_1 \cdot w_1 + b1$. Note that, a_1 in the z_1 layer is the activation of the $z1$, where $a_1 = tanH(z_1)$. Same for the second layer marked with z_2. It takes the previous node's activation as input a_1 times the $w2$ plus the bias $b2$. And it passes through the activation function once again, and this is the output of the network.

1.2.2 Back Propagation

Back propagation helps the network adjust the weights with respect to loss making the network more effective at predicting results. After applying the feed forward pass, the network computes an error by computing its output to the actual target, if this error is large, the network's prediction needs improvement. To make this work, back propagation comes into play by minimizing the error at the output node by propagating the error backwards through the network to the weight. The back propagation algorithm utilizes the mathematical chain rule from calculus to compute through the complex layers of neural network.

1.2.3 Transfer Functions

A transfer function is also referred to as an activation function in neural networks, is a mathematical function applied to the output of each within a layer before passing this value to the next layer. There are various types of transfer functions, sigmoid funciton, hyperbolic tangent function, Rectified Linear Unit functions, etc. Sigmoid activation function squishes the values within 0 to 1 range and is often used in binary classification. Hyperbolic tangent function is similar to sigmoid function but with a broader range of functions, which ranges from -1 to 1 and is useful for faster convergence and it has stronger gradiennts than the sigmoid function. ReLu ranges from 0 to infinity and used most of the time with CNN (Convulational Neural Network) architectures. Softmax function is usually applied at the output layer and ranges from 0 to 1 and usually used to for classification purpose.

1.2.4 Loss Functions

A loss function also known as cost function, measures the inaccuracy of a model's predictions by calculating the difference between the predicted and actual values. There are various loss function used in various situations. MSE (Mean Squared Error), Binary Cross Entropy, Cross Entropy, and many more.

1.2.5 Regularization & Momentum

Regularization refers to the technique to prevent overfitting by adding constraints to the model. Whereas, momentum helps with optimization of neural network by improving the efficiency of gradient descent.

1.2.6 Datasets

The datasets used in this assignment are the provided ones, and by creating combining various files. The merging algorithm takes n amount of bins, merges the features together, and outputs in a new file. The dataset contains good and bad motors, where the features

vary in size in different files. For each provided dataset, the good motors have a count of 34 and 19 for bad motors resulting a total of 53 samples.

1.3 Part(A)

Attribute	Value
Neural Network Architecture	48 X 25 X 2
Input Features	48
Hidden Layer Amount	1
Hidden Layer Size	25
Output Size	2
Transfer Function	tanh()
Epochs	1500
Learning Rate	0.001
Loss Function	Cross Entropy Loss

Table 1: Neural Network Configuration

The network architecture used in this section is a three-layer model with an input layer, one hidden layer and one output layer. The input layer derived from the count of features in a file. If a bin is given to the program, the program extracts the samples amount and the feature size or bin size of that file and uses in the file. A configuration file is given to explicitly mention how much neurons are there in a input layer. The configuration file also contains the number of hidden layer and for each layer a number should be provided. Therefore, the program extracts these information from the configuration file to create a neural network model. Furthermore, the hidden layer architecture hidden layer size was chosen using the size of the input layer plus the output layer divided by two $\{(inputLayerSize + outputLayerSize)/2\}$. The activation function or transfer function used in this model is the hyperbolic tangent function. The tanh function of x is defined as given below:

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

The learning rate or alpha used in this model is 0.001. This function helps the model converge. A smaller learning rate is used to adjust the weights such that the model approaches the minimum loss at a slower speed. If learning rate given is too big for example, 0.01 or 0.1 the model overshoots and while accelerates learning but output prediction is rather incorrect. The configuration file can control the epochs from outside the program. With higher number of epochs the the model's learning increase as it gets more chance to adjust the weights based on feedback from the loss function. However, too many epochs may lead to memorization, after trial and error a balanced number of epochs were chosen for this model. The loss function used in this network is cross entropy loss function to help converge the neural network by calculating the difference between the predicted and actual ground truth labels. By minimizing this loss using cross entropy loss function, the network adjusts its weights to improve accuracy of the model itself. The cross-entropy loss function with L2 regularization is defined as follows:

$$\text{Cross Entropy Loss} = \frac{1}{N}\sum_{i=1}^{N} -\log(a_{2,i}[y_i]) + \frac{\eta}{2}\left(\sum_j \text{weight}_{1,j}^2 + \sum_k \text{weight}_{2,k}^2\right)$$

where:

- N: Number of samples in the dataset (x.shape[0]),
- $a_{2,i}[y_i]$: The predicted probability for the correct class y_i of ith sample,
- η: Regularization parameter for penalizing large weights,
- $\text{weight}_{1,j}$ and $\text{weight}_{2,k}$: sum of each weight matrices added

$$\text{DataLoss} = -\sum_{i=1}^{N} \log(a_{2,i}[y_i])$$

$a_2[y_i]$ represents the predicted probability for each correct class y_i of sample i. The negative log of this probability for each sample provides the correct log probability for the correct class indexed by y for each data in x. The following Data Loss part of the cross entropy loss equation helps penalize the model with larger weights. The regularization term in the following equation is $\frac{\eta}{2}$, $\left(\sum_j \text{weight}_{1,j}^2 + \sum_k \text{weight}_{2,k}^2\right)$ this part computes the square of each element belongs in weight1 and weight2 and ensures the values are positive and penalizes the model for incorrect predictions with larger weights. Utilizing this method helps us train the model in a efficient way. By implementing this function, the model generalizes well by controlling the weight magnitudes which helps reducing the risk of overfitting.

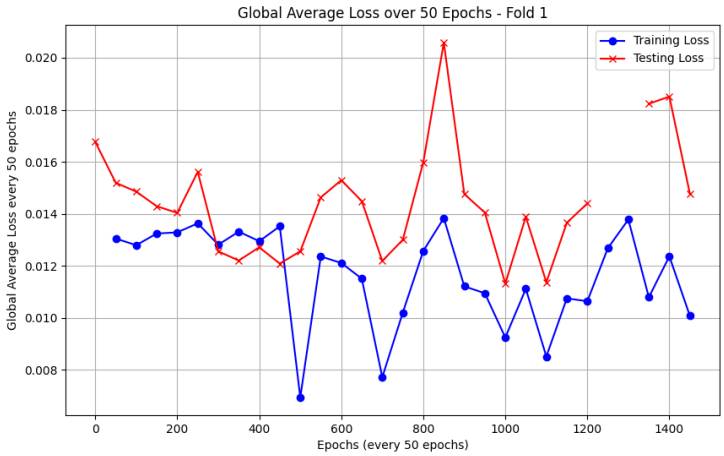
$$\text{DataLoss} = DataLoss + \frac{\eta}{2}\left(\sum_j \text{weight}_{1,j}^2 + \sum_k \text{weight}_{2,k}^2\right)$$

1.4 Part(B)

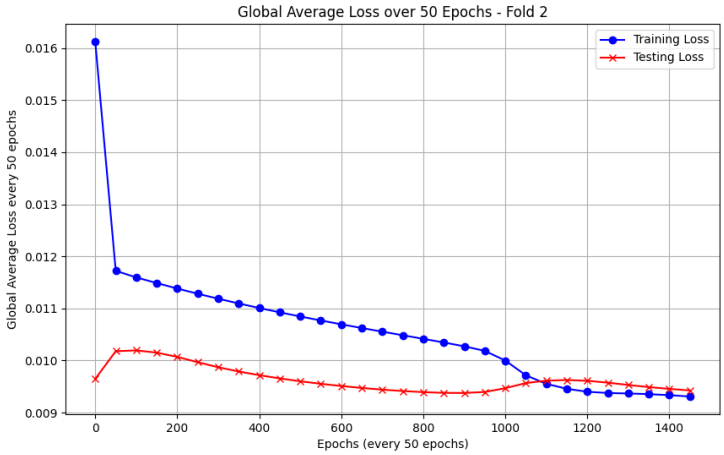
1.4.1 K-fold algorithm explanation

The algorithm finds the good and bad motors and seperately stores them in two different arrays. Then the arrays shuffled within themselves such that there is a good randomness but not true randomness since a seed was used hence it is possible to recreate the result. Following that, the good motors and bad motors array size was divided into 3 resulting in 11 good motors on each set. And 19 bad motors from the dataset was divided by 3 as well to get three more sets on the bad motors. Following that, three different combinations of dataset is created and stored in a dataset. And the dataset was passed through the back propagation algorithm tracking the training and testing dataset's loss and the loss was averaged 50 epochs across all epochs and finally providing the graphs of three fold's performance on the neural network.

