Department of Computer Science
COSC 4P80 - Artificial Neural Networks

Examination of the Feed Forward Neural Network

Prepared by: Kevin Olenic
Student #: 6814974
Instructor: Dave Bockus
Date: March 10, 2023

Abstract

A Feed Forward Neural Network (FFNN) is the most basic artificial intelligence algorithm currently capable of information processing. Its purpose: mimic the human brain's information analysis abilities. This report analyzes different parameters and their effects on the neural network, giving insight into what parameters affect the FFNN's ability to analyze and classify data. The analysis in this report will demonstrate the effect parameters, such as the number of nodes in the hidden layers, and the combination of data sets used for training and testing will have on the network's ability to classify. The report will also examine the effect momentum has on learning speed. While also looking at how the additional hidden layers affect the classification ability of the neural network. The data used to analyze the FFNN in this report pertains to electric motors. In particular, the amount of current the engine draws from its power source, with each data set separates into one of two categories; good and bad motors. These tests will aid in determining which parameters help in the classification, thus helping design future models examining different data sets.

Contents

1 Introduction

In this section, the goal of this paper and relevant topics such as Feed Forward Neural Networks (FFNN) and back-propagation is explained to improve the reader's understanding of the concepts covered in this report.

1.1 Goal of this Report

The goal of this report: examine the Feed Forward Neural Network (FFNN) artificial intelligence. This is accomplished by developing, training and testing the algorithm with information on electric motors. Using this data, we examine the FFNN's ability to classify linearly separable data (i.e. data that falls into one of two categories).

1.2 Feed Forward Neural Network

A Feed Forward Neural Network (FFNN) is an artificially created neural network inspired and based on the human brain. This algorithm's purpose is to mimic the classification and decision-making abilities of the human brain.

The network contains multiple nodes, each defined with its inputs and outputs along with an activation and transfer function, mimicking the dendrites and axons of a neuron that act to receive and send information throughout the neurons in the brain.

The activation function portion of the node determines whether a node should activate by calculating a weighted sum, additionally adding bias to the value. Upon activation of the node, the transfer function in the node translates the received input signals into output signals.

The nodes transfer function is activated once it has received sufficient input to stimulate the node, similar to how a neuron in the brain will activate upon receiving enough stimuli. Nodes in the FFNN fall into one of three layers in the network, the first is the input layer, the second is the hidden layer, and lastly, the output layer.

The first layer of the network is the input layer, which receives the initial information from the environment and sends the input information into the neural network for further analysis. Depending on the network model employed, input is fed either into the hidden layer or in absence of a hidden layer, outputs of the input layer go directly into the output layer.

The hidden layer is the optional, second layer of the network, a network can either have single or multiple hidden layers. This layer receives input from the input layer or a previously hidden layer, these subsequent layers further the examination of the data to narrow down the corresponding classification of the information given.

The output layer is the final layer of neurons in the FFNN, this layer produces the final outputs of the algorithm by providing the predicted classification of the analyzed data that corresponds to the inputs given to the network.

The value entering the activation function is calculated by taking the sum of the input (x_i) and multiplying by the weights (w_i), this is represented in the equation below:

$$y = f(\Sigma_i^n x_i w_i) \tag{1}$$

The FFNN model is trained by taking in an input vector, representative of the data we want the model to classify, and performs a forward-pass on the data to predict to get the current classification the model determines that data corresponds. The forward pass is the initial calculation process of an FFNN, at this stage of training, the expected values of the output layers are provided from the input/training data, and all the nodes from the input layer to the output layer are traversed.

Below are the equations used in the forward-pass, where X^L is the pre-activation value fed into a neuron, W^L are the weights connected to the neuron, A^L is the post-activation value the neuron outputs and σ is the activation function used by each neuron.

$$X^L = W^L * A^{L-1}$$

$$A^L = \sigma(X^L)$$

$$\sigma = \frac{1}{1 + e^{-x}}$$

After performing the forward pass on the input vector used for training, the FFNN performs a gradient descent search, an optimization algorithm commonly used to train machine learning models and neural networks to reduce the output error of the FFNN. Training data helps the model learn over time, and the cost function within gradient descent gauges its accuracy with each iteration of parameter updates.

1.3 Back-propagation

This is a supervised learning technique that produces and feeds the training data into the network, if the network computes the output vector that matches the desired target output after a forward pass, nothing is re-adjusted as there is no error and the model is unaffected. If the network outputs an erroneous result at the end of the forward pass, the weights of the network are adjusted to reduce this error in the future, the main key factor of this learning rule is assessing the blame for the incorrect assessment and dividing it among the weights that contributed to that error.

