

# Artificial Neural Network Coursework

## Table of Contents

Which Files Do What? .....	3
Data Processing.....	3
Removing Erroneous Data .....	3
Removing Impossible Data .....	3
Feature Constructing.....	4
Removing Outliers.....	5
Removing Irrelevant Predictors .....	5
Splitting Data Sets.....	7
Standardising the Data .....	7
Final Format.....	7
MSE vs. RMSE .....	8
Back Propagation Algorithm Training On ANN.....	8
Representation of the ANN .....	8
Forward Pass .....	8
Matrix Use .....	8
Backwards Pass .....	8
Updating Weights and Biases.....	9
Momentum .....	9
Bond Driver .....	9
Annealing.....	9
Weight Decay.....	10
Batch Learning.....	10
Activation Functions.....	10
Testing.....	10
Running of Tests .....	11
Results Storing.....	11
Test Processing .....	13
Comparing Algorithms .....	15
Different Modifications .....	15
Optimal Modifications .....	16
LINEST vs. ANN .....	17
Program Time Optimising .....	18

My Approach.....	18
Removing Small Repeated Variable Declarations.....	19
Not Using Some Functionality Towards Forwards Passing .....	19
Forwards Pass Function .....	20
Matrix Use and Small Repeated Variable Declarations.....	20
References .....	20

## Which Files Do What?

Although looked at in Excel, most of the functionality has been completed within python.

This list is mostly based on headings throughout the report:

- dataProcessor.py – Removing erroneous data, removing impossible data, feature constructing, removing outliers,
  - o This program exists, despite looking at the data on Excel, so that python code can systematically do all of the above functionality rather than manual work.
- dataSplitter.py – Splitting data sets, standardising the data,
- ai.py – All of back propagation algorithm training, extra data results storing (see page 13),
- aiRunner.py – Running of test, results storing,
- resultsProcessor.py – Test processing.

Comparing algorithms was done on Excel using the outputs from the code above.

## Data Processing

### Removing Erroneous Data

The first stage of cleaning the data was removing records in the dataset with non-number values (e.g., “” or “a”).

All this took was some conditional formatting in Excel to highlight non-number values. This is shown in Figure 1.

0.388	0.975	0.0575	33.39	0.35	36.6
0.428	1	0.0333	22.53	a	52.5
0.575	0.985	0.0961	94.38	0.34	34.3
0.699	0.965	0.0682	62.55	0.32	33.1
0.645	0.954	0.1198	37.58	0.34	28.9
0.476	0.996	0.0382	35.19	0.51	40.6
0.367	1	0.0599	14.28	0.52	36.6
0.623	0.984	0.0214	9.42	0.42	39.3
0.625	0.991	0.0666	19.69	0.35	35.8
0.561	0.975	0.0316	21.35		38.3
0.531	0.999	0.0348	42.61	0.44	36.2
0.598	0.973	0.1578	25.54	0.31	28.6
0.406	0.901	0.0553	30.8	0.69	57.4

Figure 1 – Erroneous Data Formatting

### Removing Impossible Data

Next, I removed impossible data values. Simple errors were the “-999” values scattered around the dataset.

After researching all the variables (“AREA”, “BFIHOST”, “FARL”, “FPEXT”, “LDP”, “PROPWET”, “RMED-1D”, “SAAR”, “Index flood”), I found that they all non-negative. This made locating the “-999” even easier as highlighting anything negative found those types of errors all at once.

From the research, I discovered that “PROPWET” is a percentage value with the range 0 to 1. This makes the value “1.74” impossible.

Those are all impossible data values.

To ensure that the rest of the headings didn’t have impossibly large values, I used the “Flood Estimation Handbook 5 (Centre for Ecology & Hydrology, 2008)” (referenced at end) to see if values greater than the greatest in the dataset exist in the real world. They do.

## Feature Constructing

Skewness and variance were rampant in the raw data. More specifically the greatest number of standard deviations from the mean.

Skewness makes the distribution of data non-ideal for the inputs of a neural network the outliers need drawing in, and the neural network has needs to accommodate the skewed values.

The greatest number of standard deviations from the mean being large causes similar issues with this attribute usually coming with skewness and requiring accommodation from the neural network during processing.

Normalisation of the data solved these issues. The normalisation I did to certain headings involves:

- Applying  $\log_{10}$  to all values in the AREA field,
  - o This minimised skew and the largest SDs from mean very effectively as seen in Figure 2,
- Applying  $\ln$  (log natural) to all values in the FPEXT, LDP, SAAR, and Index flood fields,
- Applying the Box-Cox transformation with a lambda value of 11 to all values in the FARL field,
  - o  $(\text{value}^{11} - 1) / 11$ ,
  - o Normalisation was a lot more difficult due to an enormous negative skew which left a lot of shared values (e.g., 1.000 or 0.994). Shared values cannot be distributed as there is no way to mathematically spread them without randomness.
  - o Results are shown in Figure 3