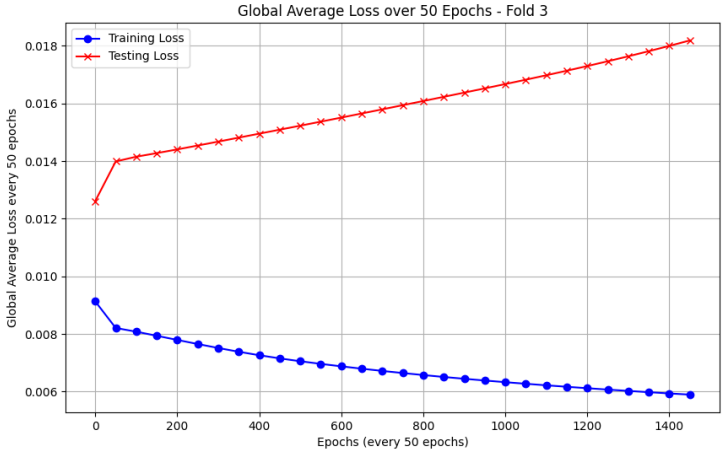
1.4.2 Graph Comparison



(a) Graph 1



(b) Graph 2



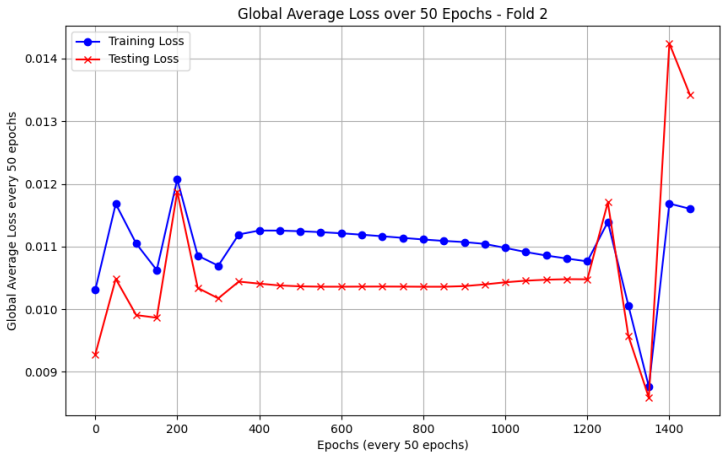
(c) Graph 3

Figure 1.1: Loss Graphs

The following graphs: graph one contains the result of the first fold from the datasets, graph two contains the second fold and graph three contains third fold. For each graph, the loss of training and testing dataset is averaged across 50 epochs till 1500 epochs for this experiment. It is possible to the change the total epochs size. From the graph2, training loss is the blue line, testing loss is the red line. Training loss shows a steady and consistent decrease as the number of epochs increases, suggesting the model is learning the data effectively. On the otherhand, the testing loss starts a little below the blue line, fluctuates till 1000 epochs then it settles to a stable loss. The graph 1, fluctuates a lot suggesting the learning is not constant. Lastly, the third graph showing that the training set consistently learns however, the testing set increases over time suggesting incorrect predictions.

1.4.3 Varying Size of Nodes in Hidden Layer

First of all, the dataset used in this section is by merging the data file of 16 bin size and 32 bin size together creating a 48 bin size dataset. Next, the hidden layer's size is denoted by $hiddenLayerNodeSize = (inputLayerSize + outputLayerSize)/2$. If the neural network is less than the hiddenLayerNodeSize variable than the network can't fit the data properly such that the network starts to predict inaccurately. On the other hand, if the hidden layer's node size is larger than the hiddenLayerNodeSize than the network starts to memorize and results in higher accuracy closer to 1 or even 1. However, on unseen data the network fails to predict accurately. So choosing the hidden layer's node size around the mean value of input and output, helps the network balance between overfitting and underfitting the data. This also helps the network generalize better on new data. From the figure 1.2: the result is gotten from using lower hidden layer node size than the original. For this graph the hidden layer size is 15, the original uses 25. Also, note that, comparing this graph with fold2 from previous section, this fluctuates more and shows that sometimes it can generalize well where sometimes can not. Lastly, this graph, the blue and the red line closely follows each other depicting the training and testing generalization is close to each other.



