Computer Science
COSC4P80 - Artificial Neural Networks

Multilayer Feed-Forward Network

Prepared by: Elya Denysova
Instructor: Dave Bockus
Date: March 16, 2023

Abstract

Feed-forward Neural Network (FFNN) is a type of Artificial Neural Network(ANN) that is multilayered: an input layer, an output layer, and one or more hidden layers in between. It extends learning in perceptions by having additional layers allowing the system to learn more complex data structures. The 'feed-forward' name comes from the acyclic nature of neuron connections. FFNN is just a network structure and to be utilized needs to be trained with some sort of learning algorithm. Back-propagation(BP) is the most popular training algorithm for ANN. It is a supervised learning that utilizes gradient descent search to reduce global error. There are many parameters that can be set like learning rate and hidden layer size or updated dynamically. There are multiple differentiable functions that can be used as an activation functions, in this assignment sigma function was used. The goal of this paper is to evaluate how FFNN behaves with different input data sets and how parameters of network structure and BP affect the performance of the trained model. First part A is focused on the basic functionality of ANN. Next part B discusses the importance of a test set and how the size of the hidden layer impacts the accuracy of a model. Part C demonstrates one of the add-ons to the BP - momentum and how it can improve the time required for the network to converge. Part D is dedicated to experimentation with input data. Finally, part E introduces a new network structure - 5-layer FFNN. To analyze all the aspects of FFNN and BP mentioned above, the system was thoroughly tested and additional input files were created along with corresponding configuration files. Not all results were as expected. On some inputs, the gradient descent was not converging and was sometimes completely distorted. Dynamic learning rate and momentum certainly have a strong relation to improving the speed of convergence. Extending FFNN to 5 layers produced surprisingly good results. Multiple layers certainly helped the model with input that had a small input vector size. The implementation section provides instructions on how to run code. The GitHub repository contains code for multilayer FFNN and BP. Although in this paper network was tested on at most 5 layers (3 hidden layers), the number of layers can be set up in the configuration file to any arbitrary number.

Contents

1 Introduction

FFNN has many applications such as classification, pattern recognition, regression, etc. This paper analyses FFNN for the task of classification. The goal is to train FFNN to classify the quality (good or bad) of motors based on the sound frequency spectrum they produce. There are multiple data files available with varying numbers of bins ranging from 16 to 1000.

FFNN works on the principle of neurons in the human brain. Each element receives the input, pushes it through the activation function, and outputs the calculated value. The following graph demonstrates the typical structure of an artificial neuron.

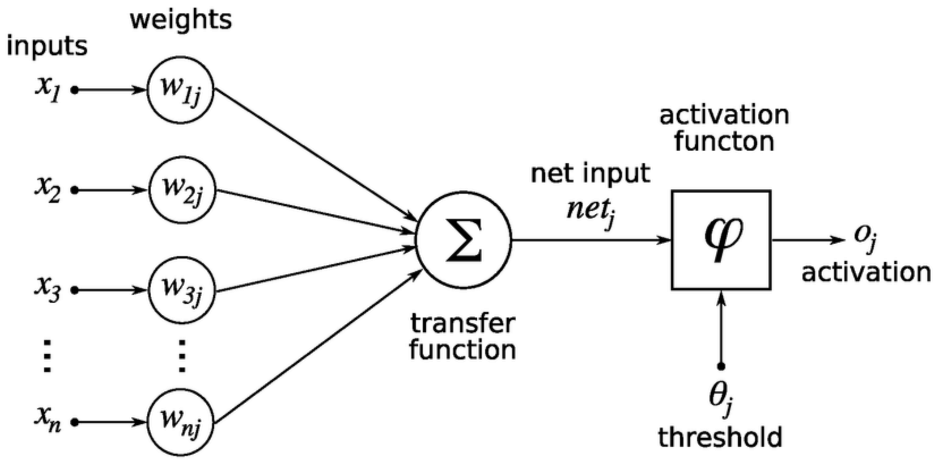

Figure 1.1: Artificial neuron structure

The human brain uses electronic impulses to send information. Similarly, FFNN can use values 1 and 0 to perform classification such that 1 corresponds to one class/type, and 0 - another. There are multiple layers including an input layer, an output layer, and some number of hidden layers. Each layer has a one-way connection with neighboring layers. Consider the following image of a usual Feed-forward architecture.

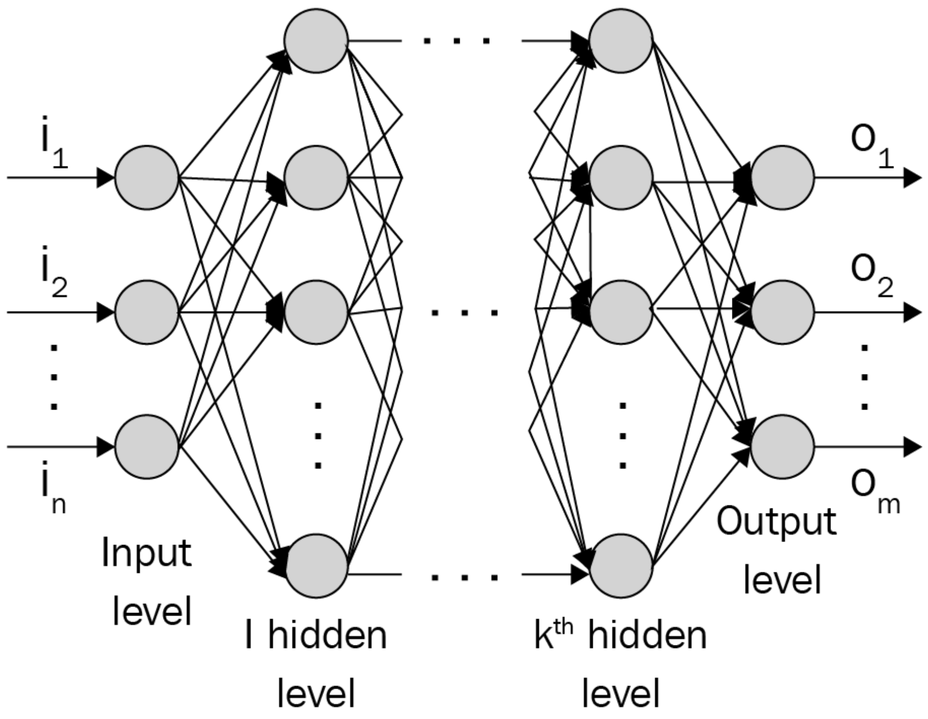

Figure 1.2: Feed-forward network layout

BP teaches FFNN based on provided training examples. BP evaluates FFNN output and compares it to the expected output. Then, it propagates the error backward (from the output layer to the input layer) and readjusts weight coefficients based on that comparison.

FFNN and BP were assembled in java utilizing OOP concepts. FFNN has the capability to initialize the network and compute forward pass. BP dictates what values are used in initialization and training. More implementation details are in part A.

2 Part A

2.1 Experiment setup

FFNN architecture consists of 4 classes: FFNN, Layer, and Neuron. The neuron represents the smallest fundamental structure. It has an output value, bias value, and an array of weights(weights between the layer the neuron belongs to and the next layer). Weights are initialized randomly between -0.5 and 0.5 excluding 0. The layer class keeps track of the neurons. The size of the layer (number of neurons) depends on the type of layer and other parameters. The input layer size depends on the data file used, Output layer size is 1 (0 - good, 1 - bad). To get classification, the output of the network is rounded to the closest whole value. The size of a hidden layer is read from a configuration file and can be arbitrary, but typically is about 70% of the input layer size. BP is implemented in BackProp class. It takes 3 file names as parameters for the configuration file, data file, and output file. Config description file defines line by line what parameter value is in a config file. The initial value for the learning rate is set in a config file and dynamically updates during the execution according to the direction of a gradient. The learning rate value is in bounds (0,1). After conducting a few experiments, it was clear that when the learning rate is dynamic, ANN converges faster and more consistently. Global error is calculated per each epoch as cumulative of absolute error values per each training example and is printed to the output file. Formatting is applied to error values to cut down decimal places and change symbols for separating decimal values from dot to coma (for Excel).

2.2 Experiment results

The graph in figure 2.1 demonstrates how data shuffling affects convergence.