After the forward pass is complete the algorithm then performs a backward pass to adjust the weights if the network made an incorrect output. To adjust the weights between the hidden and the output layer, the gradient descent rule is be applied.

Back-propagation works by randomly setting the weights (w_i) in the network a value in the range of [-1,1]. The training portion of the algorithm involves calculating the network output Error (T(target) - O(network output)) and adjusting the weights based on the error result. This process is repeated for several epochs or until convergence is reached.

The weight adjustment formula for the backward pass is given below, where W^L is the weight being adjusted, α is the learning rate and $\triangledown^L$ is the calculated gradient.

$$W^L = W^L - \alpha * \triangledown^L \tag{2}$$

$$Back - PropagationTrainingRule = \begin{cases} \Delta w_i = n(t - O)x_i \\ w_i \leftarrow w_i + \Delta w_i \end{cases} \tag{3}$$

1.4 Data

The data used in the examination of the FFNN in this report will consists of the amount of current an electric motor draws from its power source. Under normal circumstances, the current drawn by the motor will have a feedback component which causes the motor to have a smooth acceleration when starting. In the case of a bad motor, this feedback will be dirty (noisy). Separating out clean and noisy acceleration data can be used to determine if the motor is good or bad. Good motors are classified with a value of one zero (0) while bad motors are classified with a one (1).

2 Experimental Research

In this section, multiple experiments are performed, examining how generalization, momentum, training and testing data and finally how additional hidden layers in he network effect the feed forward neural network.

2.1 Assign Part A (FFNN Architecture)

This section of the report reviews the architectures and learning rate values attempted in the design of the feed-forward neural network, as well as the final architecture implemented for this report. The factors that influenced the poor performance of previous models are investigated and compared to the last architecture used for this report.

Previous implementations of the FFNN system architectures possessed a small number of neurons in each hidden layer and a static learning rate. These architectures resulted in quick training times but failed to correctly associate the test data, as they possessed significant error rates.

The reason is most likely due to the small number of neurons in the hidden layers and the static learning rate. The small number of neurons leads to under-fitting and high statistical bias. While the fixed learning rate caused limited exploration of the gradient, as high fixed learning rates limited the investigation of narrow areas, low learning rates did not allow great exploration of the slope.

Other architectures of the FFNN implemented hidden layers with multiple neurons, causing an increase in training time but allowing the model to categorize the training data correctly. However, when given the test data, the FFNN model failed to predict the correct expected output.

The reason is most likely due to the high number of neurons, causing the model to overfit and memorize the training data, resulting in the model making incorrect predictions about the data not used in the training of the network.

The implementation used for this report on the Feed Forward Neural Network possesses several neurons in each hidden layer proportional to the previous layer and a dynamic learning rate. The resulting architecture now has a decent amount of training time, and when given the test input, the model correctly predicted most of the output.

For this architecture, the input data was normalized by taking the sum of all the inputs and dividing each to obtain its percentage contribution to the totality, thus allowing the network to extract more meaningful data from the input vector, causing an increase in the network models learning ability.

The final model produced better results than previous models is likely due to the number of neurons in the hidden layers, now being proportional to prior layers, allowing better fitting. The dynamic learning rate allows for better traversal of the gradient, while the normalization of the data allows for better extraction of features from the data for better classification.

As the training begins with a considerable learning rate and periodically reduces the value, the FFNN can traverse quickly over the gradient. As training progresses and the error value reduces, causing a diminishing learning rate as time passes, permitting the examination of small and narrower regions of the slope.

The learning rate used in the model is dynamic as it will increase to eighty percent (80%) for values with significant errors and decrease to twenty percent (20%) for errors in the range of twenty to five percent, all errors below five percent (5%) will have a point one percent learning rate (.1%). Allowing the FFNN to explore the gradient for an area better suited for classifying the data while also allowing it to close in on sections of the error slope that permits the better classification of the more accurately predicted inputs.

Implementing this learning rate technique allows the FFNN to quickly jump from parameters not suited to classifying the provided data to a range of possibly ideal parameter values while also allowing the exploration of narrower areas of the loss function that provide optimal classification abilities.

The neurons used in the hidden layer of the final architecture are proportional to the number of neurons in the previous layer.

Based on the above output from the feed-forward neural network A.I, it is clear that making the number of neurons in the hidden layers proportional to the previous layer of neurons and making the learning rate dynamic created an FFNN architecture that can accurately predict the majority of the output for the test data based on the data used to train the model.

2.2 Assign Part B (FFNN Affects of Generalization)

The second experiment involves breaking the input into three sets where the contribution of good and bad motors is proportional to the number of good and bad engines. These sets are inputted into the network such that two are utilized for training the network, while the third will test the network.