(a) Graph 1

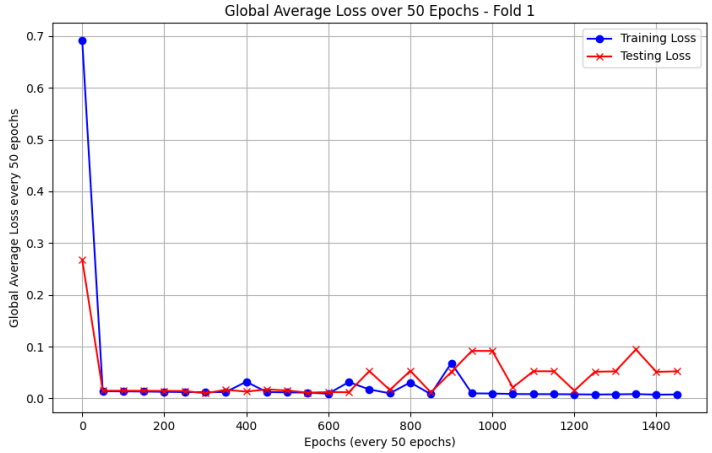
Figure 1.2: Graphs

1.5 Part(C)

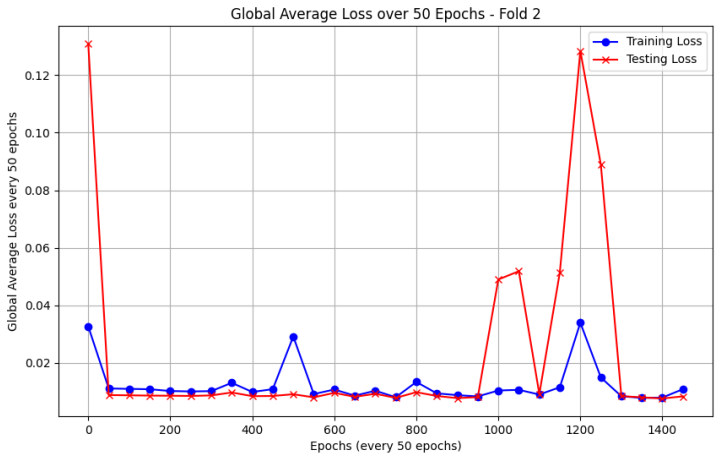
For this section the momentum mathematical formula was implemented while updating the weights and biases. The momentum algorithm for gradient descent updates is described as follows:

$$v_t = \beta \cdot v_{t-1} + (1 - \beta) \cdot \nabla \mathcal{L}(W_{t-1})$$

The β parameter controls the amount of past calculations influence the current update. A higher β value makes convergence smoother and helps the model predict correctly. $\beta \cdot v_{t-1} + (1 - \beta)$ This portion of the formula controls the previous and current gradient calculations and this part is multiplied with the loss value with respect to the weight $\nabla \mathcal{L}(W_{t-1})$.



(a) Graph 1

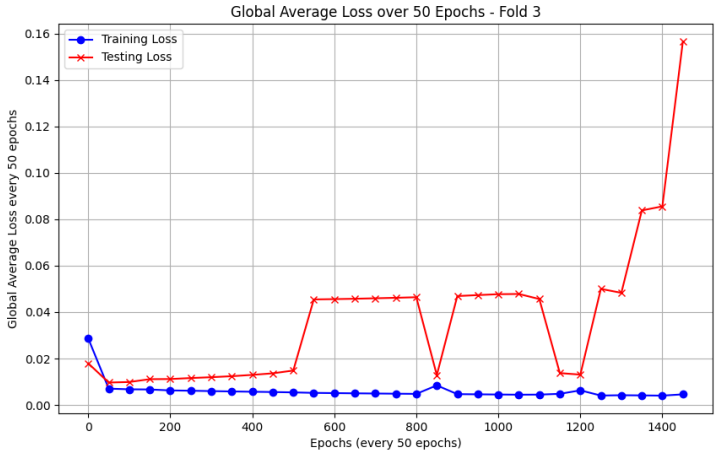


(b) Graph 2

Figure 1.3: Graphs

After adding the momentum the network provided much better results than the previous implementation without momentum from Part(B).

The comparison table 2 shows the changes in the neural network between part(B) and



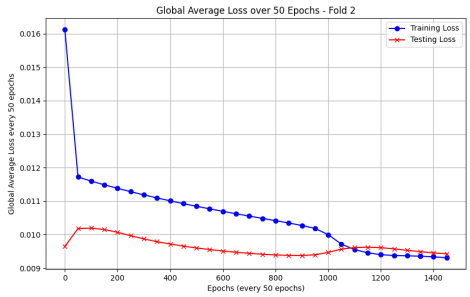
(a) Graph 3

Figure 1.4: Graphs

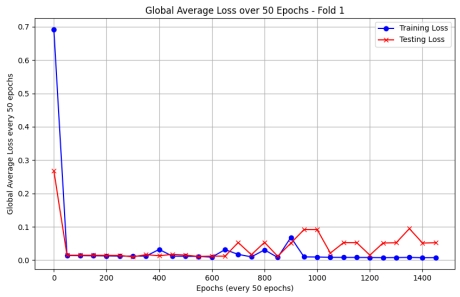
Attribute	Part(B)	Part(C)
Architecture	48 X 25 X 2	48 X 25 X 2
Learning Rate	0.001	0.001
Beta	n/a	0.5
eta	0.1	0.1

Table 2: Comparison of Attributes in Part(B) and Part(C)

part(C). In part(B) it was seen that second graph performed well. However, in the graph 2 from part(B), the network converged slowly. On the other hand, in figure 1.3, graph 1, it can be seen that the network converged pretty fast compared to the part(B)'s performance graph 2. For better visualization,