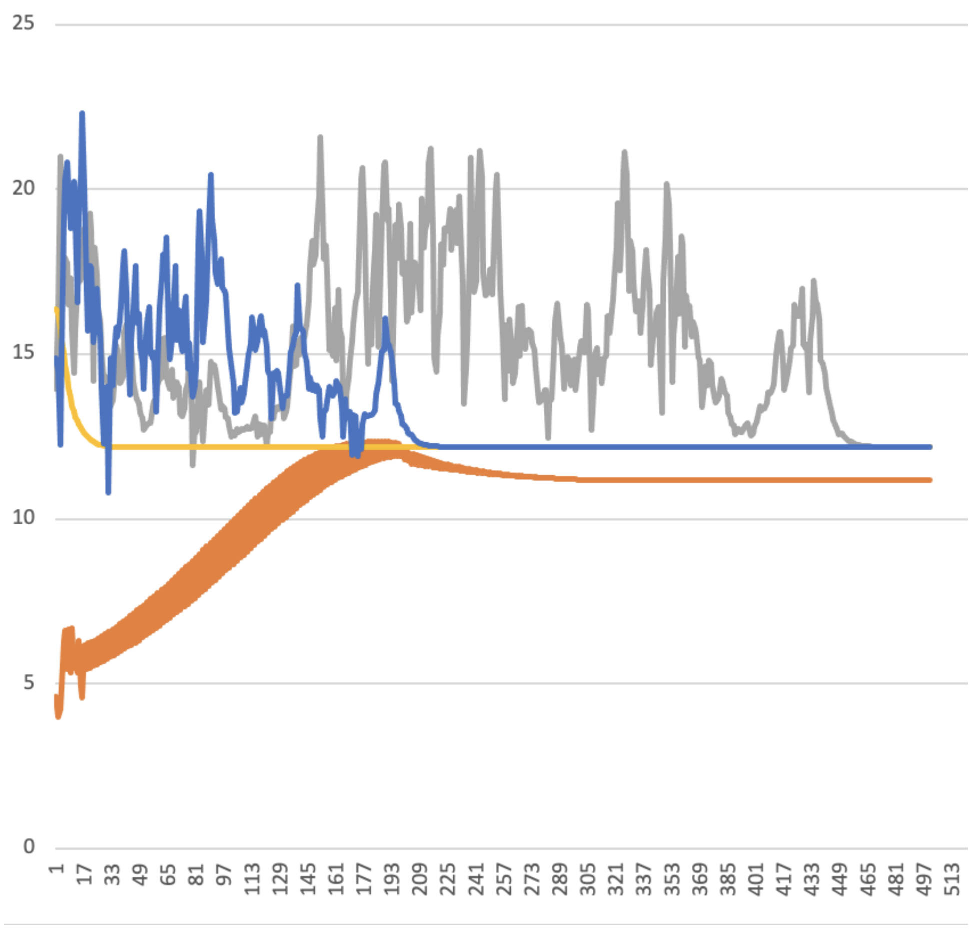

Figure 2.1: Convergence graph whole data set

Orange - data is not shuffled Gray - data shuffled at 20% swap rate every epoch Yellow - data shuffled once before training Blue - data shuffled completely before training and then every epoch at 20% swaps

The original data has all the examples of good motors before examples of bad motors so the error curve is very distorted. The yellow line has a very smooth gradient, however, the fact that there is almost no energy in the system, might have a very high chance of falling into a local minimum. From the comparison of the blue and gray gradient, it is prominent, that the addition of shuffling every epoch cut training time to converge in half.

3 Part B

3.1 Experiment setup

Considering the code for the previous sections, it really is just a memory. The errors were only reported for the training round which is not representative data to evaluate the network because, in reality, ANN is useful for new, unknown tasks rather than the same training datasets. The next improvement was to distribute data into 3 lists: Training set 1, Training set 2 and Testing. It is important that all sets have mutually exclusive data (especially training VS testing sets) and have examples of both good and bad motors. To keep it consistent the proportion of good and bad motors examples is preserved. Now, that there is proper shuffling and test set to verify results, another important parameter needs to be established. Input and output layers dimensions are based on the context of a problem, but what about the hidden layer? The size of a hidden layer can be different and set as one of the parameters of the configuration file.

3.2 Experiment results

The following images contain graphs that show error gradient for runs over data with 150 bins for the hidden layer size variations: 95, 105, 135, 150, 165. Dark and light blue - training sets, Green - test sets. These graphs demonstrate the important role of a test set. On each gradient, there is an identifiable divergence point - where the test set error starts increasing, while the training set error continues to converge. After the divergence point, the gradient for both trainings sets approaches and then flattens at almost 0. This is an epoch at which FFNN starts memorizing instead of learning. It is a good indication of when to stop training. The over-trained network produces great accuracy results for the training examples but does a very bad job classifying new, unseen data. The test set basically allows evaluating how well FFNN generalizes.