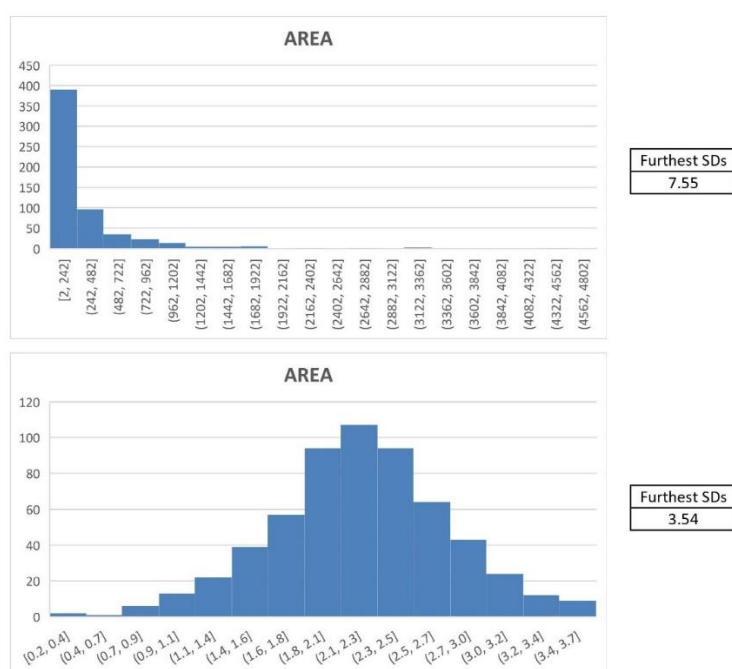


Figure 2 – AREA Changes

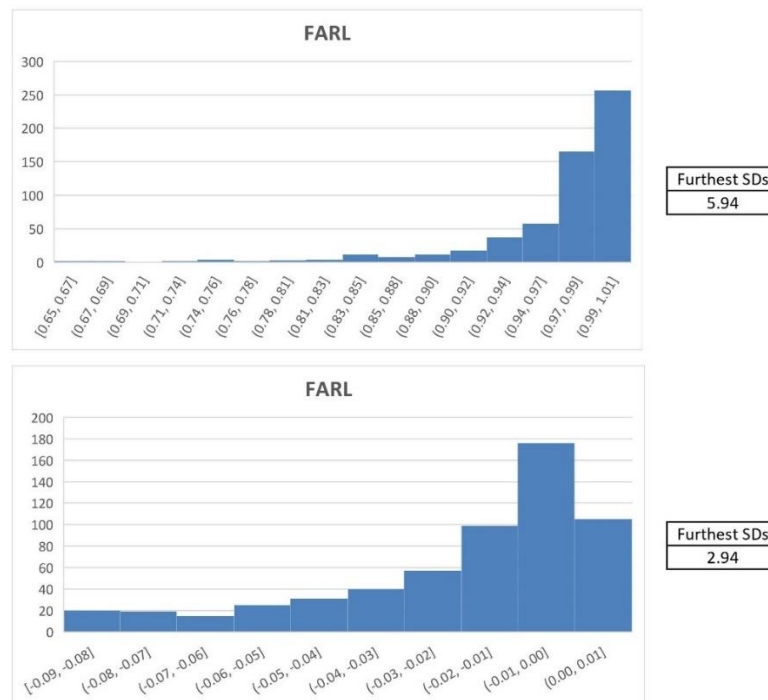


Figure 3 – FARL Change

## Removing Outliers

Initially, some of the largest number of standard deviations from the mean were around 7. This would usually immediately be discarded from datasets but after my research into the fields with the help of the “Flood Estimation Handbook 5 (Centre for Ecology & Hydrology, 2008)” I found that the values were all possible. This, in my opinion, did not make them fit the criteria of needing removal due to being “wrong” or “not useful”, which is what makes outliers redundant.

On top of all being possible, there were such many values with many standard deviations away from the mean that removing all of them would be detrimental to the size of the dataset.

Additionally, normalisation of the data reduced the largest number of standard deviations from the mean from 7 to 4.5, making the need for removal of outliers even smaller.

This aided the standardisation of data, also, mentioned later.

## Removing Irrelevant Predictors

An important concept in dataset cleaning is the removal of predictors that have so little effect on the predictand that they act as noise in the neural network. To find such predictors, I created a table comparing all combinations of pairs of predictors and the predictand. The metric for whether a field influences another that I used is the  $R^2$  Correlation Coefficient. The table is illustrated in Figure 4.

Based on the table, I plotted certain pairs of variables against each other to visualise their correlation (or lack thereof).

Moderate correlation, but nothing conclusive, is the correlation between “AREA” and “Flood Index”. This is strongest correlation between a predictor and a predictand and is visualised in Figure 5.

The three predictors that seem to have almost no correlation to “Flood Index”, the predictand, are “FARL”, “FPEXT”, and “RMED-1D”. These are also plotted in Figure 5 and clearly show no correlation.

With “RMED-1D”, on top of being uncorrelated to the predictand, there is a strong correlation with “SAAR”. This is shown in Figure 6. What this means is that “SAAR” could be used to represent both of itself and “RMED-1D”. This means that “RMED-1D” is a strong candidate for removal from the dataset.

In my testing of different improvements to the ANN, I tried not removing any predictors, removing “RMED-1D”, removing “FARL”, “FPEXT”, and “RMED-1D”, and removing “FARL”, “FPEXT”, “RMED-1D”, and “SAAR”. These were all due to low correlation to “Flood Index”.

	AREA	BFIHOST	FARL	FPEXT	LDP	PROPWET	RMED-1D	SAAR
AREA		0.0004	0.0051	0.0163	0.7894	0.0033	0.0072	0.0027
BFIHOST	0.0004		0.0116	0.0296	0.0002	0.2860	0.1089	0.1695
FARL	0.0051	0.0116		0.0018	0.0032	0.0943	0.1205	0.1541
FPEXT	0.0163	0.0296	0.0018		0.0268	0.1449	0.1737	0.1614
LDP	0.7894	0.0002	0.0032	0.0268		0.0029	0.0148	0.0086
PROPWET	0.0033	0.2860	0.0943	0.1449	0.0029		0.4028	0.6353
RMED-1D	0.0072	0.1089	0.1205	0.1737	0.0148	0.4028		0.8110
SAAR	0.0027	0.1695	0.1541	0.1614	0.0086	0.6353	0.8110	

Index flood	0.5760	0.0737	0.0049	0.0094	0.4813	0.1552	0.0327	0.0578
-------------	--------	--------	--------	--------	--------	--------	--------	--------

Figure 4 – R<sup>2</sup> Comparison Table

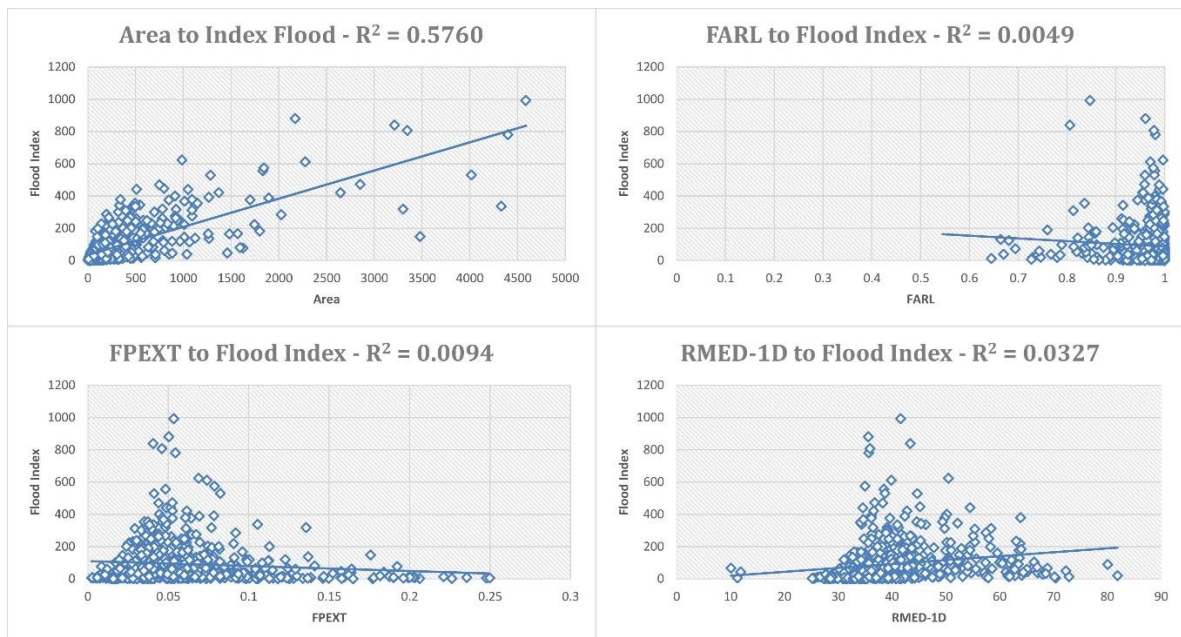


Figure 5 – General Deletion Justification

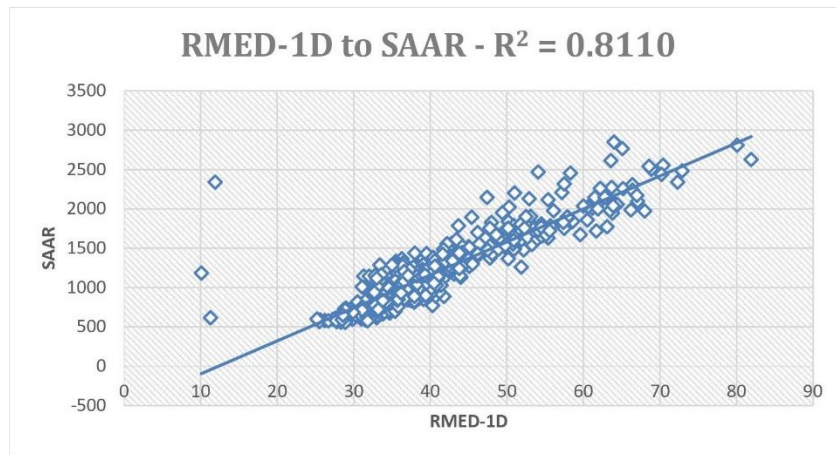


Figure 6 – Further Justification for RMED-1D

## Splitting Data Sets

Like what is standard for data splitting for neural networks, I put 60% of the data into the training data set, 20% into validating, and 20% into testing.