(a) Graph 1



(b) Graph 2

Figure 1.5: Graphs

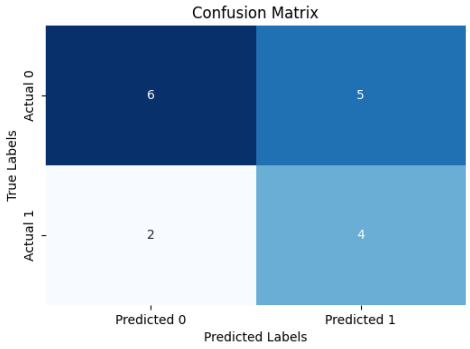
There are some fluctuations in the graph 2 and some fluctuations in graph 1, from figure 1.5. But it is clearly visible after adding momentum, the network dived to a lower error rate and converged rather faster than the earlier algorithm in part(B). The graph 2 from figure 1.3, it depicts there are some fluctuations at the end, however it does provide better accuracy in terms of training and testing algorithm rather than the graph 1 from figure

1.3. Figure 1.3, graph 3 the training performance is really good however this denotes that the model is memorizing the data. So all in all, Adding momentum helped the network converged faster with better generalization capability.

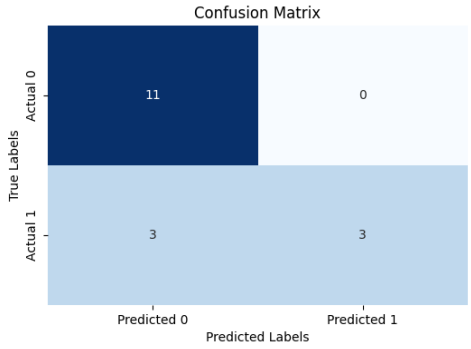
1.6 Part(D)

The combinations of files used in this section are 48 bin size, 16 bin size, and 64 bin size. For each file, a confusion matrix was generated to show the correctness of the test dataset.

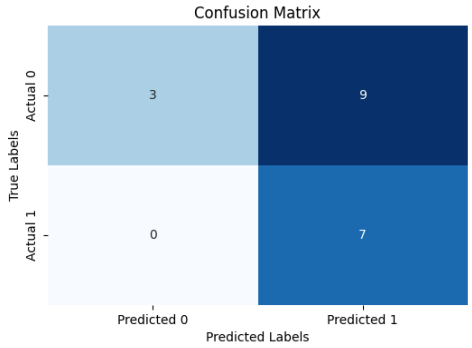
1.6.1 48 Bin Size



(a) Fold 1: test dataset



(b) Fold 2: test dataset



(c) Fold 3: test dataset

Figure 1.6: Confusion Matrix of 48 bin size

From the above matrix plots, we can see the fold 2 test dataset from figure 1.6 (b) performed better. From the equations below it is visible that the performance accuracy for fold 2 dataset is higher than the others.

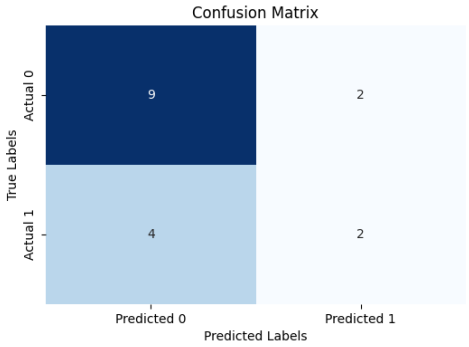
$$\text{Fold 1: Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} = \frac{4+6}{4+6+5+2} = \approx 0.0.59$$

$$\text{Fold 2: Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} = \frac{3+11}{3+11+0+3} = \approx 0.824$$

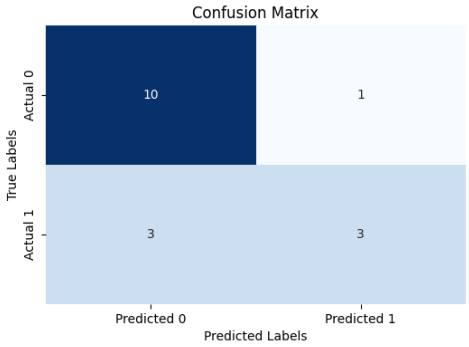
$$\text{Fold 3: Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} = \frac{7+3}{7+3+9+0} = \approx 0.526$$

1.6.2 16 Bin Size

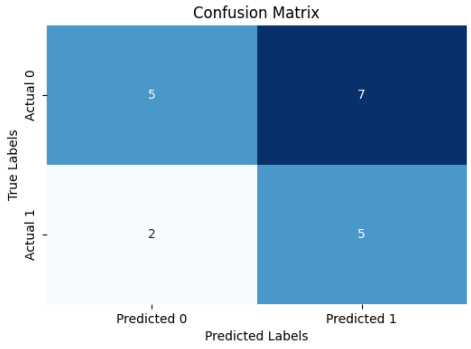
For the 16-bin Size,



(a) Fold 1: test dataset



(b) Fold 2: test dataset



(c) Fold 3: test dataset

Figure 1.7: Confusion Matrix of 16 bin size

$$\text{Fold 1: Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} = \frac{2+9}{2+9+2+4} = \approx 0.65$$

$$\text{Fold 2: Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} = \frac{3+10}{3+10+1+3} = \approx 0.77$$