The data set used during this experiment will consist of a data set containing thirty-two bins. The data set will be separated such that two groupings of the data will have eleven good and six bad motors, while the third set will possess twelve good and seven bad engines. Additionally, the ordering of these data sets will be swapped to produce three different combinations of training and testing data, to test the effect different combinations of training data will have on a testing set, resulting in a total of three assortments of input data to use in the analysis of the FFNN.

For the first examination of the neural network, the first hidden layer will possess twelve (12) neurons, and the second layer will contain seventeen (17). The graph below depicts the global error of the neural network for the training and testing set used for each epoch:

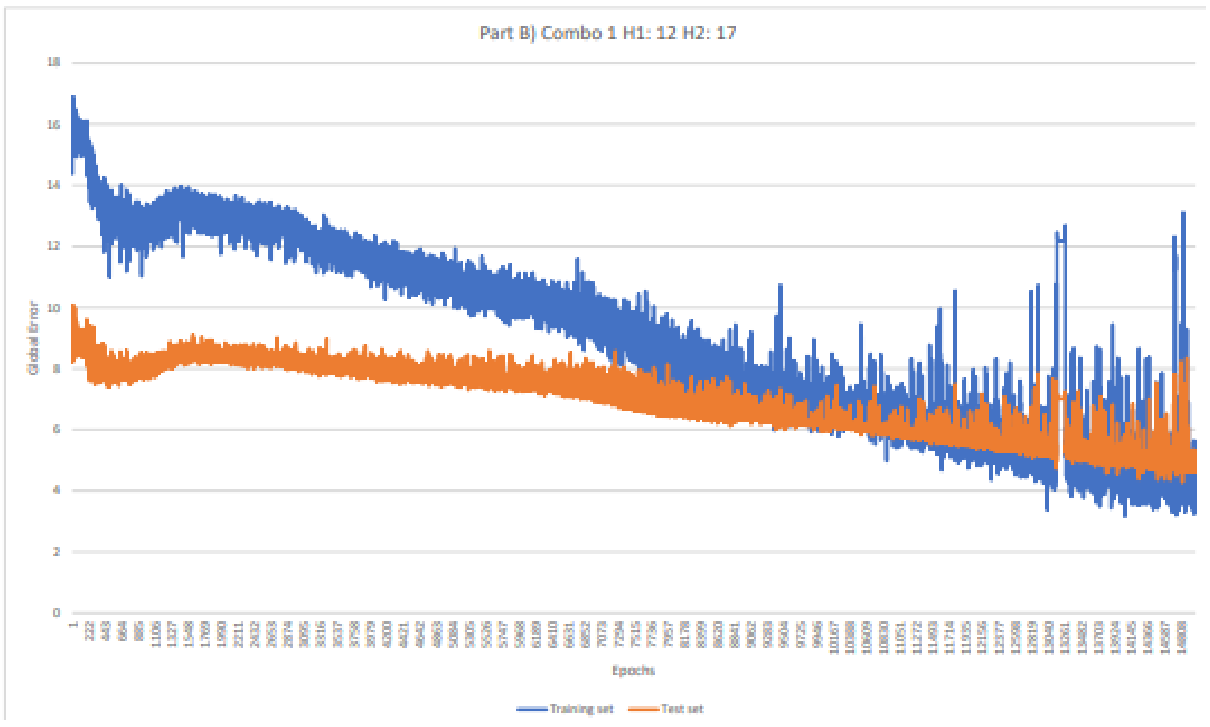


Figure 2.1: Global error per epoch of combinations 1

From the first graph, we can surmise that during the first few initial epochs, the FFNN is not very accurate when classifying the data. However, as time passes, the network begins learning from the data, causing the global error of the training and test set to decrease as it learns certain features belonging to specific categories. Looking further ahead, near the end of the graph, we see the error curves are beginning to diverge. This results from the FFNN begging to memorize the training set rather than learn generic patterns to help classify other data sets containing information on the same subject same type.

