### Checklist

• I will also be looking at the code to see that is well structured, well commented, and decent variable names have been used.

### Checklist for myself

1. Data pre-processing (including cleansing and data splitting, identifying predictors) – 15%.

2. Implementation of the MLP algorithm (including modifications / improvements) – 35%.

3. Training and network selection – 20%.

4. Evaluation of final model (including comparisons between different modifications to the algorithm) – 20%.

5. Comparison with another data driven model – 10%.

### Using python

I decided to use python as the programming language of choice, this is because I have practice with using the language and it provides good libraries to work with. I will be using a few external libraries in my coursework. Firstly, I will be using **Pandas** for data handling and manipulation and **NumPy** for support for arrays and matrices and **Matplotlib** for plotting the data into graphs. After reading in the data with Pandas, the data will mainly be handled using NumPy arrays or python lists.

I aim to code the neural network using three separate classes: **Node**, **Layer** and **Network**. The goal of the **Node** class will be to handle the main computation, including forward propagation, backwards propagation, and weight updates. It will store the weights, last inputs and have its own delta value. The **Layer** class will act as a connection between the layers so that every previous layer is connected to the next and we can control the nodes by controlling the layers first, which makes it simpler to think about. The **Network** class will handle the main MLP training algorithm and adding layers.

I took an Object-Oriented approach to the development of the neural network so that nothing was hard-coded, and the user can easily change the structure and parameters of the network at any point.

### Data pre-processing

The first step I took was to clean the data using excel, removing any obvious errors. After my first review I found the following errors:

* There were a few values
* There were a few ‘a’ values
* There was a “#” value in row 784
* There were a few outlying rows (for example, there was a value of 5000 for Arkengart when the closest value to that was 225, and in Malham the highest was 9000 and the closest to that was) so I deleted these rows to prevent it from skewing the distribution of the data by too much.

I then and deleted each of these rows manually using Excel. I then standardized the data set using the formula:

Text

Description automatically generated

To get the data within a range of . I then checked the data for any correlations between features and I found that Crakehill, Westwick and Skip Bridge (gauging stations) all had a linear correlation with Skelton,

GRAPHS HERE

Whereas the rainfall stations had no such correlation, rather it was more scattered as can be seen below

GRAPHS HERE

Which isn’t necessarily useless, but a linear trend is much more feasible to predict. Therefore, I chose the gauging stations as my input features; the aim of my neural network prediction was to predict this linear trend for new data.

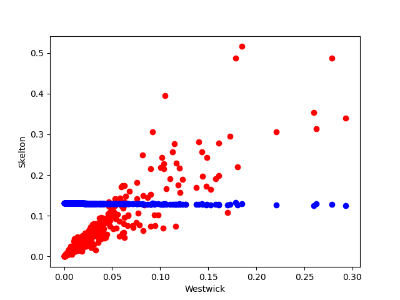
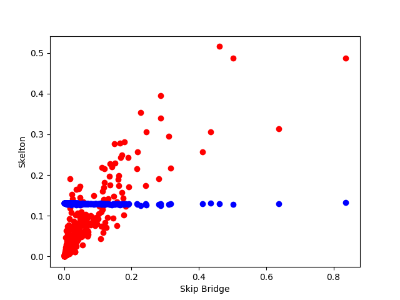
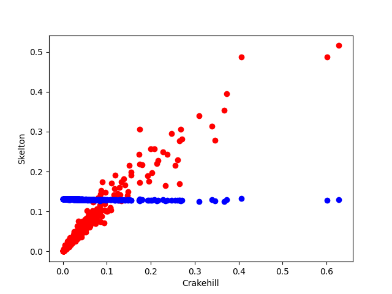
### Implementing the MLP Learning Algorithm

After first implementation all of the output values were converging to the same/similar value each time for different inputs. For example, after training the neural network with 6 hidden nodes and 1 output node