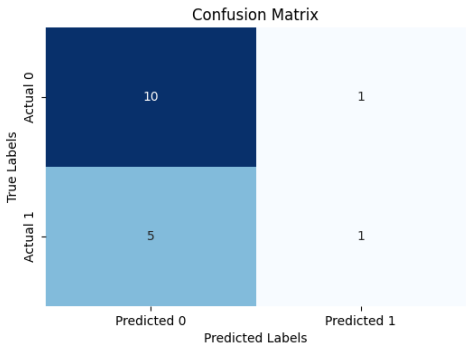
$$\text{Fold 3: Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} = \frac{5+5}{5+5+7+2} = \approx 0.53$$

For the 16 bin size the accuracy acquired is 0.77. So far, bin size 48 has the highest accuracy. Let's try the 64 bin size.

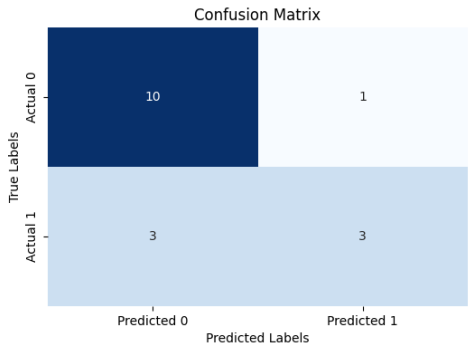
1.6.3 64 Bin Size

$$\text{Fold 1: Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} = \frac{1+10}{1+10+1+5} = \approx 0.65$$

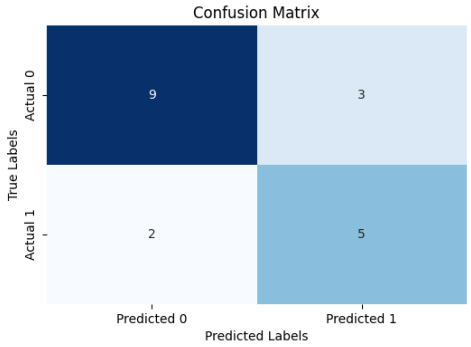
$$\text{Fold 2: Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} = \frac{3+10}{3+10+1+3} = \approx 0.77$$



(a) Fold 1: test dataset



(b) Fold 2: test dataset



(c) Fold 3: test dataset

Figure 1.8: Confusion Matrix of 64 bin size

$$\text{Fold 3: Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} = \frac{5+9}{5+9+3+2} = \approx 0.74$$

From each of 48 bin size, 16 bin size, and 64 bin size the highest accuracies are from fold 2 dataset.

1.7 Part(E)

The Part(D) is extended with layer five neural network. The following architecture of the extended model is given below in the table.

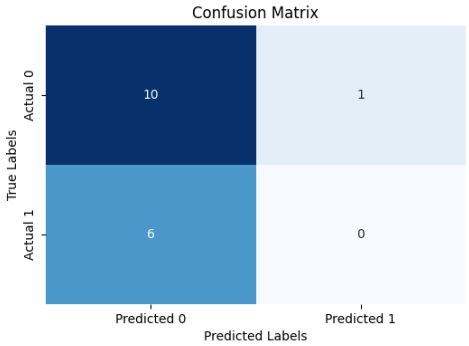
The following architecture consists of 5 layer neural network, where one input layer, 3 hidden layers and one output layer. The 48 bin size dataset is being used for this part since it's the best from the part(D). This architecture uses tanh activation function at the output layer. Other than that this neural network is extended to 5 layers.

1.7.1 48 Bin Size Compared with Part(D)

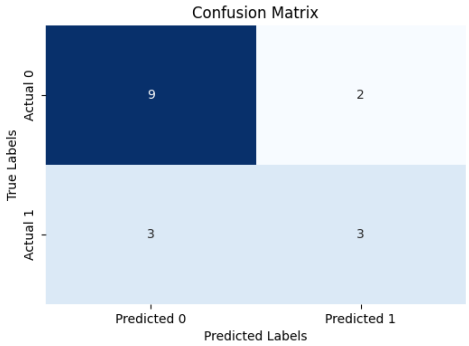
$$\text{Fold 1: Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} = \frac{0+10}{0+10+1+6} = \approx 0.59$$