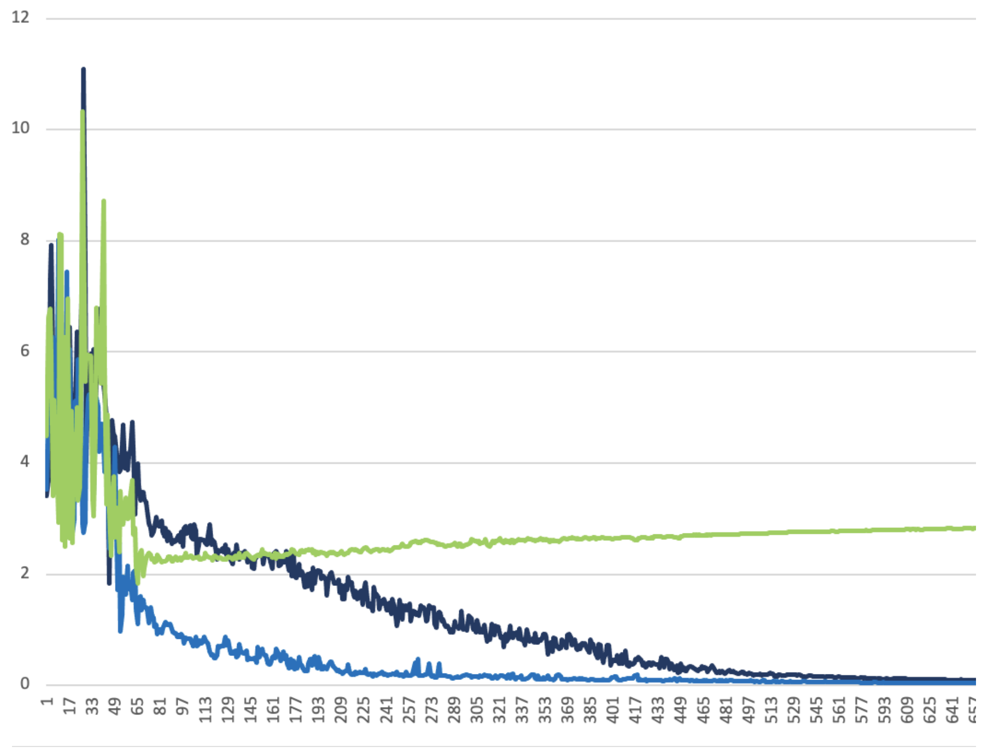

Figure 3.1: Convergence graph: hidden layer size 95

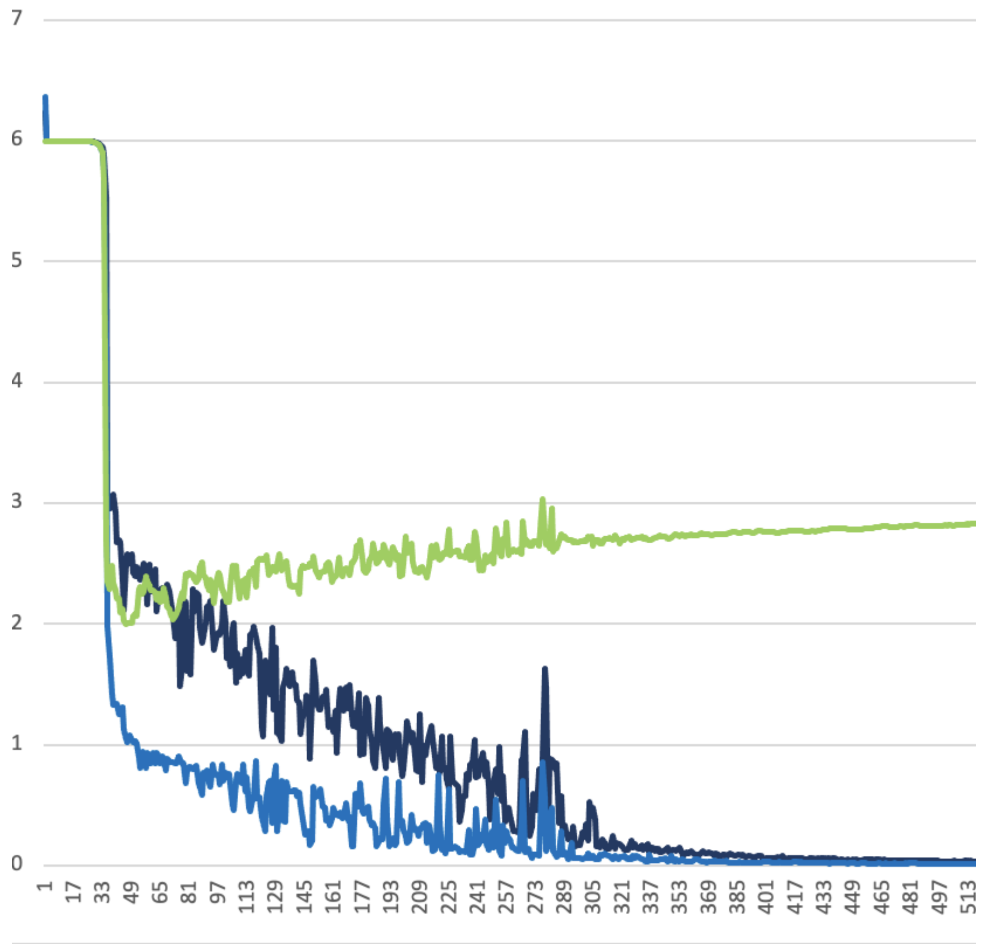

Figure 3.2: Convergence graph: hidden layer size 105

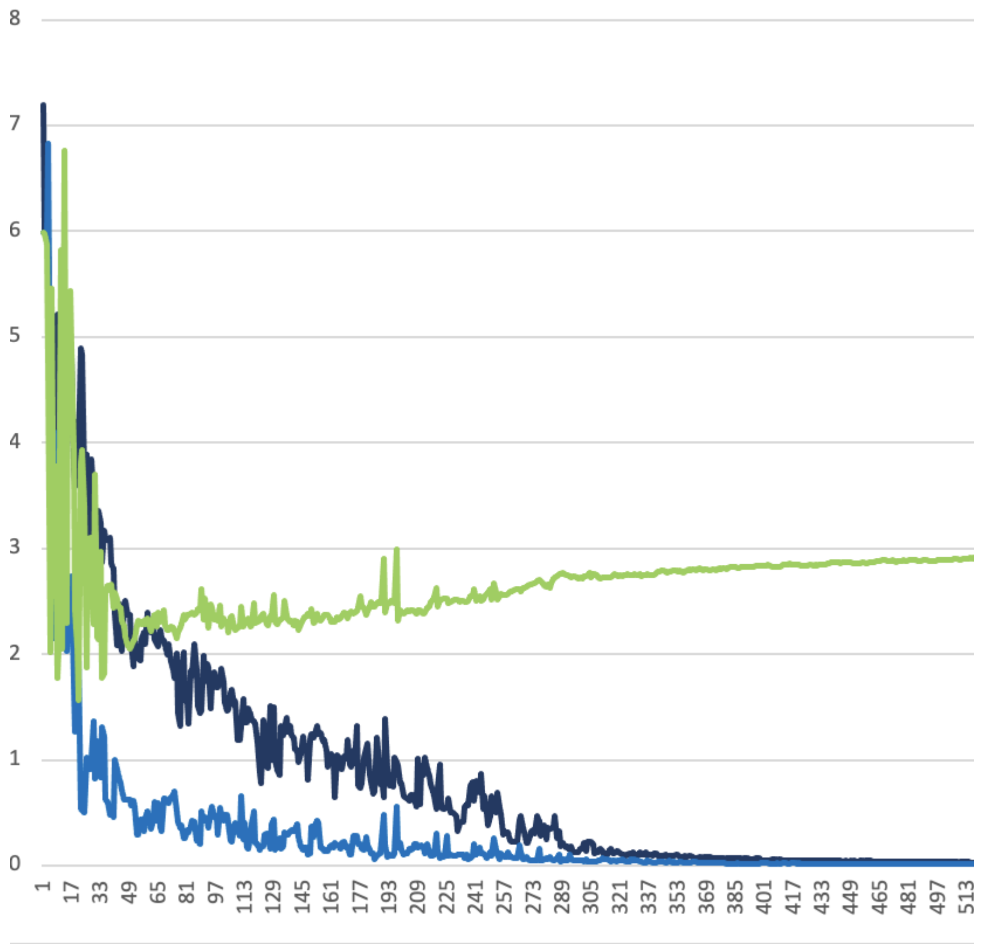

Figure 3.3: Convergence graph: hidden layer size 135

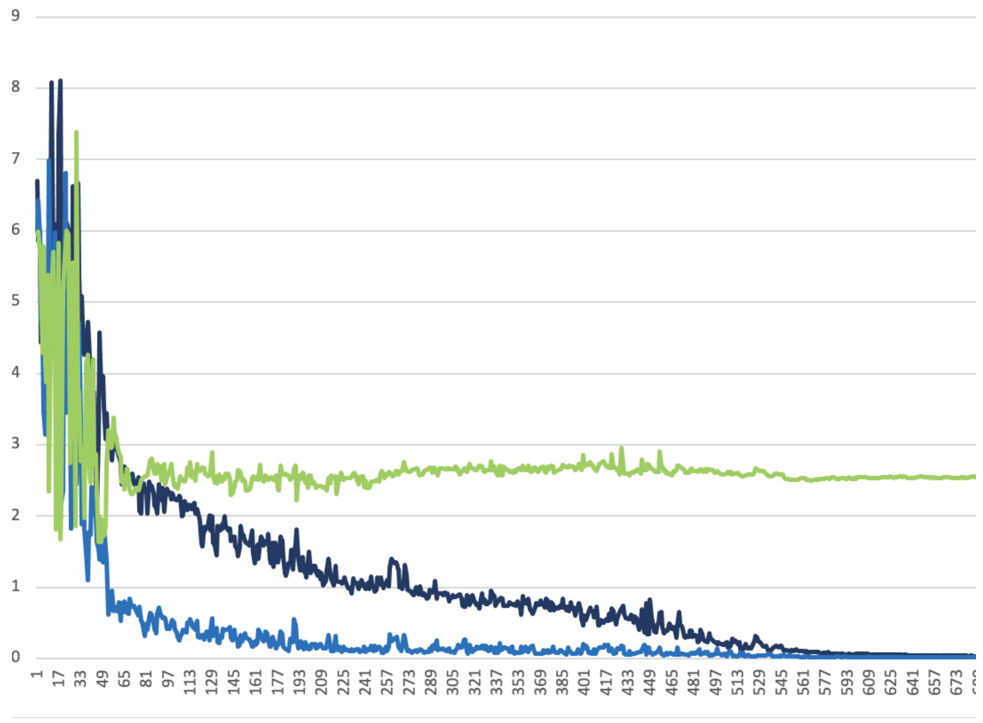
Figure 3.4: Convergence graph: hidden layer size 150

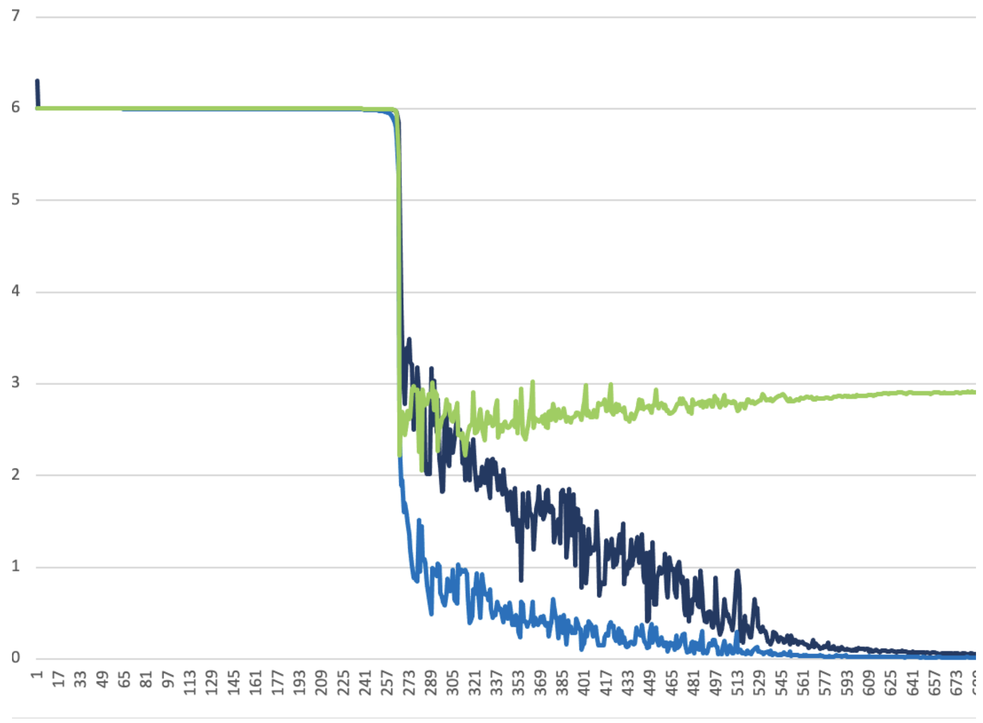
Figure 3.5: Convergence graph: hidden layer size 165