For the second examination, the first hidden layer will have twenty (20) neurons in the first layer and twenty-five (25) in the second layer for this combination of the data sets. The graph below depicts the global error of the neural network for the training and testing set used for each epoch:

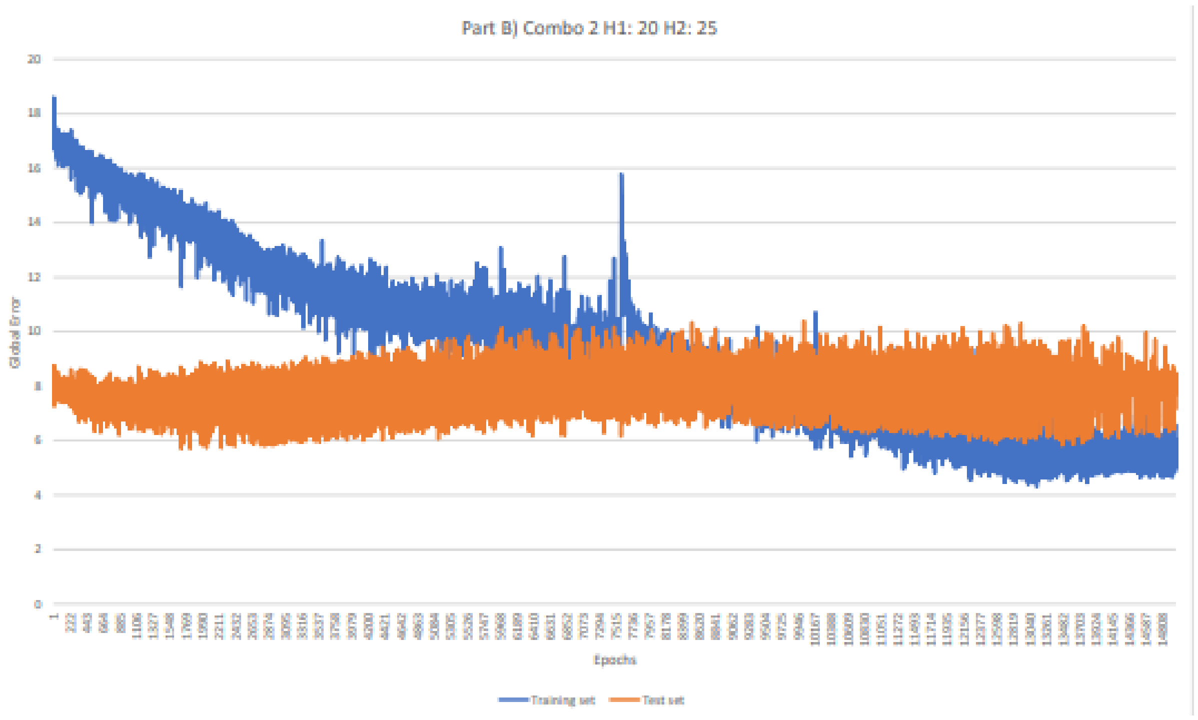


Figure 2.2: Global error per epoch of combination 2

For the second graph, we have a similar pattern as the first graph depicting the global error of the first combination over each epoch. Initially, the FFNN is subpar at classifying both the training and testing data correctly, but as each epoch passes the network becomes increasingly accurate at classifying the data until reaching a certain period, then the network begins to memorize the training data. Causing the accuracy of the test set classification to converge, ceasing all improvement of its classification ability for the test data.

For the final examination, the first hidden layer will have twenty (20) neurons in the first layer and thirty (30) in the second layer for the last combination of the data sets. The graph below depicts the global error of the neural network for the training and testing set used for each epoch:

