**COB107 – ANN Coursework Documentation**

**Student: Russell de Roeper**

**F027497**

*Note: All figures written in bold italic font are displayed in the references section.*

**Data pre-processing (15%):**

**Removing and Replacing spurious data:**

To clean the data, I first began highlighting any values with the wrong data type. For example, in ***figure 1***we have a letter “a” within the Crakehill mean daily flow column, which should only accept positive real numbers. I found about 4 other wrong data types including a hashtag.

Next, I highlighted obvious outliers such as negative numbers within the mean daily flow columns as shown in ***figure 2***. There was even a value of 80,000 in the dataset that I highlighted as well in ***figure 4****.* For finding other outliers as explained in the next paragraph, I had to remove these values so that it won’t disturb my calculations for finding the upper and lower outlier boundaries.

*Some examples of outliers from the raw dataset.*

I have set my outliers to be minus-plus three times the standard deviations away from the mean. I chose this calculation as the lecture slides recommended using either 2 or 3. Furthermore, looking over each column there are frequent high values during the winter months that make sense to include. Therefore if I were to reduce the outlier upper boundary to be 2 standard deviations away from the mean I will be cutting out a lot of data points. To calculate the standard deviation and mean to find the other outliers, I processed each column in excel and used AVERAGE() and STDEV() to find the values shown in ***figure 5***. By removing the outliers, I have consequently created empty data points. To fill these points in I used linear interpolation between the top and bottom cells. I have also used averages of ***figure 5*** and adjusted them accordingly with estimation with the Skelton column.

*Two examples of removing outliers. It can be compared with the other two graphs by their correct years. For instance, this Arkengarthdale graph matches the time of the last positive outlier.*

**Data Splitting:**

I then split the dataset into three sections: training, validation, and test set. The original dataset contains 4 years’ worth of “mean daily flow” and “daily rainfall total” data which are all organized by a common date. I split that dataset by year groups with the training set having two years worth of data and the other two with one years’ worth of data. I find this arrangement suitable for two reasons. First, the training set must have more data points than the other two as in theory the ANN will have more data points to create a better model from. Secondly, each dataset has an even number of seasons which I think would benefit the neural network during training as it might find relationships between months, daily mean flow, and rainfall. The first graph above is found in the training set, whilst the second is found in my validation set.

**Standardization:**

I have also used MIN() and MAX() functions between the validation and training sets for use in the standardization. For translating each data point to a standardized format I used **Si =** **0.8\*(Ri – Min)/(Max-Min)+0.1** which gives a range between [0.1,0.9]. I did this to allow the predictand values to be within the range of the sigmoid and tanh activation functions range. So we can compare the standardized input to the standardized output data correctly.

**Correlations:**

I began exploring correlations between all the columns with the Skelton column. To do this I used the inbuilt excel command correl(). I figured out that the Crakehill, Skip Bridge and Westwick columns have an improved correlation to Skelton after being lagged by one day. After realizing this I deleted the first and newly generated last row from the dataset as those two rows have missing data points due to lagging. Similarly, I did the same with the rainfall data and again a day lag outputs the best results.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | *Figure rainfall correlation to Skelton* | | |  |  |
|  | t | t-1 | t-2 | t-3 | t-4 |
| Arkeng. | 0.298037 | 0.506558 | 0.40917 | 0.312275 | 0.242109 |
| East Cow. | 0.183328 | 0.340419 | 0.257521 | 0.192658 | 0.144858 |
| Malham T. | 0.275119 | 0.495171 | 0.410857 | 0.333811 | 0.275119 |
| Snaizeholme | 0.345594 | 0.584959 | 0.487052 | 0.389125 | 0.342289 |

|  |  |  |  |
| --- | --- | --- | --- |
|  | *Mean Daily Flow to Skelton* | | |
|  | T | T-1 | T-2 |
| Arken. | 0.876543 | 0.973021 | 0.825494 |
| Skip Bridge | 0.899654 | 0.947692 | 0.895488 |
| Westwick | 0.834773 | 0.865423 | 0.795439 |