Attribute	Value
Neural Network Architecture	48 X 25 X 12 X 6 X 2
Input Features	48
Hidden Layer Amount	3
Hidden Layer 1	25
Hidden Layer 2	12
Hidden Layer 3	6
Output Size	2
Transfer Function	tanh()
Epochs	2500
Learning Rate	0.001
eta	0.1
Beta	0.5
Loss Function	Cross Entropy Loss

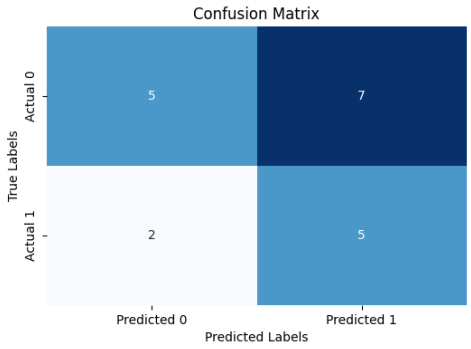
Table 3: Neural Network Configuration



(a) Fold 1: test dataset



(b) Fold 2: test dataset



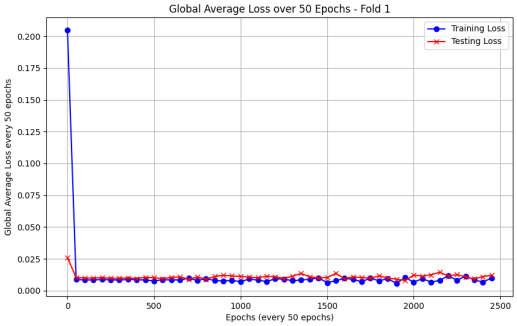
(c) Fold 3: test dataset

Figure 1.9: Confusion Matrix of 48 bin size

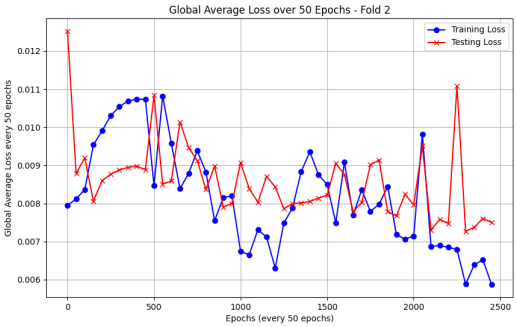
$$\text{Fold 2: Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} = \frac{3+9}{3+9+2+3} = \approx 0.71$$

$$\text{Fold 3: Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} = \frac{5+5}{5+5+7+2} = \approx 0.53$$

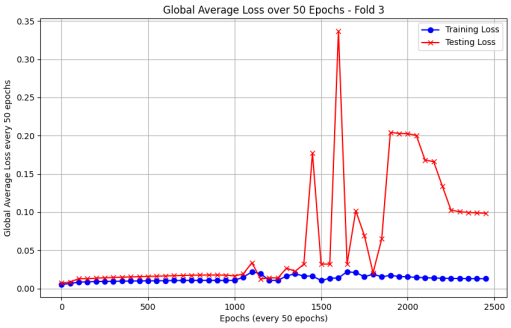
From this confusion matrix from figure 1.9 it is visible that extending to layer 5 didn't help improved where in part(D) figure 1.6 Fold 2, performs much better than the newly build model. Therefore, confusion matrix in figure 1.9 performs better.



(a) Fold 1: test dataset



(b) Fold 2: test dataset



(c) Fold 3: test dataset

Figure 1.10: Graph of 5 Layered Neural Network

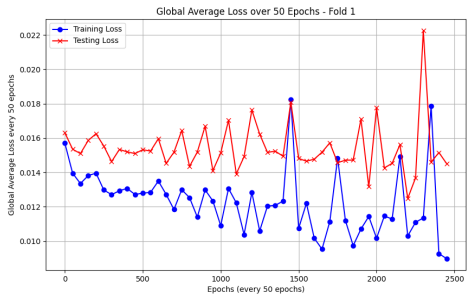
The follow graphs are created from extending the neural network. The graph in fold 2 from figure 1.10, shows a lot of fluctuations and it has the highest test accuracy among other Fold datasets. On the other hand, Fold 1's training accuracy and testing accuracy are better but it doesn't perform well according to the confusion matrix from figure 1.9(a). Furthermore, The third graph from figure 1.10(c) behaves the same graph 1 from 1.10(a).

In general we can say without softmax output, adding momentum and utilizing extended hidden layer didn't help in this case.