When the size of the hidden layer is 105, 135, and 150 the divergence point comes before epoch 100 with an error value higher than 2. With a hidden layer size of 95, the divergence point happens at an epoch closer to 200. Even though the convergence happened later, the actual error value is much closer to 95 and was probably not sufficiently large. The last graph is a good demonstration of how FFNN can get stuck in the local minimum. The error gradient curve flattens quickly but at a high error value, then drops significantly. Perhaps a dynamic learning rate allowed the network to jump out of the local minimum. To investigate further and determine the best dimension for the hidden layer, the network was tested with the size of the hidden layer set to 70%, 85%, and 100% of input vector size.

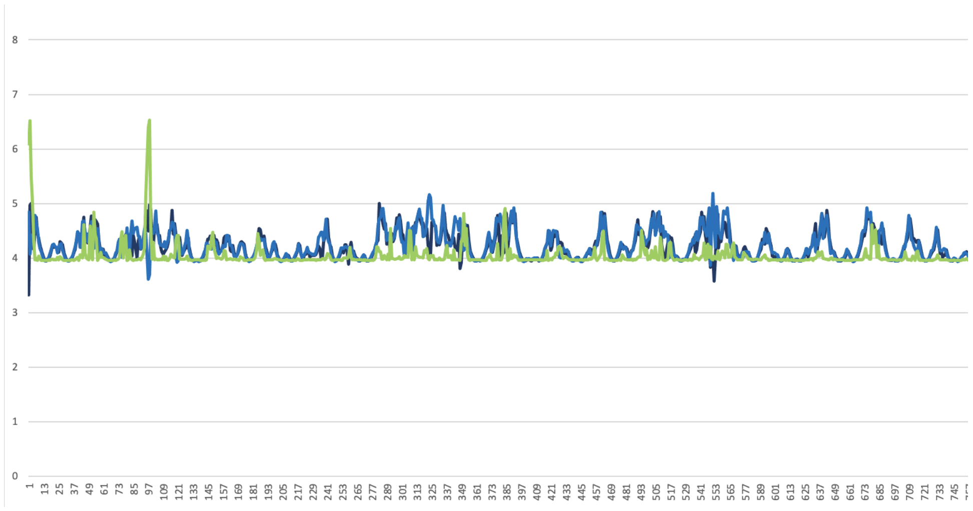

Figure 3.6: Convergence graph: hidden layer size 12

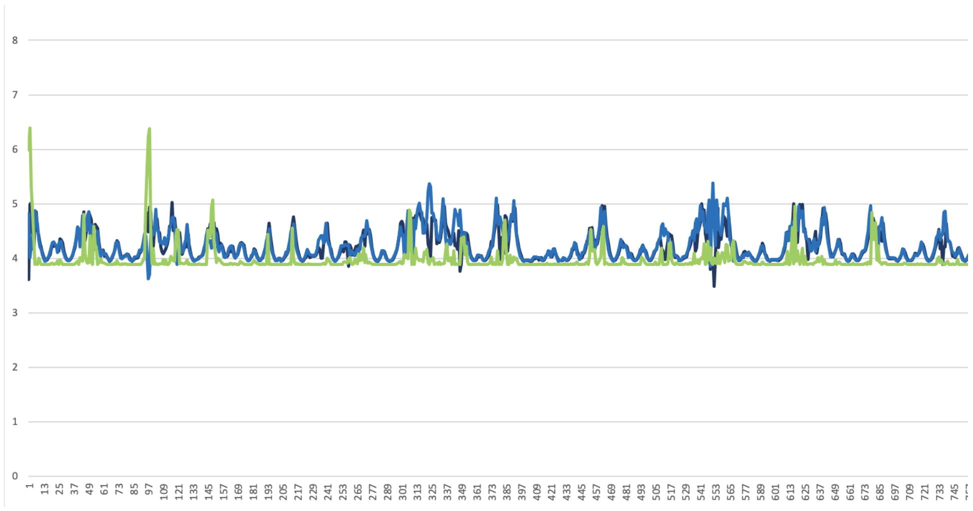

Figure 3.7: Convergence graph: hidden layer size 14

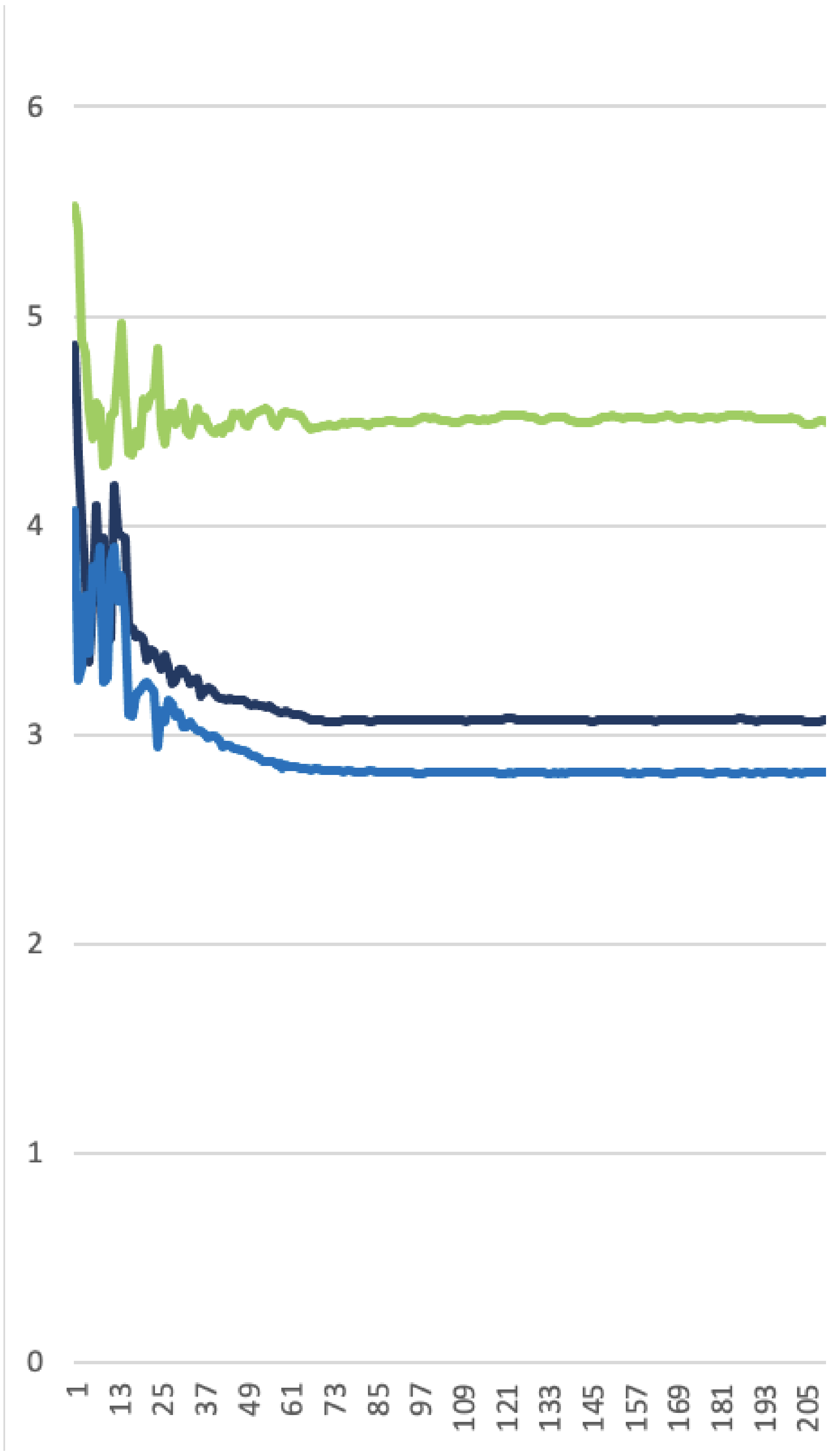

Figure 3.8: Convergence graph: hidden layer size 16

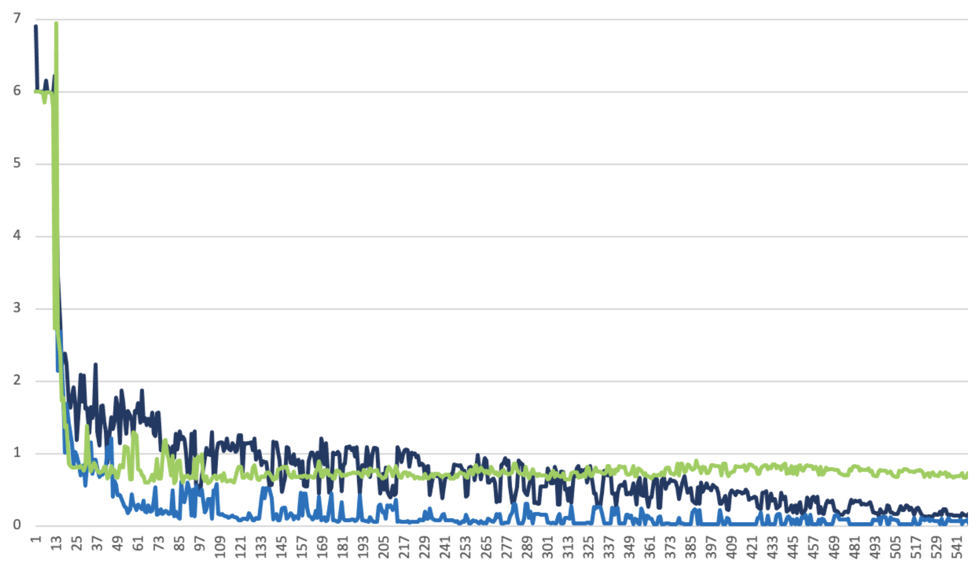

Figure 3.9: Convergence graph: hidden layer size 850

Graphs 3.6 - 3.8 show results for data with 16 bins, and images 3.9 - 1000 bins. FFNN with 16 bins only converged with a hidden layer of size 16. The sample is too small, and even after 10000 epochs, the network was not able to generalize. On the other hand data with 1000 bins converged well for 850 sizes hidden layer, but 700 and 1000 had a very distorted result. Overall, data should have enough attributes to generalize, but if it is too large it might take longer to converge to the global minimum.

4 Part C

4.1 Experiment setup

To improve the learning speed of an FFNN, momentum is introduced. When training with momentum, a portion of a previous change is added to the current change. It helps in avoiding local minima. Like the learning rate, the momentum is between 0 and 1 and decreases as the error is reduced. It is important to decrease the momentum or it might not be able to reach the global minimum.