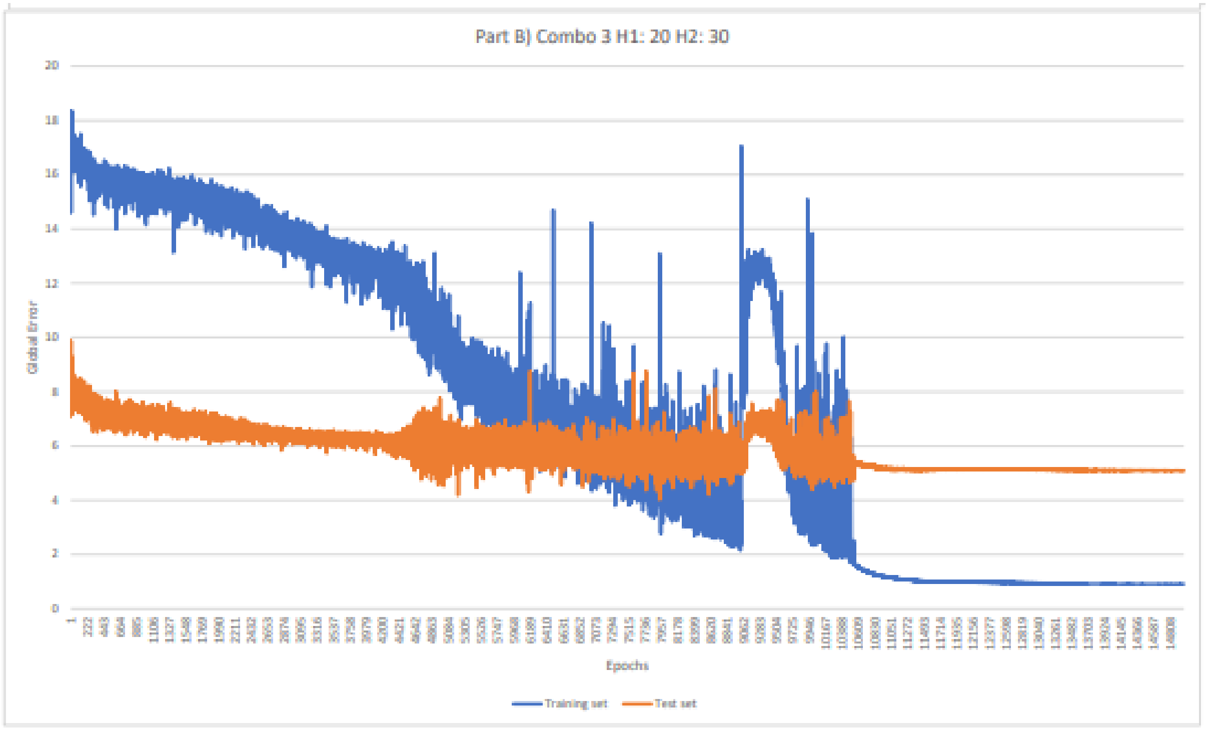


Figure 2.3: Global error per epoch of combination 3

With the final combination, it is a rehash of the first two, with the network classification of the data beginning poorly, but with each epoch, the error decreases as it learns features of the data from the training set. Then at a later epoch, the network begins memorizing the training data causing a divergence between the testing and training data resulting in a convergence in the testing data or an increase in the classification error for that set. Whereas the training sets error continuously decreases until it too converges as the network has memorized the data.

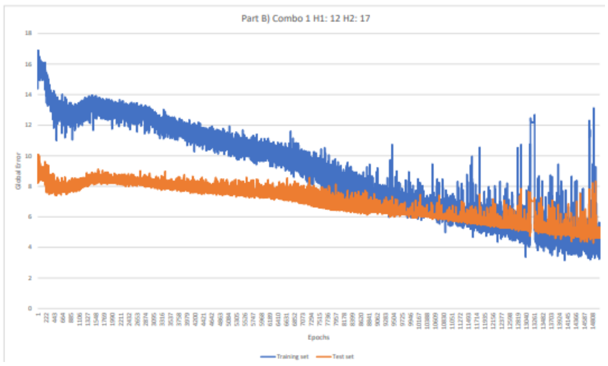
From the above data, we can glean points of interest about the learning abilities of the feed-forward neural network and how the generalization of the test data affects classification.The first point we can note is that the error of the network will begin at a high to mid-level of error depending on the initial network, and as each epoch passes, both the test set and training sets' error will decrease. It is also worth noting that after a certain number of epochs, the mistakes in classification the model makes for each grouping will diverge, most likely due to the network beginning to memorize the data.

On this note, we can say that the amount of generalization present in the network from processing the training data affects the network's classification ability. A high level of generality allows accurate classification of data sets it has not seen before, unlike a low level of generalization in the network, meaning the network has begun to memorize the training data, the model will have a difficult time classifying data it has not seen before unless the inputs of that data are nearly identical to the training data inputs.

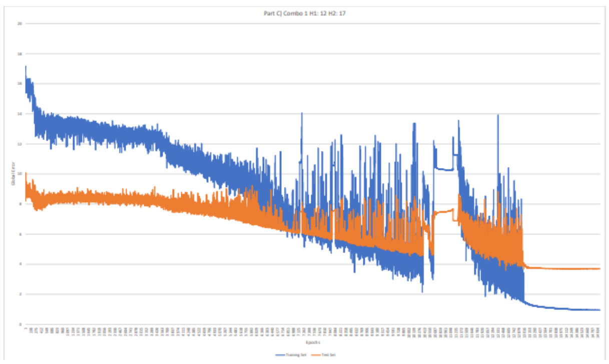
2.3 Assign Part C (FFNN effects of Momentum)

For the second experiment of this report, research into the effects of adding momentum into the feed-forward neural network, this experiment will use a momentum rate of fifty percent (50%). This experiment's goal, examine the effect momentum has on the training rate of the neural network, determining whether this factor has a positive or negative influence on the overall speed of the network. The examinations of the momentum's effect on the training speed, the parameters and data sets used in part B will are reused, to compare how long it takes the training and test set to diverge or for each set to converge on their final error value.