This could be randomly or not. In the code, I implemented non-random split use by inputting -1 as the index of the random split folder containing training, validation, and test data. Randomly split folders were indexed from 0.

To produce a non-random split, the first 60% were put into training data, next 20% into validating, and final 20% into testing. The same is true for a random split but the step of shuffling the records before separating.

## Standardising the Data

Once the data is split, all data (including the testing data) is standardised to the training and validation data sets. The standardisation I used, applied the formula:

$$\frac{value - min}{max - min} * 0.8 + 0.1$$

where:

- *min* is the minimum value for that field in the training and validation sets, and
- *max* is the maximum value for that field in the training and validation sets.

which puts all the values in the training and validation data sets in the range of [0.1, 0.9]. This allows testing data to have values slightly greater than 0.9 or less than 0.1 but still being in the range [0, 1].

## Final Format

The final format of the processed data was CSV in several .txt files.

To make the files more understandable, I created my own implementation of comments. Lines beginning with “#” would be ignored by the CSV reader allowing me to program comments between sections of CSV data.

This did not come in especially useful here but in future data storage, such as the storage of final weights and biases this was very useful.



## MSE vs. RMSE

Throughout this project I decided to measure error using mean square error (MSE) rather than root mean square error (RMSE) because MSE is measured in units that are the square of the target variable and so penalises larger errors more severely. This aligns with my goals of not just trying to minimise error but also minimise large errors amongst existing errors.

## Back Propagation Algorithm Training On ANN

### Representation of the ANN

The representation that I chose allows for multiple output nodes. It also allows for all passing functions (forwards and backwards) to be adapted more easily to more than one hidden layer due to all layers having the same representation.

The (one hidden layer) representation is:

- 2D array for weights from input nodes to hidden nodes,
- 2D array for weights from hidden nodes to output nodes.
- 1D array for biases from input nodes to hidden nodes,
- 1D array for biases from hidden nodes to output nodes.

### Forward Pass

A forward pass through a node to work out its output uses the equation:

activation function of current node(  
sum of (inputs \* weights from input node to current node) + bias from input node to current node  
)

I made the forward pass a function so that it could be used on any number of layers. This function gets ran from the input layer to the output layer in order.

### Matrix Use

As shown in the slides, I used matrix multiplication to do all of the calculations of each forward pass, immediately. Using NumPy, this was an easy to set up process. Each matrix is a 2D array which it was fast to convert 2D representations of the ANN into.

Because of the slightly different data storage methods in NumPy, using matrix multiplication has minor change to maths. This is noticeable at ~15<sup>th</sup> decimal place when multiplying the very small numbers that are in my ANN.

### Backwards Pass

A backwards pass for an output node is:

change to weights and biases of current node inputs = (correct output - output) \*  
first differential of activation function of current node(output of current node)

A backwards pass for any other type of node is:

change to weights and biases of current node inputs =  
sum of (weight from current node to next node \* change to weights and biases of next node) \*  
first differential of activation function of current node(output of current node)



The backwards pass works from the output layer to the input layer in order.

## Updating Weights and Biases

Once the changes to weights and biases of nodes have been calculated, the weights are updated using the following formula:

$$w_{i,j}^* = w_{i,j} + \rho \delta_j u_i$$

where:

- $w_{i,j}$  is the weight from the previous node to the current node,
- $w_{i,j}^*$  is the new weight from the previous node to the current node,
- $\rho$  is the learning rate,
- $\delta_j$  is the change to weights and biases of current node, and
- $u_i$  is the output from the previous node.

## Momentum

When implementing momentum, I noticed that:

$$w_{i,j}^* = w_{i,j} + \rho \delta_j u_i + \alpha \Delta w_{i,j}$$

$$\Delta w_{i,j} = w_{i,j} + \rho \delta_j u_i - w_{i,j} = \rho \delta_j u_i$$

$$w_{i,j}^* = w_{i,j} + \rho \delta_j u_i + \alpha \rho \delta_j u_i = w_{i,j} + (1 + \alpha) \rho \delta_j u_i$$

where:

- $w_{i,j}$  is the weight from the previous node to the current node,
- $w_{i,j}^*$  is the new weight from the previous node to the current node,
- $\rho$  is the learning rate,
- $\alpha$  is the momentum parameter,
- $\delta_j$  is the change to weights and biases of current node, and
- $u_i$  is the output from the previous node.

so in my implementation of momentum I used the more simple right hand side of the last equation.

Momentum is an optional modification that I test the improvement of performance with and without.

## Bond Driver

The way that I implemented bold driver is if the MSE of the actual vs predicted validation data “Index flood” values increases by a predetermined amount (0.04) after a certain number of epochs, reset the weights and biases and decrease the learning rate by 30%. If the MSE decreases by the predetermined amount, reset the weights and biases and increase the learning rate by 5%.

Bond driver is an optional modification that I test the improvement of performance with and without.

## Annealing

My annealing start parameter was the default learning rate (0.1) and end parameter was 0.01. As the epochs increased the learning rate approached the end parameter in the fashion shown in figure 7.

Annealing and bold driver cannot work simultaneously so I created a parameter called `eitherBoldDriverOrAnnealingInUse` which paired with the variable `boldDriverInUse` allows for avoiding

the possibility of both being enabled at once. An independent variable that determines whether annealing is in use does not exist.

Annealing is an optional modification that I test the improvement of performance with and without.

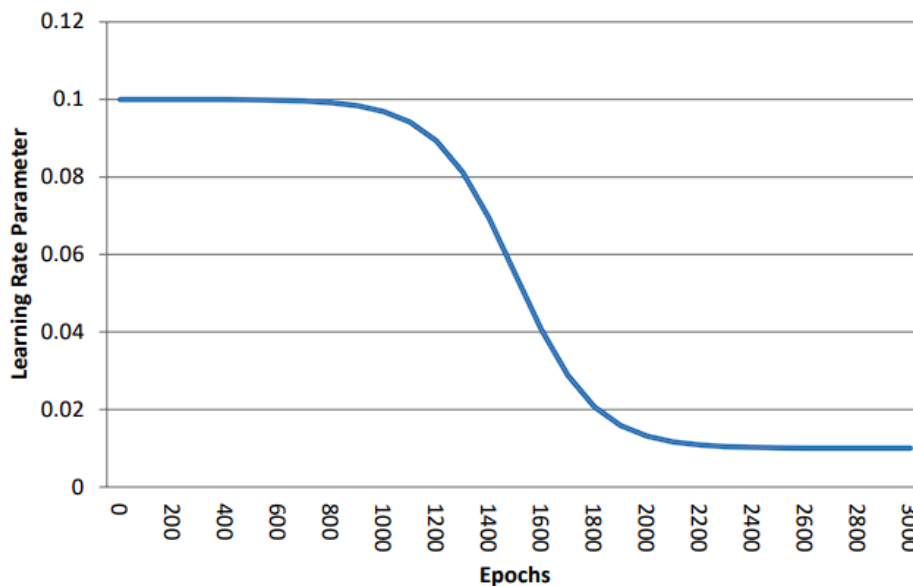


Figure 7 - Annealing Representation

## Weight Decay

Weight decay penalises larger weights by introducing a variable into the equation for change of output nodes so that the error from the prediction of correct final output is increased if the average weight is large, so that weights get slightly more decreased over many epochs.