4.2 Experiment results

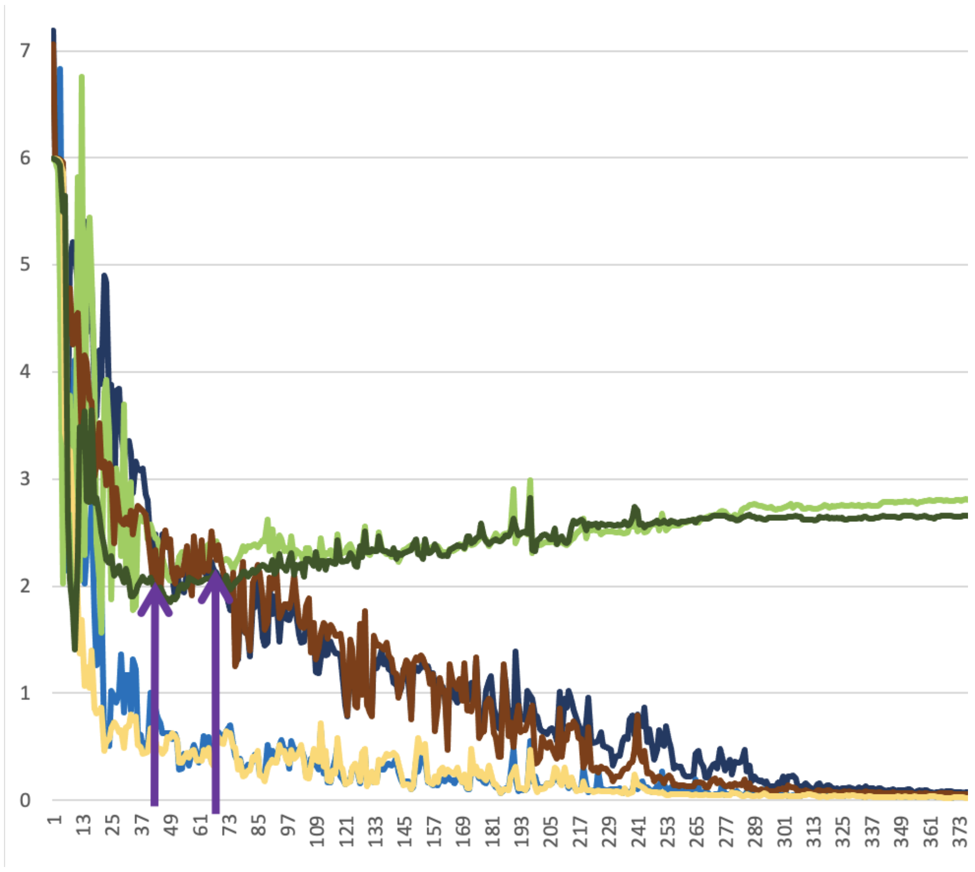

Figure 4.1: Convergence graph: hidden layer size 135: With Vs Without momentum

Dark and light blue - training set without momentum Light green - test set without momentum Brown and yellow - training set with momentum Dark green - test set with momentum

Network was tested on the same data and configuration as from part B with hidden layer size 135 with the addition of dynamic momentum = 0.15. Arrows on the diagram point to the divergence point. Arrow closer to the left points to the divergence point for a network with momentum. Not only it comes faster, but the actual error value is lower too. It proves the premise that momentum can reduce required training time.

5 Part D

5.1 Experiment setup

The effectiveness of a final trained network depends on many parameters of BP and FFNN. However, data plays a significant role. The selection of diverse, but representative data for training is crucial. This section evaluates the network for different input data sets while other parameters are kept the same. Besides 6 original data files, FFNN was tested on a few more files generated by combining data from the original files.
1. 16 bins original

2. 25 bins original
3. 32 bins original
4. 64 bins original
5. 150 bins original
6. 1000 bins original
7. 16 bins rows from 1. and 2. combined
8. 150 bins rows from 5. and 6. combined
9. 32 bins rows from 3. and 4. combined
10. 32 bins rows from 5., 4. and 3. combined
11. 41 bins columns from 1. and 2. combined
12. 96 bins columns from 3. and 4. combined
13. 1150 bins columns from 5. and 6. combined

5.2 Experiment results

Overall, only a few networks converged properly but some inputs were not sufficient for the system to train on.