The graph below depicts the global error of the neural network for the training and testing set of combination one for each epoch with momentum added:



(a) No Momentum

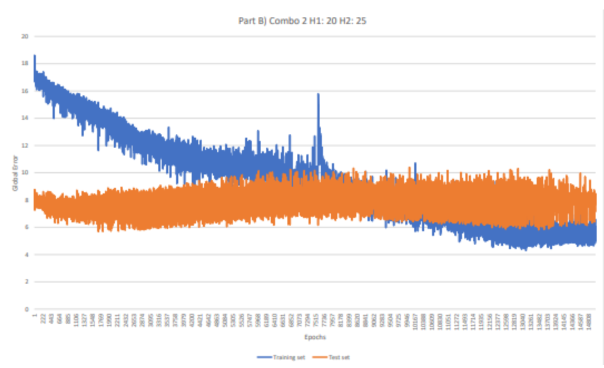


(b) Momentum

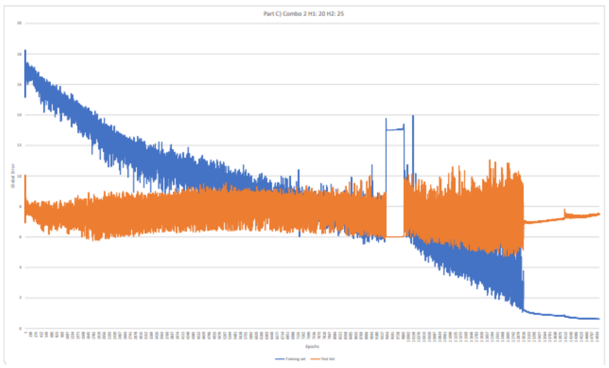
Figure 2.4: Affect of Momentum Combo 1

From the graph above we can see that by adding momentum to the model the error, attained convergence quicker than it did in part B) which did not use convergence in its training.

The graph below depicts the global error of the neural network for the training and testing set of combination two for each epoch with momentum added:



(a) No Momentum

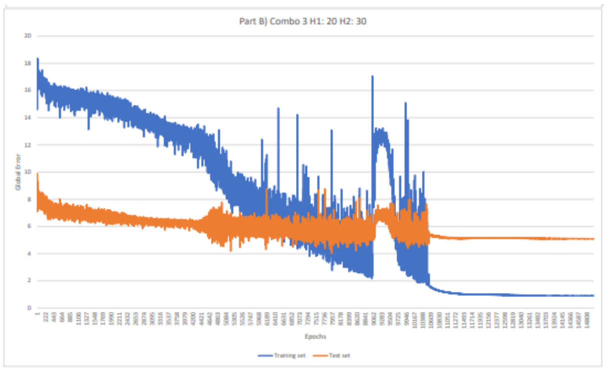


(b) Momentum

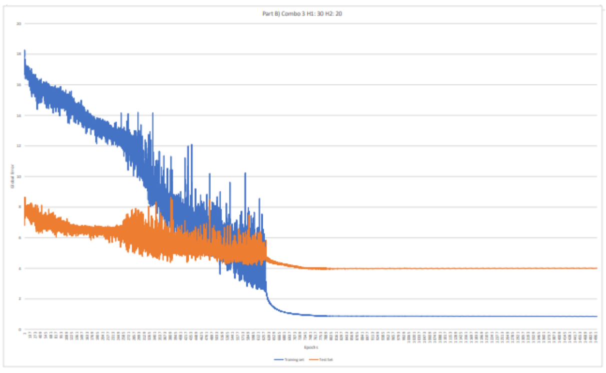
Figure 2.5: Affect of Momentum Combo 2

The same result in the previous graph can be seen here as well, the inclusion of momentum in the training of the network caused the netowrk to reach convergence, unlike its counterpart which was still training.

The graph below depicts the global error of the neural network for the training and testing set of combination three for each epoch with momentum added:



(a) No Momentum



(b) Momentum

Figure 2.6: Affect of Momentum Combo 3

The above figure is a repeat of the previous two experiments as the graph more rapidly achieved convergence than the experiment in part B which did not use momentum.

Thus based on the data above, we can surmise that the inclusion of momentum caused the learning rate of the FFNN to increase, making the network achieve convergence quicker than the neural network in part B that did not use this parameter in its training. This is seen in the above graphs as the networks utilizing momentum had the point of divergence when the network memorized the training data causing the error in the test data to stop decreasing, which occurred at a much earlier epoch. The networks using momentum also reached convergence sooner. As seen in the graphs above, the momentum caused the error of the network to plateau much sooner than the networks that did not use the momentum parameter if they were able, during those epochs, even able to achieve convergence.