Weight decay is an optional modification that I test the improvement of performance with and without.

## Batch Learning

Batch learning holds the weight and bias changes from many records at a time then updates the weights and biases once a portion of an epoch has been held.

Batch learning is an optional modification that I test the improvement of performance with and without.

## Activation Functions

The options for activation functions that I use in my program are sigmoid, tanh, and rectified linear unit (ReLU). These all use different equations to work out the output of each node and the gradient that the weights and biases need to consider when updating.

I experiment with different activation functions on different layers in my testing.

## Testing

The tests that I have mentioned several times in this report consist of making small modifications to the default program and seeing the change in MSE of actual vs predicted testing data “Index flood” values, number of epochs taken, and number of backward passes completed. I used the metric of “number of

backward passes completed” because “epochs taken” does not take the number of layers and number of nodes in each layer into account. These changes of results are saved along with the final weights and biases of the model.

Each small modification is tested with 3 different starting weights and biases and 4 different random split folder numbers of the data. This makes 12 unique tests per modification tweak.

## Running of Tests

Doing 12 unique tests per modification tweak would have taken my laptop a lot of time to do by itself so to run the tests I used the access I have to laboratory computers to my advantage.

I split all the testing into 12 different python files with a different combination of starting weights and biases and random split folder numbers (i.e., (0, 0), (0, 1), (0, 2), etc).

As shown in Figure 8, I ran each of these files in a different lab computer. This turned a roughly 15 hour job into a 90 minute job.

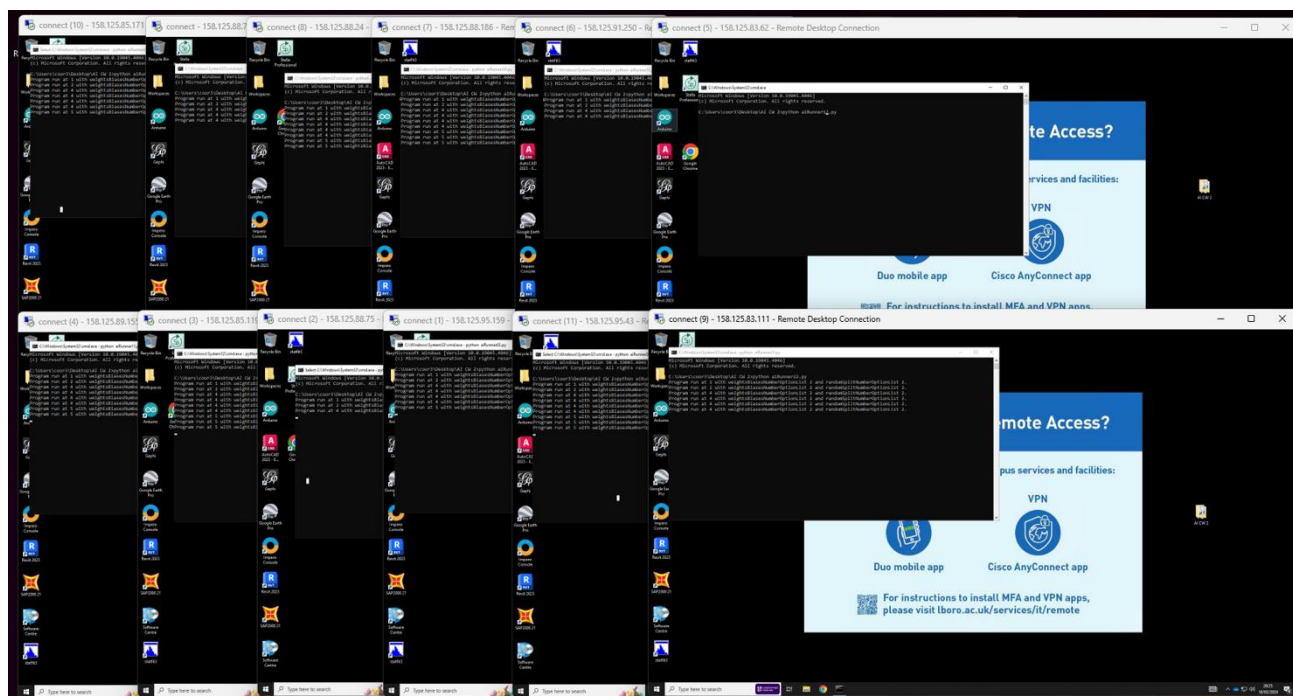


Figure 8 – Twelve Devices Running

## Results Storing

The results of testing how adding and removing modifications are stored in files with unique names. These file names are generated following the following rules:

“weightsBiases\_

\*binary number of with 1 representing that input is used and 0 representing that it is not (in order), i.e., 1111111 means all inputs are used, 1100000 means only AREA and BFIHOST are inputs\*,

\*number of hidden layers, i.e., 1\*,

(\*number of hidden nodes in hidden layer 1, i.e., 8\*, \*(if exists) number of hidden nodes in hidden layer n, i.e., 6\*),

(\*activation function for hidden layer 1, i.e., sigmoid or tanh\*, \*(if exists) activation function for hidden layer n, i.e., sigmoid or tanh, activation function for output layer n, i.e., sigmoid or tanh\*),

- \*number of output nodes, i.e., 1\*,
- \*initial learning rate\*,
- \*maximum number of epochs\*,
- \*frequency of validation checks\*,
- \*weight biases file number\*,
- \*random split folder number\*,
- \*momentum multiplier in use, i.e., 0 or 1\*,
- \*momentum multiplier value\*,
- \*either bold driver or annealing in use, i.e., 0 or 1\*,
- \*bold driver in use, i.e., 0 or 1\*,
- \*bold driver increase factor\*,
- \*bold driver decrease factor\*,
- \*frequency of bold driver checks\*,
- \*maximum error increase\*,
- \*maximum learning rate\*,
- \*minimum learning rate\*,
- \*annealing start parameter\*,
- \*annealing end parameter\*,
- \*weight decay in use, i.e., 0 or 1\*,
- \*batch learning in use, i.e., 0 or 1\*,
- \*number of batches split into\*

With this zero-conflict file naming system, it allowed for seamless collating of files once the 12 computers were done processing the tests.

If the variable called “storeExtraData” is True then, every 200 epochs, the progression of the MSE of the actual and predicted testing data is stored in a format that allows for copying and pasting into Excel to generate a table:

```
for dataPoint in meanErrorProgression:
    file.write(f"{dataPoint[0]}\t{dataPoint[1]}\n")
```

The file in question has a name very similar to the results of modifications.

“meanErrorProgression\_  
 \*binary number of with 1 representing that input is used and 0 representing that it is not (in order), i.e., 1111111 means all inputs are used, 1100000 means only AREA and BFIHOST are inputs\*,  
 ...”

The saved file can easily be pasted into a table like in Figure 9.