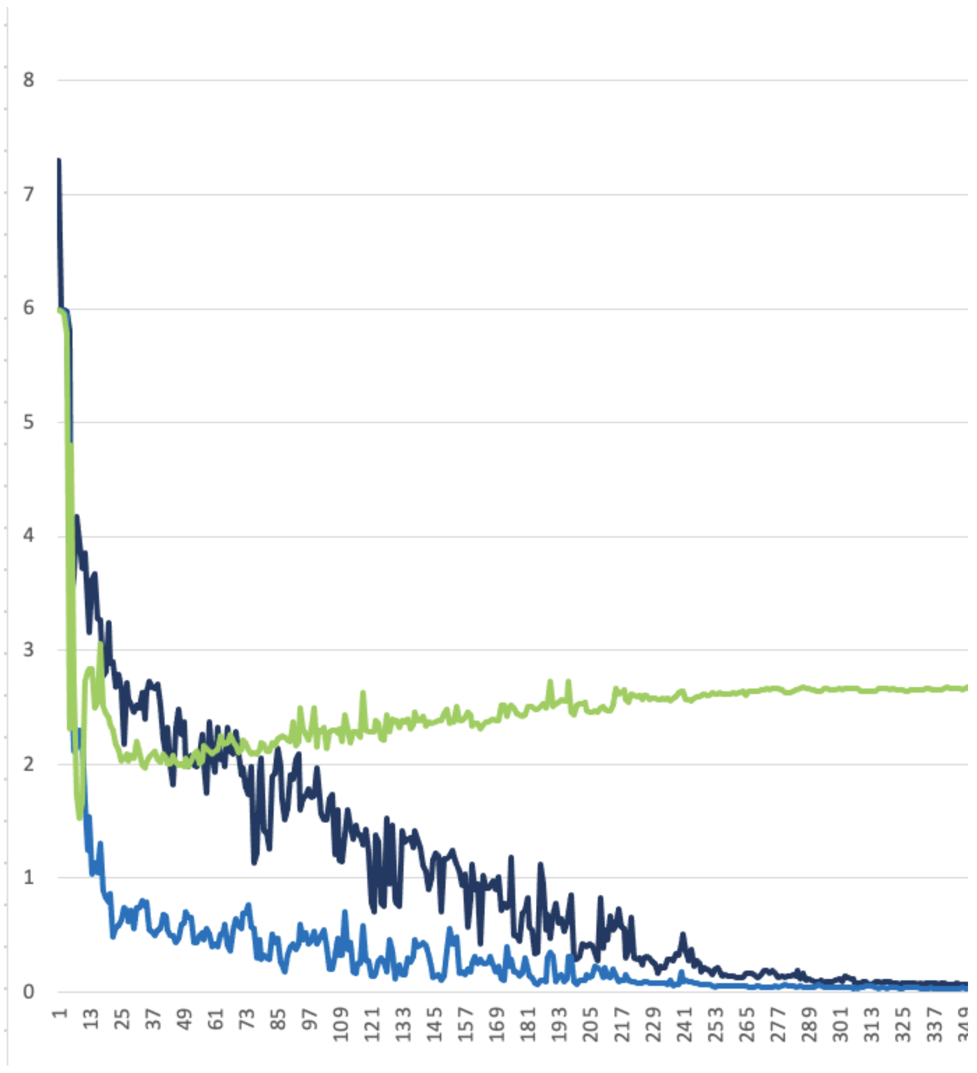

Figure 5.1: Convergence graph: input file 5

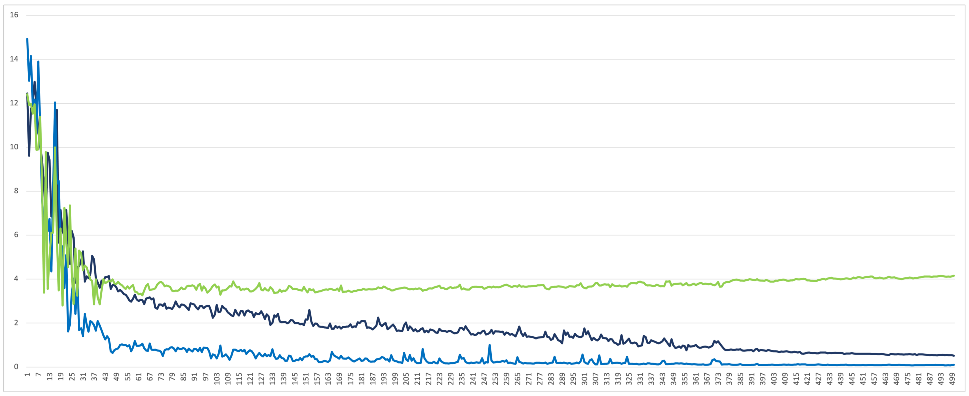
Figure 5.2: Convergence graph: input file 8

Figures 5.1 and 5.2 demonstrate good convergence with descending gradient and visible divergence points. Both files contained data distributed with 150 bins. Perhaps, 150 is a good amount of features for the network to learn to be able to do a generalized classification. The following images are some examples of bad convergence.

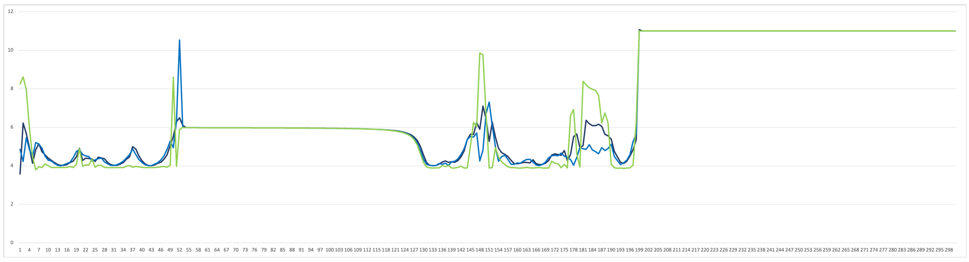
Figure 5.3: Convergence graph: input file 3

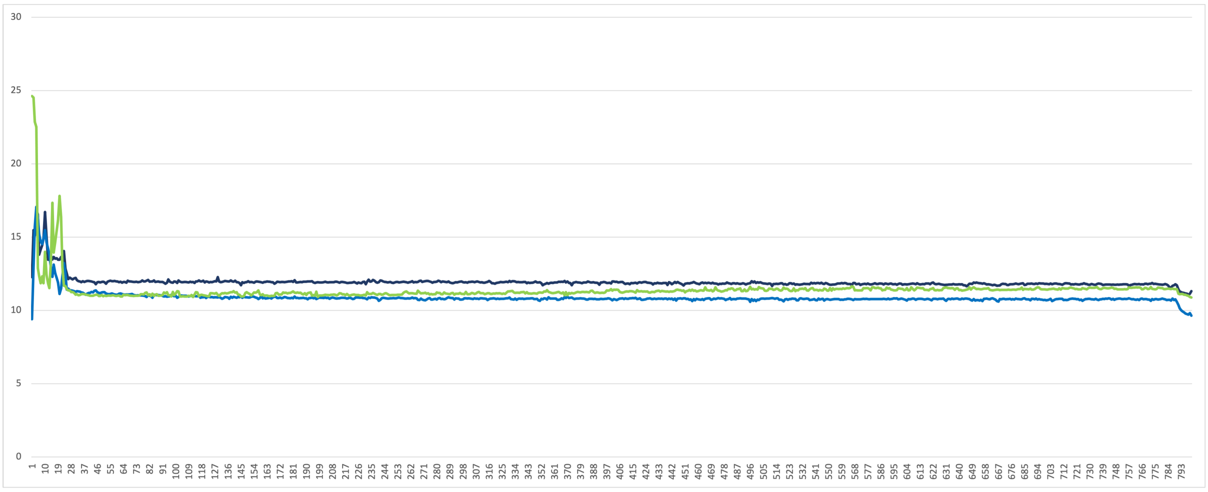
Figure 5.4: Convergence graph: input file 10

In these cases network was not able to converge. Both input files have 32 bins. The first graph is really distorted, but the second looks more promising and yet it doesn't reach the divergence point even after thousands of epochs.

6 Part E

6.1 Experiment setup

FFNN is not restricted to a single hidden layer. In fact, the network can be extended to have multiple hidden layers. This increases the complexity of a model so that it can represent more complex data. Multiple layers can improve accuracy because each feature of the input vector can be considered in greater detail improving the training of the network. Additionally, each hidden layer can have unique dimensions allowing more control over the FFNN structure. For example, for input vector size 25, a 5-layer network could look like this: 25-28-40-20-1. BP for such a network doesn't change fundamentally. The error propagates through each layer.

For this part configuration file is changed. Now, the user can provide a number of hidden layers, and then specify a size for each.

6.2 Experiment results

Figures 6.1 and 6.2 contain graphs from good convergence results from part D compared to results of using 5 layer network to train on the same input.