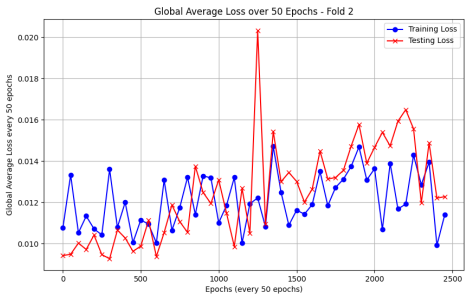
1.8 Part(F)

Attribute	Value
Neural Network Architecture	48 X 25 X 12 X 6 X 2
Input Features	48
Hidden Layer Amount	3
Hidden Layer 1	25
Hidden Layer 2	12
Hidden Layer 3	6
Output Size	2
Transfer Function	tanh()
Output Function	$SoftmaxFunction$
Epochs	1500
Learning Rate	0.001
eta	0.1
Beta	0.5
Loss Function	Cross Entropy Loss

Table 4: Neural Network Configuration



(a) Fold 1: test dataset



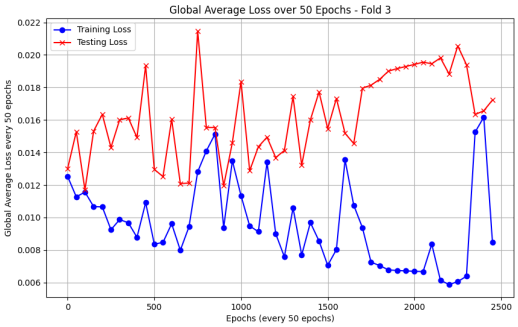
(b) Fold 2: test dataset

Figure 1.11: Graph of 5 Layered Neural Network

$$\text{Fold 1: Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} = \frac{3+9}{3+9+2+3} = \approx 0.71$$

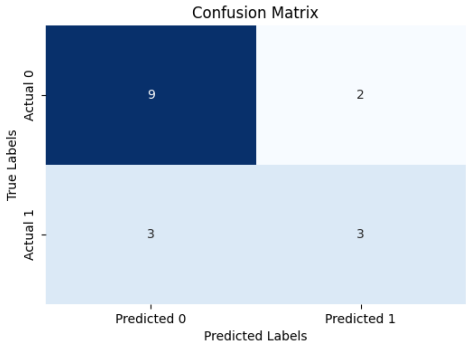
$$\text{Fold 2: Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} = \frac{1+10}{1+10+1+5} = \approx 0.65$$

$$\text{Fold 3: Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} = \frac{0+12}{0+12+0+7} = \approx 0.63$$

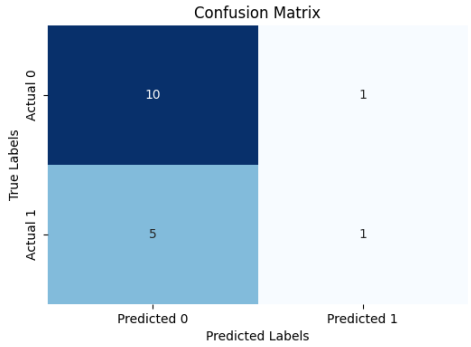


(a) Fold 1: test dataset

Figure 1.12: Graph of 5 Layered Neural Network

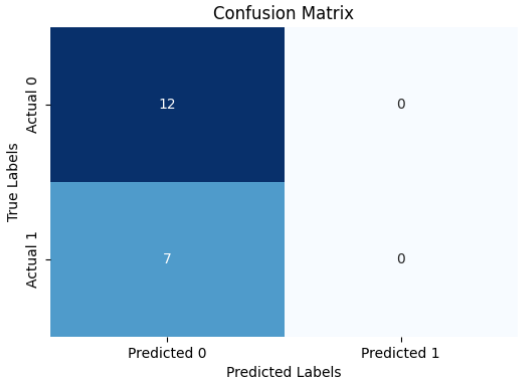


(a) Fold 1: test dataset



(b) Fold 2: test dataset

Figure 1.13: Graph of 5 Layered Neural Network



(a) Fold 1: test dataset

Figure 1.14: Graph of 5 Layered Neural Network

From the above calculations and from part(E), it is visible that the second fold set performs well. And the softmax function actually improved the test output. From part(E) the Fold 1 and Fold 3 has lower accuracy than Fold 2 and 3 from Part(F). Implicating that softmax function has actually improved test accuracy.

1.9 Conclusion

Overall, This report implements a 3 and 5 layered neural network. Where everything is shown experimentally. The implementation successfully classifies the provided dataset. And the model is actually able to classify properly. In general the report tested three different bin size of dataset to show how well the model predicts. In the report, it is shown that the network generalizes well and predicts well on test dataset without extending the neural network. For a small dataset, extending the neural network doesn't necessarily means the model will perform well. Since overfitting may happen due to an increase in memory size of the neural network meaning extending the hidden layer size. Finally, extending the hidden layers doesn't guarantee better performance. It is always essential to balance the model based on complexity of the data.