Default	
Epoch	Testing Data MSE
0	0.017970
200	0.001787
400	0.001703
600	0.001667
800	0.001640
1000	0.001616
1200	0.001595
1400	0.001576
1600	0.001559
1800	0.001544
2000	0.001530
2200	0.001518
2400	0.001508
2600	0.001499
2800	0.001492
3000	0.001485
3200	0.001480
3400	0.001476
3600	0.001472
3800	0.001469
4000	0.001466
4200	0.001464
4400	0.001463
4600	0.001462
4800	0.001461
5000	0.001460
5200	0.001460
5400	0.001460
5600	0.001460
5800	0.001461
6000	0.001462
6200	0.001463
6400	0.001464

Figure 9 – Generate MSE Progression

## Test Processing

After all the tests were complete, I ran a program that I created (dataProcessor.py) to go through all the tests and formats them in a way that can be copied and pasted in Excel to form tables as shown in Figures 10 and 11.

This program combines tests with the same modifications into one table which can be averaged during the comparing because results tend to vary depending on the starting weights and biases and the random split of the data.

The process is like the generation of Figure 9 but with more complexity. This complexity includes drawing data from more locations and combining the tables of each modification into one file ready to be copied and pasted.

The processed test data is used in comparing algorithms.

Title			Headings Used			
Value (Default)			['AREA', 'BFIHOST', 'FARL', 'FPEXT', 'LDP', 'PROPWET', 'RMED-1D', 'SAAR', 'Index flood']			
			Weights and Biases Start File Number			
			0	1	2	
MSE of Testing	Random Split File Number	0	0.00182227	0.001300306	0.001510343	
Epochs			2500	4000	5500	
Backwards Pass Calculations			11444576	18308576	25172576	
		1	0.001369743	0.001815761	0.001247181	
			10001	3000	6500	
			45764576	13732576	29748576	
		2	0.001436627	0.001395159	0.001840207	
			6000	10001	2000	
			27460576	45764576	9156576	
		3	0.001258531	0.001387455	0.001372552	
			3000	10001	10001	
			13732576	45764576	45764576	
Value				['AREA', 'BFIHOST', 'FARL', 'FPEXT', 'LDP', 'PROPWET', 'SAAR', 'Index flood']		
				Weights and Biases Start File Number		
				0	1	2
MSE of Testing	Random Split File Number	0	0.001819716	0.001244434	0.001408942	
Epochs			2500	4000	4000	
Backwards Pass Calculations			11444576	18308576	18308576	
		1	0.001387501	0.00183396	0.001190733	
			6500	3000	10001	
			29748576	13732576	45764576	
		2	0.001371963	0.001398303	0.001818649	
			5000	10001	4000	
			22884576	45764576	18308576	

Figure 10 – Example Result Tables

Title		Headings Used				Number of Hidden Layers			Number of Hidden Nodes in Each Layer				Activation Functions		
Value (Default)		['AREA', 'BFIHOST', 'FARL', 'FPEXT', 'LDP', 'PROPWET', 'RMED-1D', 'SAAR', 'Index flood']				1			[12]				['sigmoid', 'sigmoid']		
		Weights and Biases Start File Number				Weights and Biases Start File Number			Weights and Biases Start File Number				Weights and Biases Start File Number		
		0	1	2		0	1	2	0	1	2	0	1	2	
MSE of Testing	Random Split File Number	0.0018223	0.001300306	0.0015103		0.0018223	0.0013003	0.0015103	0.0018223	0.0013003	0.0015103	0.0018223	0.0013003	0.0015103	
Epochs		2500	4000	5500		2500	4000	5500	2500	4000	5500	2500	4000	5500	
Backwards Pass Calculations		11444576	18308576	25172576		1144576	18308576	25172576	1144576	18308576	25172576	1144576	18308576	25172576	
		0.0013697	0.001815761	0.0012472		0.0013697	0.001815761	0.0012472	0.0013697	0.001815761	0.0012472	0.0013697	0.001815761	0.0012472	
		10001	3000	6500		10001	3000	6500	10001	3000	6500	10001	3000	6500	
		45764576	13732576	29748576		45764576	13732576	29748576	45764576	13732576	29748576	45764576	13732576	29748576	
		0.0014366	0.001395159	0.0018402		0.0014366	0.001395159	0.0018402	0.0014366	0.001395159	0.0018402	0.0014366	0.001395159	0.0018402	
		6000	10001	2000		6000	10001	2000	6000	10001	2000	6000	10001	2000	
		27460576	45764576	9156576		27460576	45764576	9156576	27460576	45764576	9156576	27460576	45764576	9156576	
		0.0012585	0.001387455	0.0013726		0.0012585	0.001387455	0.0013726	0.0012585	0.001387455	0.0013726	0.0012585	0.001387455	0.0013726	
		3000	10001	10001		3000	10001	10001	3000	10001	10001	3000	10001	10001	
		13732576	45764576	45764576		13732576	45764576	45764576	13732576	45764576	45764576	13732576	45764576	45764576	
Value		['AREA', 'BFIHOST', 'FARL', 'FPEXT', 'LDP', 'PROPWET', 'SAAR', 'Index flood']							[8]				['tanh', 'tanh']		
		Weights and Biases Start File Number							Weights and Biases Start File Number				Weights and Biases Start File Number		
		0	1	2		0	1	2	0	1	2	0	1	2	
MSE of Testing	Random Split File Number	0.0018197	0.00124434	0.0014089		0.0018457	0.0012363	0.0013664	0.0018457	0.0012363	0.0013664	0.0018457	0.0012363	0.0013664	
Epochs		2500	4000	4000		1267576	1031168	3163168	1267576	1031168	3163168	1267576	1031168	3163168	
Backwards Pass Calculations		11444576	18308576	18308576		0.0013702	0.0018067	0.0012878	0.0013702	0.0018067	0.0012878	0.0013702	0.0018067	0.0012878	
		0.0013875	0.00183396	0.0011907		3163168	6333168	9501168	3163168	6333168	9501168	3163168	6333168	9501168	
		6500	3000	10001		0.0014238	0.0013704	0.0018072	0.0014238	0.0013704	0.0018072	0.0014238	0.0013704	0.0018072	
		29748576	13732576	45764576		10000	10001	4000	10000	10001	4000	10000	10001	4000	
		0.001372	0.001398303	0.0018186		3163168	3163168	6333168	3163168	3163168	6333168	3163168	3163168	6333168	
		5000	10001	4000		0.0012601	0.0014252	0.0013923	0.0012601	0.0014252	0.0013923	0.0012601	0.0014252	0.0013923	
		22884576	45764576	18308576		1031168	1031168	3163168	1031168	1031168	3163168	1031168	1031168	3163168	
		0.0012034	0.00135326	0.0015792											
		8500	10001	5000											
		38300576	34324576	43476576											
Value		['AREA', 'BFIHOST', 'LDP', 'PROPWET', 'SAAR', 'Index flood']							[10]				['relu', 'relu']		
		Weights and Biases Start File Number							Weights and Biases Start File Number				Weights and Biases Start File Number		
		0	1	2		0	1	2	0	1	2	0	1	2	
MSE of Testing	Random Split File Number	0.0019452	0.001358183	0.0015863		0.0018561	0.0012368	0.0014321	0.0018561	0.0012368	0.0014321	0.0018561	0.0012368	0.0014321	
Epochs		6000	1000	10001		1355872	11239872	17427872	1355872	11239872	17427872	1355872	11239872	17427872	
Backwards Pass Calculations		27460576	4580576	45764576		0.0013396	0.0018723	0.0012534	0.0013396	0.0018723	0.0012534	0.0013396	0.0018723	0.0012534	
		0.0013586	0.001383552	0.001408		10001	2500	3000	10001	2500	3000	10001	2500	3000	
		10000	3000	10001		38723872	3683872	15555872	38723872	3683872	15555872	38723872	3683872	15555872	
		45764576	13732576	45764576		0.0014831	0.001342	0.0018182	0.0014831	0.001342	0.0018182	0.0014831	0.001342	0.0018182	
		0.001836	0.001517321	0.0013785		3500	10001	3000	3500	10001	3000	3500	10001	3000	
		3500	10001	2500		13555872	38723872	11619872	13555872	38723872	11619872	13555872	38723872	11619872	
		16020576	45764576	11444576		0.0012578	0.0014702	0.0014244	0.0012578	0.0014702	0.0014244	0.0012578	0.0014702	0.0014244	
		0.001431	0.00151004	0.0015135		15439172	17427872	15363872	15439172	17427872	15363872	15439172	17427872	15363872	
		10001	10001	10001											
		45764576	45764576	45764576											
Value		['AREA', 'BFIHOST', 'LDP', 'PROPWET', 'Index flood']							[14]				['sigmoid', 'tanh']		
		Weights and Biases Start File Number							Weights and Biases Start File Number				Weights and Biases Start File Number		
		0	1	2		0	1	2	0	1	2	0	1	2	
MSE of Testing	Random Split File Number	0.0023044	0.002354115	0.0021935		0.0018622	0.0012717	0.0015614	0.0018653	0.001313	0.0016018	0.0018653	0.001313	0.0016018	
Epochs		2000	4500	5000		29045280	20565280	18485280	29045280	20565280	18485280	29045280	20565280	18485280	
Backwards Pass Calculations		3155576	2053676	22884576		0.0014157	0.0014038	0.0018257	0.0014157	0.0014038	0.0018257	0.0014157	0.0014038	0.0018257	
		0.0020732	0.002352776	0.0020101		3000	10001	3000	3000	10001	3000	3000	10001	3000	
		10001	1000	2500		47525280	52805280	15845280	47525280	52805280	15845280	47525280	52805280	15845280	
		45764576	4580576	11444576		0.0018413	0.0016693	0.0014034	0.0018413	0.0016693	0.0014034	0.0018413	0.0016693	0.0014034	
		0.0022063	0.002043325	0.0020216		4000	10001	5000	4000	10001	5000	4000	10001	5000	
		3500	10001	2000		2125280	2125280	29045280	2125280	2125280	29045280	2125280	2125280	29045280	
		16020576	45764576	3155576											
		0.0022906	0.00209037	0.0020985											
		10001	10001	4000											
		45764576	45764576	18308576											