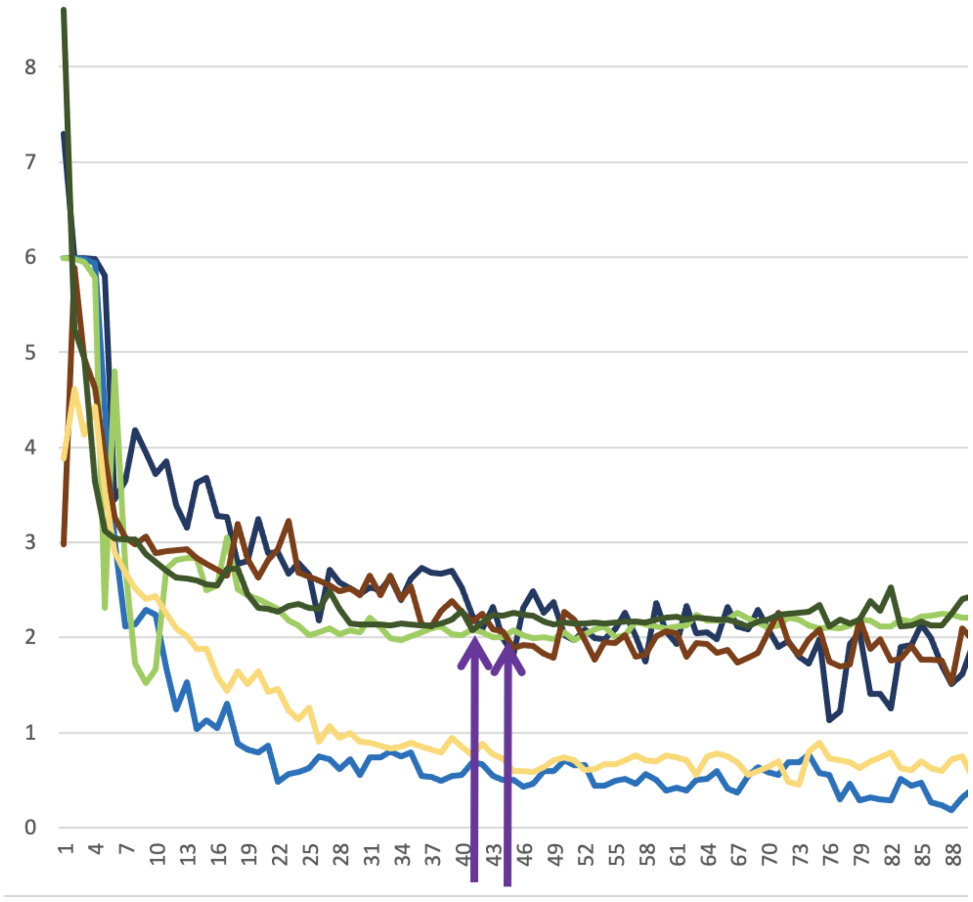

Figure 6.1: Convergence graph: input file 5: 3 layer FFNN Vs 5 layer FFNN

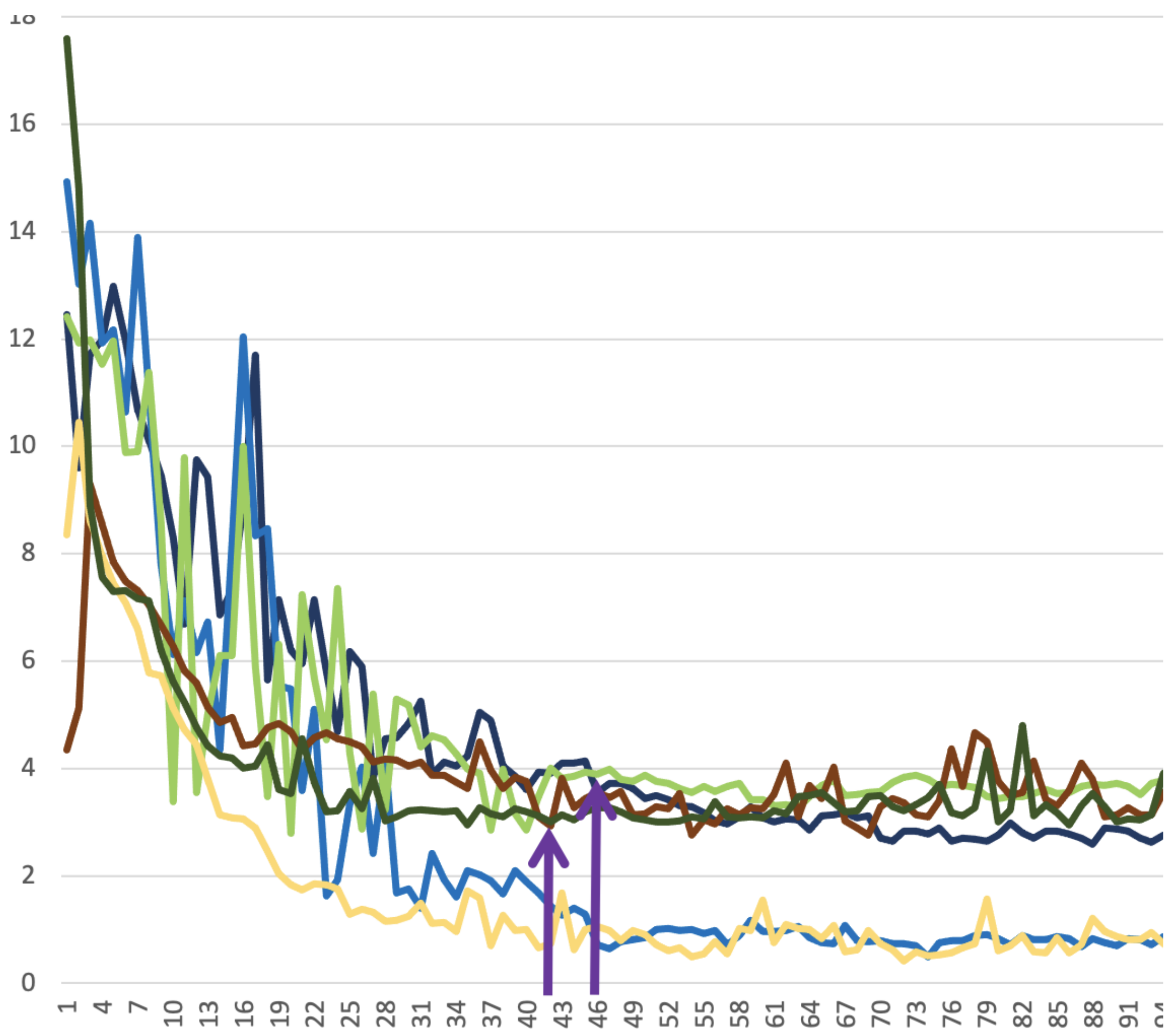

Figure 6.2: Convergence graph: input file 8: 3 layer FFNN Vs 5 layer FFNN

Dark and light blue - training set 3 layers Light green - test set 3 layers Brown and yellow - training set 5 layers Dark green - test set 5 layers

Arrows are pointing to divergence points. In both cases, divergence points came in earlier epoch for 5 layers than 3 layers network. For input file 8, 5 layer network was also able to achieve a lower error rate which is a sign of improvement. Now, FFNN needs to run again and stop at the epoch before the divergence point, right before the network starts to memorize as opposed to generalizing. The following table depicts classification results for the 5-layer network with input file 8.

Expected	Output	Correct?
0	0.0	yes

Continued on next page

Table 1 – continued from previous page

Expected	Output	Correct?
0	0.0	yes
0	0.0	yes
0	0.0	yes
0	0.0	yes
0	0.0	yes
0	0.0	yes
0	0.0	yes
0	1.0	no
0	0.0	yes
0	0.0	yes
0	0.0	yes
0	0.0	yes
0	0.0	yes
0	0.0	yes
0	0.0	yes
0	1.0	no
0	0.0	yes
0	0.0	yes
0	1.0	no
0	0.0	yes
0	0.0	yes
1	1.0	yes
1	1.0	yes
1	1.0	yes
1	1.0	yes
1	1.0	yes
1	1.0	yes
1	1.0	yes
1	1.0	yes

Continued on next page

Table 1 – continued from previous page

Expected	Output	Correct?
1	1.0	yes
1	1.0	yes
1	1.0	yes
1	0.0	no

Accuracy of classification is at 88% over the test set which is a great value showing that this model can actually be useful on new, previously not seen data.

Now, let's consider examples that didn't produce good results in part D: input files 3 and 10.