**Moving Averages:**

I have also explored moving averages of the rainfall data set. I thought another useful predictand would be the given rainfall dataset, as a cause of flooding is prolonged rain. But to my surprise, most of the correlations to Skelton were below 0.5, which isn’t that high compared to the daily mean flow. So, I tried finding the moving averages with 2-4 steps. There was an improvement for correlation by about 0.2 for each column with Snaizeholme having the best results.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | *Rainfall Moving Averages Correlations to Skelton* | | | | |  |
|  | Ark. |  |  |  | Malham T. |  |
| 2step | 3step | 4step |  | 2step | 3step | 4step |
| #DIV/0! | 0.580637 | 0.586258 |  | 0.457563 | 0.435432 | 0.165424 |
|  |  |  |  |  |  |  |
|  | East Cow. |  |  |  | Snaizel. |  |
| 2step | 3step | 4step |  | 2step | 3step | 4step |
| 0.357423 | 0.390701 | 0.398881 |  | 0.577865 | 0.672299 | 0.668934 |

I even tried lagging the data set but the correlations had little improvement after one day of lagging. Then an immediate drop in correlation after 1 day of lagging. I even tried creating new data, by calculating the average of each row of rainfall data and lagging each day to T-4. But then again the correlation values are disappointing again. So I decided not to carry on using the moving averages for my neural network.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | *Rainfall moving average* | | |  |
| **T** | **T-1** | **T-2** | **T-3** | **T-4** |
| 0 |  |  |  |  |
| 0.2 | 0 |  |  |  |
| 0.2 | 0.2 | 0 |  |  |
| 22.4 | 0.2 | 0.2 | 0 |  |
| 40.4 | 22.4 | 0.2 | 0.2 | 0 |
| 0.6 | 40.4 | 22.4 | 0.2 | 0.2 |
| 15.8 | 0.6 | 40.4 | 22.4 | 0.2 |
| 1.4 | 15.8 | 0.6 | 40.4 | 22.4 |
| 43.8 | 1.4 | 15.8 | 0.6 | 40.4 |
| 59 | 43.8 | 1.4 | 15.8 | 0.6 |

A sample of *feature construction from the lagged average of rainfall in a day.*

**Feature Construction:**

I created new predictands by using the previous days’ results of Skelton. This had a strong correlation of nearly 0.9. I chose this predictand as there must be a relationship between the current and last daily mean flow values. I created this feature by copying the column into an empty column and lagged the column by one day. This would then give the results of the previous Skelton result, for use in predicting the current day.

|  |  |
| --- | --- |
| **Skelton Pre** | **Skelton** |
| 26.1 | 24.86 |
| 24.86 | 23.6 |
| 23.6 | 23.47 |
| 23.47 | 60.7 |
| 60.7 | 98.01 |
| 98.01 | 56.99 |

*Feature construction of Skelton pre. By using Skelton data.*

Also, mentioned in the moving averages section I constructed the rainfall moving averages. More detail about it is written in that section.

**Feature Encoding:**

|  |  |
| --- | --- |
| 30/01/1993 | 1 |
| 31/01/1993 | 1 |
| 01/02/1993 | 2 |
| 02/02/1993 | 2 |

Feature encoding is how features can be converted in a way a statistical model or neural network can utilize it. The raw data set is ordered by dates in the format DD/MM/YYYY. These values cannot be used as it is not a real-number. To make this data useful, I created a new column and entered the corresponding month to each row.

The left column are the dates, and the right column are the corresponding month numbers. So, this column will have a range between 1-12 for each month within a year. I have decided to use feature encoding on the dates as floods tend to be seasonal. So, I feel the neural network should pick up a pattern between certain seasons with high daily flows.