Figure 11 – Zoomed Out Results Table View

# Comparing Algorithms

## Different Modifications

From the information displayed in Figure 10 and Figure 11 the results of the tests with the same modifications can be compared. To do this I used a new sheet and got rid on the parameters which only has one option during testing to create the table in Figure 12 and Figure 13.

In this table, using conditional formatting I set up that if a value of the result is greater than the default result value, then the cell gets coloured green, else red. For the parameters with values better than the default result, I highlighted them yellow.

Headings Used	Number of Hidden Nodes in Each Layer
['AREA', 'BFIHOST', 'FARL', 'FPEXT', 'LDP', 'PROPWET', 'RMED-1D', 'SAAR', 'Index flood']	[12]
0.001479678	0.001479678
6042	6042
27651242.67	27651242.67
['AREA', 'BFIHOST', 'FARL', 'FPEXT', 'LDP', 'PROPWET', 'SAAR', 'Index flood']	[8]
0.001451341	0.001473229
6208.5	5958.666667
28413909.33	18879168
['AREA', 'BFIHOST', 'LDP', 'PROPWET', 'SAAR', 'Index flood']	[10]
0.001700349	0.001489887
7167.166667	4958.5
32799242.67	19202538.67
['AREA', 'BFIHOST', 'LDP', 'PROPWET', 'Index flood']	[14]
0.002219874	0.001499379
5375.333333	4791.75
24600576	25305280
	[16]
	0.001483721
	5125.166667
	30673984

Figure 12 – Average Comparison Table

Headings Used	Number of Hidden Nodes in Each Layer	Activation Functions	Momentum Multiplier In Use	Either Bold Driver or Annealing In Use	Bold Driver In Use	Weight Decay In Use	Batch Learning In Use	Number of Batches Split Into
['AREA', 'BFIHOST', 'FARL', 'FPEXT', 'LDP', 'PROPWET', 'RMED-1D', 'SAAR', 'Index flood']	[12]	['sigmoid', 'sigmoid']	FALSE	FALSE	FALSE	FALSE	FALSE	20
0.001479678	0.001479678	0.001479678	0.001479678	0.001479678	0.001479678	0.001479678	0.001479678	0.001686839
6042	6042	6042	6042	6042	6042	6042	6042	6584.166667
27651242.67	27651242.67	27651242.67	27651242.67	27651242.67	27651242.67	27651242.67	27651242.67	30261909.33
['AREA', 'BFIHOST', 'FARL', 'FPEXT', 'LDP', 'PROPWET', 'SAAR', 'Index flood']	[8]	['tanh', 'tanh']	TRUE	TRUE	TRUE	TRUE	TRUE	1
0.001451341	0.001473229	0.001158454	0.001472217	0.001488	0.001476536	0.001486998	0.001686839	0.005504368
6308.5	5958.666667	8125.163333	5125.333333	5625.25	5000.25	6917	8584.166667	10001
28413909.33	18879168	37884576	23486876	25744576	22884576	31689242.67	36581909.33	45764576
['AREA', 'BFIHOST', 'LDP', 'PROPWET', 'SAAR', 'Index flood']	[10]	['relu', 'relu']						
0.001700349	0.001489887	0.38652275						
7167.166667	4958.5	10001						
32799242.67	19202538.67	40764576						
['AREA', 'BFIHOST', 'LDP', 'PROPWET', 'Index flood']	[14]	['sigmoid', 'tanh']						
0.002219874	0.001499379	0.001012714						
5375.333333	4791.75	3875						
24600576	25305280	17736576						
	[16]	['tanh', 'relu']						
	0.001483721	0.38652275						
	5125.166667	8084.163333						
	30673984	41569609.33						
		['relu', 'sigmoid']						
		0.001221985						
		2416.866667						
		11963242.67						
		['tanh', 'sigmoid']						
		0.00107736						
		7500.5						
		94524576						

Figure 13 – Zoomed Out Results Comparing View



## Optimal Modifications

Based on the data in Figure 13, I produced the ANN with the optimal modifications for this data set. The non-default parameters for this were:

- Removing only “RMED-1D” from the predictor list,
- Having 8 nodes in the hidden layer,
- Using tanh() as the activation function for the hidden layer and sigmoid() for the output,
- Enabling momentum,

Annealing, weight decay, and batch learning all ended up hindering my algorithm as Figure 13 shows in the right-hand-most three columns.

In Figure 14 and Figure 15, the results of the optimal modifications side-by-side with the default results show how it greatly outperforms.

Also, the side-by-side progression of the MSE of the actual and predicted testing data, in Figure 16, does not reveal a lot about the algorithms but the MSE values cement that the ANN with optimal modifications is better immediately.