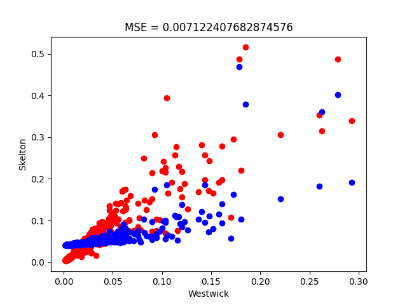
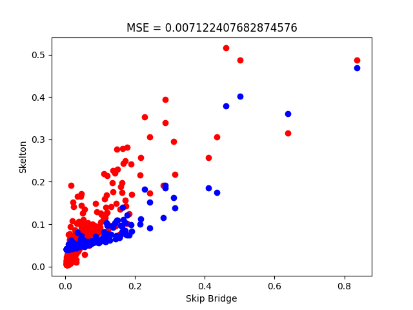
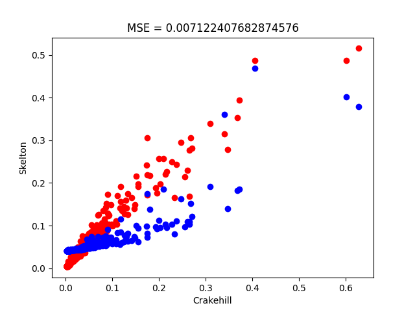
Diagram

Description automatically generated

I got the following results (red is the expected linear trend we want, and blue is the trend we have predicted):

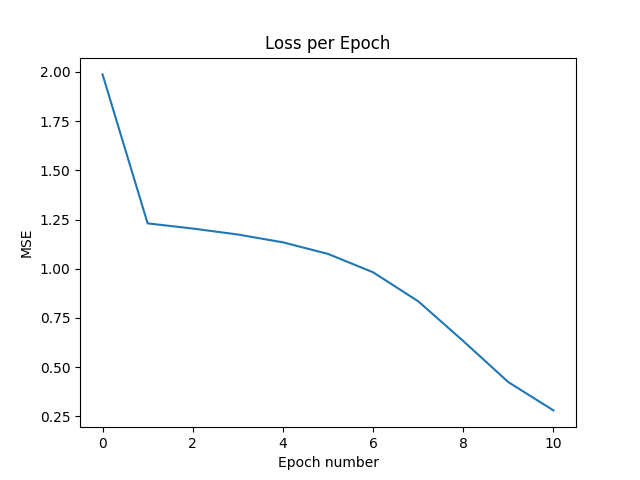


After playing around with the values of the learning algorithm, I eventually noticed that the algorithm was changing the weights very minimally each update, so I tried increasing the learning rate from 0.1 to 0.5 to increase the size in the change of the weights, which improved the model’s accuracy drastically:

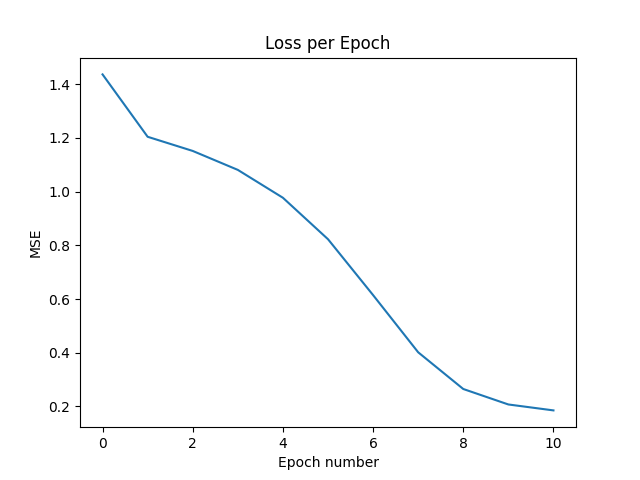


#### Changing the learning rate

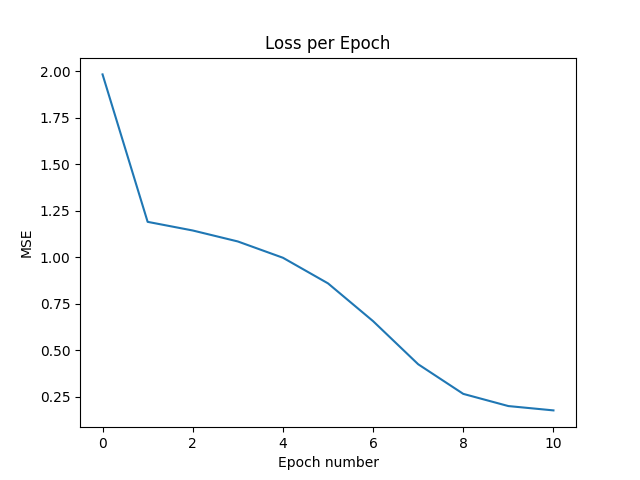
From here I trialled different learning rates, with 11 epochs, to find the best outcome for our given problem:



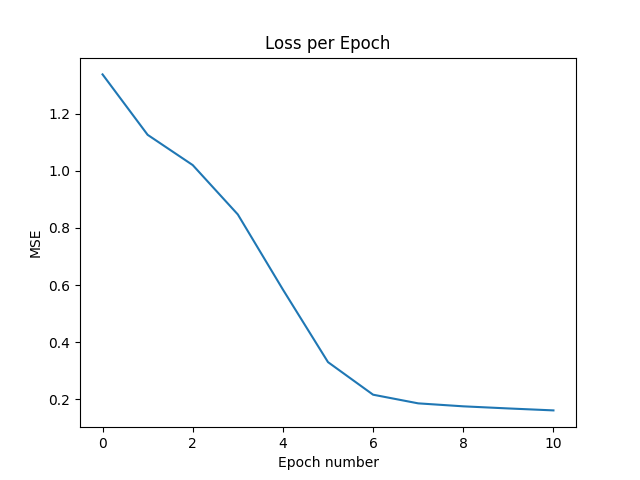
Learning rate = 0.6



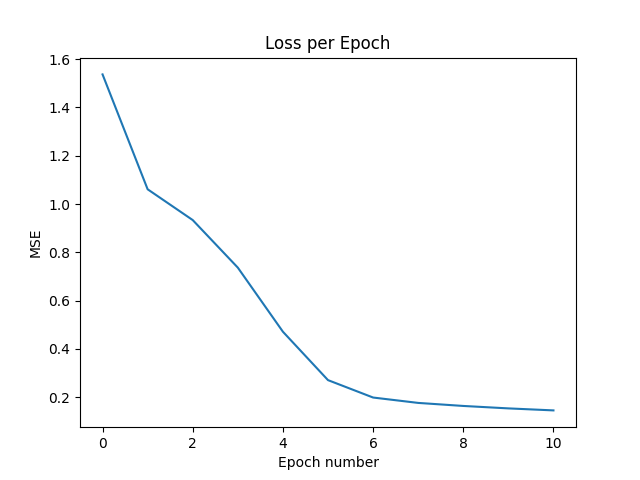
Learning rate = 0.7



Learning rate = 0.8



Learning rate = 0.9



Learning rate = 1.0

Clearly this network benefits from having a large learning rate. From a high-level glance it looks like the most optimal learning rate is somewhere between 0.7 and 0.9, but later I will implement Bold Driver to set the learning parameter automatically.

#### Testing different Structures

I then tested different structures for the model to see if I could get any improvement there:

Diagram

Description automatically generated

Firstly, I tried adding an extra layer of 6 hidden nodes. After the initial test with 0.7 as the learning rate, I got similar results that what I was getting at first implementation where the results of the predictions were all one value again, so I tried testing different learning rates, but this time I found no improvement in accuracy.

Chart

Description automatically generated

2 hidden layers, Learning rate = 0.7

Chart

Description automatically generated

2 hidden layers, Learning rate = 0.1

### Adding Momentum

Moving forward with our 3-layer network, after finding a model that predicts on our data with a fair amount of accuracy, I started to make improvements, firstly I attempted to add momentum to the weight updates in order to reach the minimum faster. This required adding the following code:

Text

Description automatically generated

In line with the formula

A picture containing clock, watch

Description automatically generated

After the first epoch we can observe the following changes to the weights:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | with momentum |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

As we can see that is a slight improvement in terms of the efficiency of convergence

### Bold driver

### Annealing

Unable to implement

### Weight decay

Unable to implement

### Batch Learning

### Trying different activation functions

#### TanH

#### Relu

#### Leaky Relu

### Trying different training algorithms

#### Line Searching

#### Conjugate Gradients

#### Newtons method

#### Levenberg Marquardt