**Determine predictands to be used in the neural network:**

**???????????????????????????**

**????????????????????????????**

**Implementation of MLP (35%):**

**Contents of my MLP Implementation:**

* Base Algorithm
* Re-shape neural network structure as needed. The only constraint is that this implementation is only limited to one hidden layer.
* Sigmoid and Hyperbolic-Tan Activation Functions.
* Momentum
* Weight Decay
* Simulated Annealing

**Justification of using Python:**

I have decided to program my implementation in the python programming language. Originally, I decided to program this in Java which is my most comfortable language. But from looking over all the modules python has, and the fact that python is the most dominant language in data science right now; I decided to use python as it will help prepare me for my future interests in machine learning.

**Imports used for this implementation:**

For my implementation of this code, I have used only three modules. The first is the math module for simple maths calculations. It is the least used module out of the three as most of the maths calculations are matrix operations, which are handled via the NumPy module.

NumPy is a mathematics module that is widely used for scientific computing. In my implementation, that module is mainly used for the matrix operations for the feedforward and backpropagation algorithms. Matrix operations are used as it is a very convenient way to calculate the outputs of the nodes and acts as an efficient data structure for them. Operations such as the dot product, Hadamard product give a convenient way to pass data from one layer to the next. More on this will be explained later.

Lastly, pandas is an open-source data analysis and manipulation module. For this implementation, I have used this package to retrieve certain excel sheets from a .xlsx file containing all the flood and rainfall data. This package makes it very easy to do these operations as the commands to do so are just a couple of lines.

Text

Description automatically generated**MLP Constructor:**

The \_\_init\_\_ constructor method allows the user to specify the structure of the multi-layer perceptron. They can specify any number of input, hidden, and output nodes. But with the constraint that this implementation cannot allow multiple hidden layers.

The constructor stores the number of nodes and then uses them to create the dimensions of the weight matrices and biases. Initially, the weights are randomly assigned between the ranges of [-2/ Input Count, and 2/ Input Count].

We initially set the learning rate to 0.1, but through later evaluation, the final model can change. The learning rate is the rate at which gradient descent passes through the weight space. Too high, and it can start oscillating between minima. If too low, the algorithm will barely learn and may have to go through many epochs to find the global minima.

The next attribute is the validation error. This is initially set to 1 which is later on compared to the next validation root mean squared error. By being set to one it will allow the program to continue running a bit longer before the validation error stops training. This is to allow the chance for the validation error to go back down. More on this will be explained later.

Next, the learning rate range and momentum are attributes used for the improvements to the base algorithm. Self.learningRateRange is a boundary for the simulated annealing algorithm to make the learning rate stay within at. Self.momentum is simply a coefficient for quickening the process of moving past the weight space.

Lastly, self.currentEp is simply a value to set a new neural networks epoch to 0.

Text

Description automatically generated**Activation Functions & Derivatives:**

Text

Description automatically generatedThe def sigmoid and def tanH functions are used to apply the activation functions to the weighted sums of the hidden and output layers. As the gradients for the node outputs are needed to be calculated for gradient descent, I have implemented def dSigmoid and def dTanH functions to find the derivatives of each hidden and output node.

**Feed-Forward:**

This method takes a row of data from the excel spreadsheet and creates a 2-dimensional NumPy array out of it for matrix operations. These standardized input values are used to create the weighted sum of the hidden nodes via the dot product. Each weighted sum is then added by its respective hidden bias. When all of this is done, we apply the appropriate activation function to the sum. In this case, hyperbolic tan is used. Likewise, the same process is repeated for the output layer where the hidden layer results are now passed to the output layer. This method then returns the guess of the neural network and then the hidden and input data in matrix form.

**Back-propagation:**

Text

Description automatically generated

The purpose of the backpropagation algorithm is to return the hidden and output deltas for the update weights method. This algorithm takes the returned values of the feed-forward algorithm and also takes the correct answer of that specific row of data.

The output errors are calculated using the formula **Oerrors = correctValue – guess**. Both of which are matrices of the same size. If weight decay is applied to the algorithm, we reduce the difference between correct and guessed answers. This is to reduce the rate of over-fitting to the model.