Title			Presets		
Value (Default)			Default		
			Weights and Biases Start File Number		
			0	1	2
MSE of Testing	Random Split File Number	0	0.00182227	0.001300306	0.001510343
Epochs			2500	4000	5500
Backwards Pass Calculations			11444576	18308576	25172576
		1	0.001369743	0.001815761	0.001247181
			10001	3000	6500
		2	45764576	13732576	29748576
			0.001436627	0.001395159	0.001840207
		3	6000	10001	2000
			27460576	45764576	9156576
		3	0.001258531	0.001387455	0.001372552
			3000	10001	10001
			13732576	45764576	45764576
Value			Optimal		
			Weights and Biases Start File Number		
			0	1	2
MSE of Testing	Random Split File Number	0	0.001360627	0.000842794	0.000910089
Epochs			10001	3500	7500
Backwards Pass Calculations			31683168	11091168	23763168
		1	0.001026807	0.001396797	0.000874443
			6000	10001	2500
		2	19011168	31683168	7923168
			0.00081657	0.001064101	0.001521204
		3	10001	6500	10001
			31683168	20595168	31683168
		3	0.00091	0.00096446	0.001084899
			2000	3500	10001
			6339168	11091168	31683168

Figure 14 – Default vs. Optimal Modifications

Presets
Default
0.001479678
6042
27651242.67
Optimal
0.001064399
6792.083333
21519168

Figure 15 - Default vs. Optimal Averages Comparison

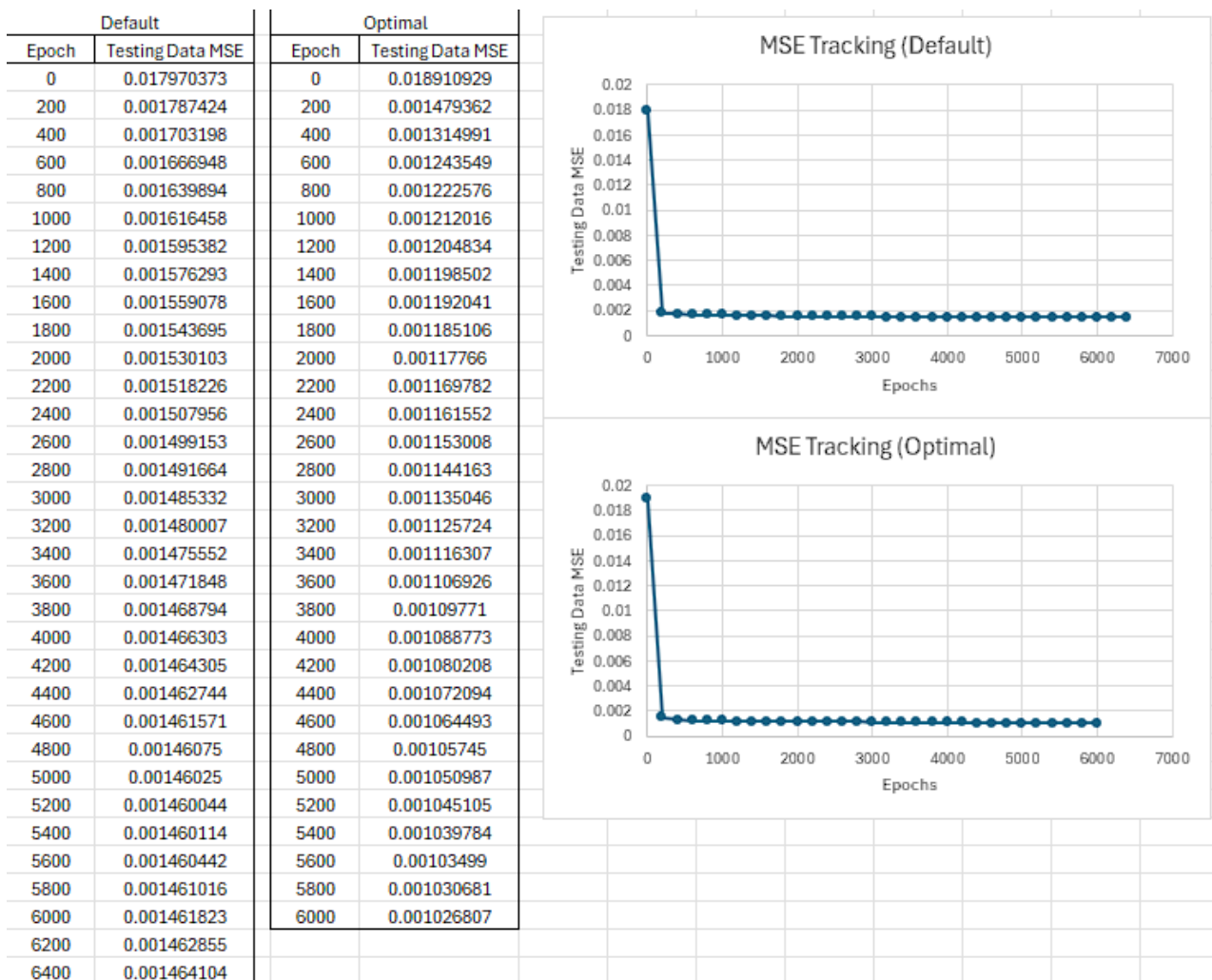


Figure 16 - Default vs. Optimal MSE Progression

## LINEST vs. ANN

In Excel, I used the LINEST() function to create the coefficients for a 9 dimensional regression line with an intersect. Using this regression line formula which will look something like:

$$IndexFlood = a_0x_0 + a_1x_1 + a_2x_2 + a_3x_3 + a_4x_4 + a_5x_5 + a_6x_6 + a_7x_7 + c$$

the regression lines prediction of the flood index can be generated.

The “train” the LINEST I used the first 80% of the data and then tested the regression line on the last 20% of the data.

My optimal modification ANN performed a lot better than the Excel regression line predictions.

The MSE is drastically different to usual MSEs because I unstandardised and unnormalised the data.