2.4 Assign Part D (FFNN Effect of Training Data)

The third experiment of this report will examine the effect different data sets produced the best results overall, allowing an examination of the effect training sets have on the generalization abilities of the feed-forward neural network as it attempts to classify data not part of the training set.

The data for this experiment will come from four files constructed by concatenating two data files with different bin sizes, the data is then fed into the network several times. For each run, the minimum error value of the test training data is collected, as this is the inflection value of the data set (i.e. the error value of the training set before divergence). These values are averaged to determine the average error of each input combination, determining which input had a more positive or negative effect on the overall training of the network.

The four combinations of the data files used during this experiment will be a concatenation between the sixteen (16) and twenty-five (25) data files making a size forty-one (41) bin, sixteen (16) and thirty-two (32) bin to make the size forty-eight (48) bin, the twenty-five (25) and thirty-six (32) bin to make the size fifty-seven (57) bin, and finally, the sixteen (16) and sixty-four (64) size bins to make the size eighty (80) bin.

The first table of values shows the results of the global error per run and the averaged value from the concatenated data set constructed from the files containing sixteen and twenty-five bins. The hidden layers used are H1: 27, H2: 33.

Table 1: Part D) Combo 41 Analysis

Test set Global Error	1	2	3	4	5	6	Average
Combo D) 41 (1)	5.0007	5.0004	5	5.0019	5.0001	5	5.001
Combo D) 41 (2)	2.749	2.092	4.636	2.892	2.825	3.086	3.047
Combo D) 41 (3)	7.96	7.57	7.4	8.1	7.8	7.6	7.7

As we can see in the above data, it is clear that depending on the sets used for training and testing can affect the average classification ability of the neural network. This is observable by comparing the second and third combinations, as the second combination of data correctly classified 4.3 inputs than the third training and testing combination.

The second table of values shows the results of the global error per run and the averaged value from the concatenated data set constructed from the files containing sixteen and thirty-two bins. The hidden layers used are H1: 27, H2: 36.

Table 2: Part D) Combo 48 Analysis

Test set Global Error	1	2	3	4	5	6	Average
Combo D) 48 (1)	5.001	5	5.0003	5.012	5.015	5.0002	5.005
Combo D) 48 (2)	3.5	3.4	2.7	1.4	2.3	4.7	3.01
Combo D) 48 (3)	7.68	6.29	7.99	5.18	6.80	7.68	6.94

For the above data, we get a similar result to the first table, that depending on which combination of the data we use for training and testing can alter the network's ability to classify the data to its correct category. We can see this result with the second and third combinations, just as with the first table, the second combination, on average, was able to classify 3.93 data input more than the third combination.

The third table of values shows the results of the global error per run and the averaged value from the concatenated data set constructed from the files containing twenty-five and thirty-two bins. The hidden layers used are H1: 47, H2: 38

For the third table of data, we again see the trend of the network being able to classify the input data more accurately depending on the data set, for if we compare the average

Table 3: Part D) Combo 57 Analysis

Test set Global Error	1	2	3	4	5	6	Average
Combo D) 57 (1)	4.141	4.483	4.027	4.108	4.074	4.062	4.137
Combo D) 57 (2)	4.37	4.57	4.48	4.4	4.4	4.2	4.4
Combo D) 57 (3)	6.52	7.27	7.18	6.88	7.24	6.4	6.91

error of the first combination compared to the third combination, we can see that the first combo got around 2.8 more of the inputs classified correctly than the third combination.

The fourth table of values shows the results of the global error per run and the averaged value from the concatenated data set constructed from the files containing sixteen and sixty-four bins. The hidden layers used are H1:65, H2:57.

Table 4: Part D) Combo 80 Analysis

Test set Global Error	1	2	3	4	5	6	Average
Combo D) 80 (1)	5.0001	5.0008	5	5	5	5	5.0002
Combo D) 80 (2)	5.0001	5.001	5.00002	5.00001	5.00002	5.000003	5.00015
Combo D) 80 (3)	6.53	5.11	5.05	6.37	5.49	7.37	5.99

For the fourth table, we see a slight divergence from the previous tables since the ordering of the data sets used for training and testing did not impact the amount of error the FFNN outputted. So while the sequence affected the neural network's ability to classify data in previous data files, it had little effect on the data here.

Based on the results recorded in the above table, depending on the ordering of the training and test sets used in the neural network affects the network's ability to classify data. As we saw in the tables above, depending on which sets the network uses for training and testing can significantly affect the classification abilities, as was the case for the first and second tables. We also saw that for some data, the ordering of the data sets had little effect on the network's average ability to classify data. However, this could be due to the model used for the experiment (i.e. the number of hidden layers and the nodes they possess), further experimentation is required to determine if it was a factor.