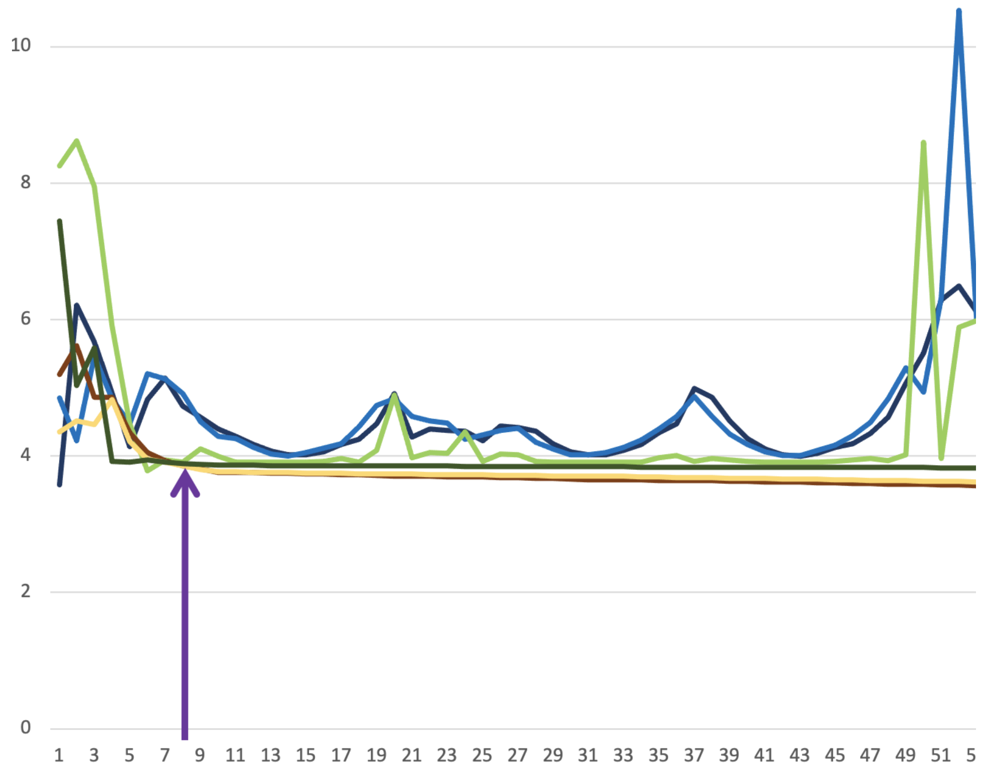

Figure 6.3: Convergence graph: input file 3: 3 layer FFNN Vs 5 layer FFNN

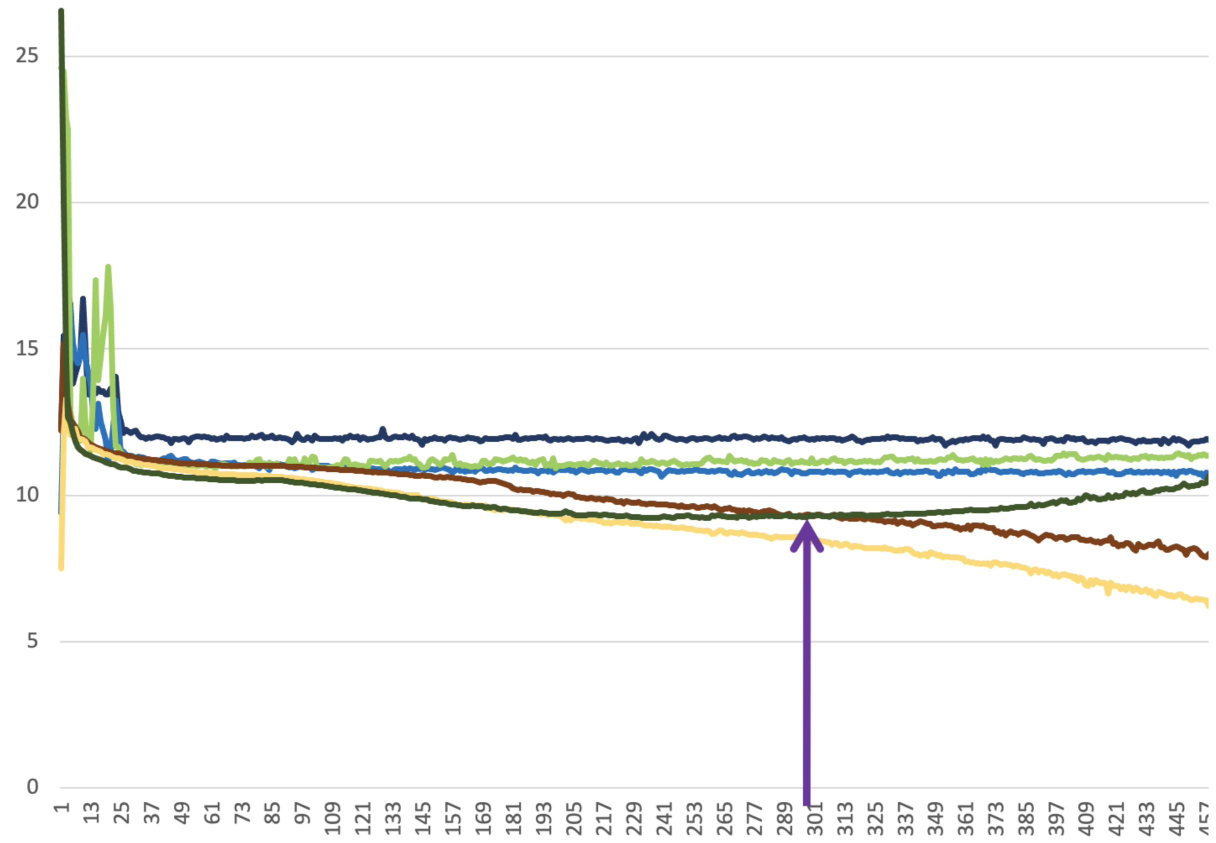

Figure 6.4: Convergence graph: input file 10: 3 layer FFNN Vs 5 layer FFNN

An improved 5-layer network was able to learn these inputs and achieve convergence. The difference in both cases is prominent. As opposed to random distorted graphs of old data, the gradient descent is clearly visible in the images.

Again, set the epoch right before the divergence point to get the optimal model. Instead, the trained model can be stored after training and used for classification separately if needed.

Expected	Output	Correct?
0	0.0	yes
0	0.0	yes
0	0.0	yes
0	0.0	yes
0	1.0	no
0	0.0	yes
0	0.0	yes

Continued on next page

Table 2 – continued from previous page

Expected	**Output**	**Correct?**
0	0.0	yes
0	0.0	yes
0	1.0	no
0	0.0	yes
0	0.0	yes
0	0.0	yes
0	0.0	yes
0	0.0	yes
0	0.0	yes
0	0.0	yes
0	1.0	no
0	1.0	no
0	0.0	yes
0	1.0	no
0	0.0	yes
0	0.0	yes
0	0.0	yes
0	0.0	yes
0	0.0	yes
0	0.0	yes
0	0.0	yes
0	1.0	no
0	1.0	no
0	1.0	no
0	1.0	no
0	1.0	no
0	0.0	yes
1	0.0	no
1	0.0	no

Continued on next page

Table 2 – continued from previous page

Expected	Output	Correct?
1	1.0	yes
1	0.0	no
1	1.0	yes
1	1.0	yes
1	1.0	yes
1	1.0	yes
1	1.0	yes
1	0.0	no
1	1.0	yes
1	1.0	yes
1	1.0	yes
1	1.0	yes
1	1.0	yes
1	1.0	yes
1	1.0	yes
1	1.0	yes
1	1.0	yes

Clearly, there are more correctly classified examples than wrong and the accuracy is at about 73% which is a good result since 3 layer network did not converge with this input at all.

7 Implementation

The final implementation consists of 4 java classes and requires a minimum of 2 additional files: configuration, and data. The output is written to another file, while the classification output is written to the terminal on the final epoch. BackProp class contains a runnable main method.

8 Conclusion

In conclusion, this paper provides a detailed analysis of FFNN trained with BP. Coding such a structure requires attention to detail and persistent debugging. Implementation can be done using matrices or classes with arrays.

There are many parameters that affect the performance of the model, which were examined with multiple data files. There are a few add-ons that can help achieve convergence such as data shuffling, dynamic learning rate, momentum, multiple hidden layers, etc. Nonetheless, selecting a large representative data set is essential. Testing takes time. Especially if data has a long input vector or if the model has multiple large hidden layers.

In the end, there were developed successful models with an expected accuracy rate above 70%. When looking for a divergence point, it is good to set a large epoch. Then, based on the gradient descent graph, the required number of epochs can be adjusted to get the best model (one that produces an error rate over the test set equal to the global minimum).

List of Figures