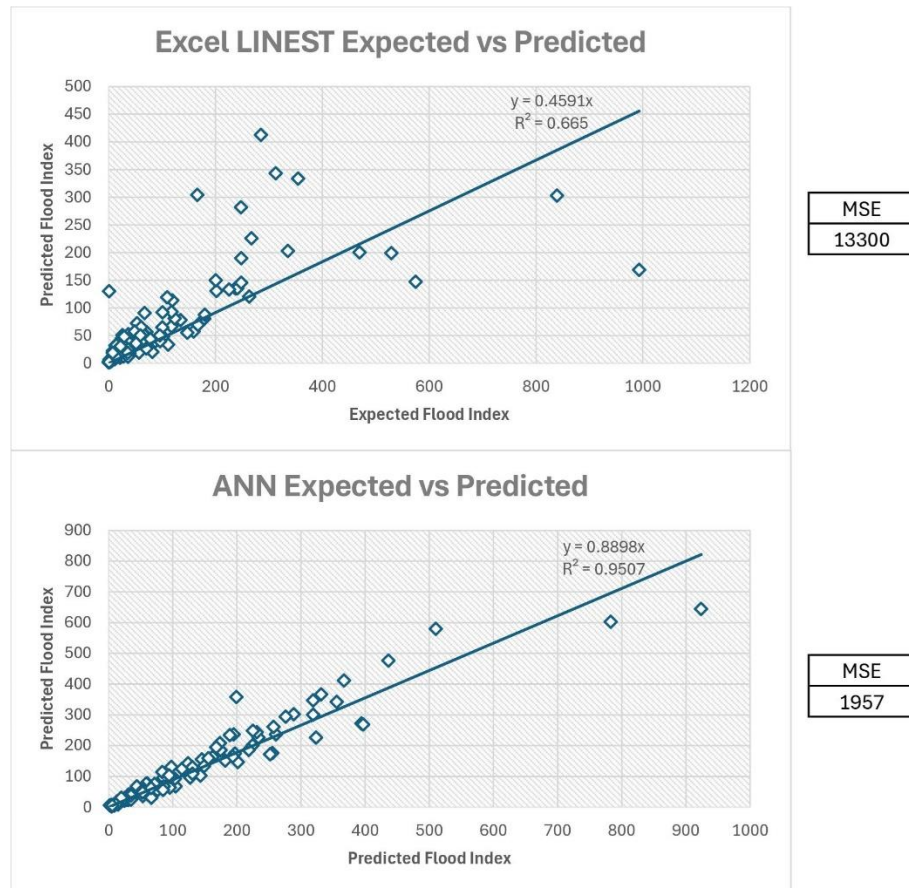


Figure 17 - LINEST vs. ANN

## Program Time Optimising

### My Approach

Using the code:

```
if __name__ == "__main__":  
    firstTime = time.time()  
    main(maxEpochs=200)  
    print(f"Time taken: {time.time() - firstTime}")
```

and:

```
# Print the amount of time passes since the last time current time was set  
print(f"time {time.time() - currentTime}")  
currentTime = time.time()
```

I was able to see how small tweaks to my code changed the run time.

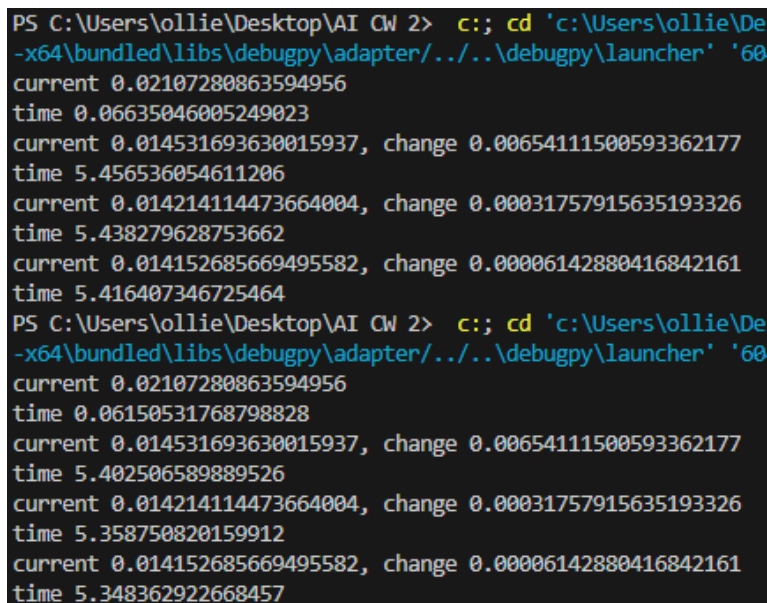
## Removing Small Repeated Variable Declarations

One very slight inefficiency I found was with the following lines of code:

```
# Calculate the changes in the output nodes
changesFromHiddenToOutputs = [0 for _ in range(numOfOutputNodes)]
for i in range(numOfOutputNodes):
    if activationFunctions[numOfHiddenLayers - 1] == "sigmoid":
        changesFromHiddenToOutputs[i] = ...
# Calculate the change in the hidden nodes
changesFromInputToHidden = [0 for _ in range(numOfHiddenNodes)]
for i in range(numOfHiddenNodes):
    if activationFunctions[0] == "sigmoid":
        changesFromInputToHidden[i] = sum(...
```

Move the second lines to before the overarching while loop because although they look like relatively small variable declarations, if the program runs for 200 epochs with 200 records in the training data, they will be run 40,000 time which adds up.

As shown in Figure 18, the change this made was not very big but with the testing that I did this ended up saving roughly 30 minutes.



```
PS C:\Users\ollie\Desktop\AI CW 2> c::; cd 'c:\Users\ollie\Desktop\AI CW 2\debugpy\launcher' '60'
current 0.02107280863594956
time 0.06635046005249023
current 0.014531693630015937, change 0.00654111500593362177
time 5.456536054611206
current 0.014214114473664004, change 0.00031757915635193326
time 5.438279628753662
current 0.014152685669495582, change 0.00006142880416842161
time 5.416407346725464
PS C:\Users\ollie\Desktop\AI CW 2> c::; cd 'c:\Users\ollie\Desktop\AI CW 2\debugpy\launcher' '60'
current 0.02107280863594956
time 0.06150531768798828
current 0.014531693630015937, change 0.00654111500593362177
time 5.402506589889526
current 0.014214114473664004, change 0.00031757915635193326
time 5.358750820159912
current 0.014152685669495582, change 0.00006142880416842161
time 5.348362922668457
```

Figure 18 - Variable Repetition Removal Time Change

## Not Using Some Functionality Towards Forwards Passing

There are a few features that I changed to optimise the program to do with the forwards pass:

- Removal of the forwards pass function,
- Removal of the matrix use,
- Removal of the small repeated variable declarations.

The same way removing small repeated variable declarations did, as seen in Figure 19, the change this made was not very big but with the testing that I did this ended up saving roughly 2 hours.

```

PS C:\Users\ollie\Desktop\AI CW 2> c:; cd 'c:\Users\ollie\Desktop\AI CW 2\debugpy\adapter\..\..\debugpy\launcher' '49'
current 0.024426914112471516
time 0.06926083564758301
current 0.0026073937557298315, change 0.02181952035674168330
time 6.015635251998901
current 0.0024852964618181698, change 0.00012209729391166177
time 5.945523738861084
current 0.002452430067329679, change 0.00003286639448849080
time 6.201164960861206
PS C:\Users\ollie\Desktop\AI CW 2> c:; cd 'c:\Users\ollie\Desktop\AI CW 2\debugpy\adapter\..\..\debugpy\launcher' '49'
current 0.024426914112471516
time 0.0641572475433496
current 0.0026073937557298315, change 0.02181952035674168330
time 6.053376913070679
current 0.0024852964618181706, change 0.00012209729391166090
time 5.792435884475708
current 0.0024524300673296807, change 0.00003286639448848994
time 5.812279939651489
PS C:\Users\ollie\Desktop\AI CW 2>

```

Figure 19 - Forward Pass Editing Time Change

## Forwards Pass Function

Taking the computations out of the forward pass function saved computation time. Assumably, this is because variables do not need to be copied to a stack of parameters in the function. This computation would usually take no time but it takes more time when ran thousands of times per epoch.

## Matrix Use and Small Repeated Variable Declarations

The concept of repeated variable declarations taking a lot of time comes up again when declaring matrices. Any time saved with the efficiency of multiplying matrices with NumPy is negated by the amount of time spent setting up the matrices. Removing matrix use altogether solves this issue.

## References

Centre for Ecology & Hydrology. (2008). Flood Estimation Handbook 5. Retrieved from [https://www.ceh.ac.uk/sites/default/files/2021-11/Flood-Estimation Handbook-5-Catchment-Descriptors\\_Adrian-Bayliss.pdf](https://www.ceh.ac.uk/sites/default/files/2021-11/Flood-Estimation%20Handbook-5-Catchment-Descriptors_Adrian-Bayliss.pdf)