Also, based on this data, it is evident that the number of input values the data file contains also affects the neural networks' classification abilities. For example, take the table containing analysis on the data file possessing forty-one and the file with fifty-seven bins. The lowest average error between the two files comes from the data file containing forty-eight inputs, as its lowest average error is 3.01. Now by comparing the highest average error, we see that it also comes from the file possessing forty-eight bins, as its largest average error is 6.94. This shows that the data file we use can also affect the learning of the network, as information in the files may be more complex than others, causing difficulty in understanding the data. Alternatively, the data within the file could contain more simplistic data, allowing the network memorizes the traits for each classification more easily.

Therefore based on the analysis above, depending on the data in the files and the sets that are used for training and testing the network can affect the overall classification ability of the network, as files and/or orderings of the data sets may be easier to train the network on, allowing for a lower global error. While other data sets may be more complex, causing the network to struggle to learn each classification's traits, preventing accurate sorting.

2.5 Assign Part E (FFNN Effect of Additional Layers)

For the final experiment of this report, the feed-forward neural network will be expanded from three (3) layers to five (5) layers, allowing an examination of the effects additional hidden layers will have on the feed-forward neural network.

For this experiment, the best and worst cases from part D are used in the analysis to determine if the additional neurons either improved or deteriorated the feed-forward neural networks' generalization abilities. The best case from part D resulted from the second combination for the concatenated file with forty-eight bins, as it had the lowest global error at 3.01. The combination with the worst global error from part D came from the third combination of the concatenated file with forty-one bins, with an average global error of 7.7.

Table 5: Part E) Correctness of Classification

Test set Global Error	1	2	3	4	5	6	Average
Combo D) 48 (2)	4.03	6.48	4.86	4.14	8.23	5.27	5.5
Combo D) 41 (3)	7.26	7.59	7.53	7.62	7.53	7.64	7.53

From the results in the above table it is observable that the addition of hidden layers in the neural network had a mixed reaction to the amount of correct classification for the individual test sets. The test set containing forty-eight bins the additional hidden layers caused the amount of error to increase as the average error increased from 3.01 to 5.5, thus decreasing the amount of correct classification. However, for the combination containing forty-one bins the average global error did not vary much from the network containing only three hidden layers.

Based on the results recorded in the above table we can come to the conclusion that the addition of hidden layers, for this data, did not do much to improve the correctness of classification for the data being analyzed, as it did not create a model that provided the best decision boundary for the data. Therefore, adding more hidden layers to the network in certain cases will not lead to an improvement in the classification of the data, as in some cases it may lead to a loss in accuracy.

3 Conclusion

In conclusion, through the experiments performed in this report on the feed-forward neural network, we have seen the effect generalization, momentum, input data and adding additional hidden layers have on the feed-forward neural network.

The first experiment involving generalization showed that as the FFNN begins to memorize the training data, the test set will diverge from the training network's gradient descent. Thus as the network gets better at classifying the training information, it gets worse at sorting the test data.

The second experiment involving momentum demonstrated that when included in the network, the training speed of the network increases. Shown during the graph comparisons portion of the investigation, where networks with identical parameter configurations, the only difference between the graphs that converged slower and the graphs that consolidated quickly, was that the graphs with quick training times possesed momentum in their training. Thus adding the momentum parameter to the network decreased the time needed for train, allowing for faster convergences in the neural network.

The third experiment of testing the effect of different training files, each having the test and training sets arranged in a different combination, showed that the data used for training has an impact on the neural network's ability to classify data. Simple inputs allowed the neural network to learn characteristics of the data very skillfully, unlike complex data, which the network struggles to organize.

The final experiment demonstrated that adding hidden layers into the neural network does not improve the classification abilities of the neural network for every instance of data. However, in cases it does improve classification, the cost of time to the improvement is insufficient. But that it can also cause a decrease in the accuracy of the neural network's ability to classify data correctly, as shown in our experimentation as the neural network caused the error to increase for one data set and slightly decrease for another, however the time it took to train the network had increased exponentially reducing the value gained from the decreased error.

Therefore based on the results of the information collected from the above experiments on generalization, momentum, training data and number of hidden layers, the feed-forward neural network is a powerful tool that is very useful for classifying large complex sets of linearly separable data. However, the network model will need extensive tempering and testing to ensure that the model used and the training data fed into the network can train the model properly, so it can correctly sort the data into its corresponding category.

List of Figures

List of Tables