To find the gradient of the current point within the weight space, we use the derivative functions of the activation function currently used. In this case, the derivative of hyperbolic tan is used to get the gradient. We then find the output’s delta by the formula **Odelta = gradientOutput \* output Error.** Similarly, the hidden deltas are then calculated via the formula **Hdelta = Weight of hidden to output layer \* OutputDelta \* gradient of hidden**. Then these deltas are correctly ordered in a matrix by a dot product between hidden deltas and input row transposed. These delta matrices for the input to hidden, hidden to output layers and output and hidden deltas are returned.

**Updating Weights:**

Text

Description automatically generated

After the backpropagation algorithm is finished, the returned values are passed to the update weights method. The weight matrices are updated first using the formulas **NewWeight = OldWeight + (delta \* LearningRate).** If the matrix is input to hidden, the hidden deltas are being used, otherwise it’s the output deltas.

But as mentioned in the backpropagation algorithm, these input-to- hidden and hidden-to-output deltas are stored in a matrix that will correctly give the right deltas for this formula to work. That is why the formula looks a bit different from the lecture slides as I had to give a solution in matrix form.

If the momentum algorithm is applied to the base algorithm, the weights are further adjusted to include **NewWeight = OldWeight + (delta \* LearningRate) + (momentumValue \*(delta \* LearningRate))**. As **(delta \* LearningRate)** is the amount of how much adjustment the old weights needed, we simply needed to multiply this value by the momentum coefficient to go down that change a bit more through the weight space.

Lastly, the hidden and output biases are updated using the formula **newBias = oldBias + (learning rate \* delta) \* 1.** The delta used is the delta that is of the same type as the bias. For example, if we are looking for the hidden bias, the formula will be using the hidden deltas. Again, my implementation is using matrices so all the deltas and biases are stored in the matrix.

**Simulated Annealing:**

A screenshot of a computer

Description automatically generated with medium confidence

Simulated annealing in the context of neural networks is a process where the learning rate changes over each epoch to a smaller learning rate. Within the constructor method a self.learning rate range contains an array of two values specifying the highest learning rate and the lowest learning rate. During training, the learning rate gets decreased proportionately to the number of epochs left to do. So by the end of training the learning rate will reach the lowest value in that learning rate range. This is done by calling this method each time to adjust the current learning rate value.

**Weight Decay:**

Text

Description automatically generated

Weight decay is a process that penalizes large weight values by adjusting the error function of the output nodes. Large weights on the output node can lead to output results beyond the range of the training data. If these values were not restricted in some way, the observed-modelled graph will show a noisy response to the data and give a lot of curvatures. This is over-fitting.

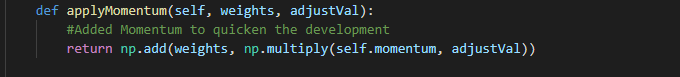
Firstly a secondary method called sumWeight is used to calculate the sum of all squared weight values in a matrix and returns that sum to the apply Weight Decay method. The number of weights is also calculated by adding the number of dimensions of each matrix together. We then find omega and upsilon via:

Diagram

Description automatically generated with low confidenceDiagram, schematic

Description automatically generated

The method then returns the product of these values. During the back-propagation algorithm and add the result to the output error.

**Momentum:**

As mentioned in the update weights section, this modification allows the training phase to be sped up by adding a fraction of the increment or decrement value to the new weights. This algorithm will create faster progress across the weight space.

Text

Description automatically generated**Train (part one):**

This is part one of the train method. This method allows the neural network to cycle through a randomly shuffled training data set. Each epoch contains 729 days of data.

This method starts by retrieving the “Train” excel sheet and storing that data into a pandas data frame. At the start of each training phase, we have several empty lists that will be used to contain all the assessment model values per epoch and predicted and actual results. This will be used later on to plot are results.

Whenever a new epoch begins the training set is shuffled to prevent bias and help prevent the model to learn the order of training. This is very important as when it comes to unseen data, it will create a more generalized output.

This method uses two for loops to track the current epoch and iterate through a shuffled training set. Whenever the second for loop begins, we iterate each row of the data frame and take in the values necessary. Right now, it is taking 9 values as seen above as input and takes one value as output. This proportion of inputs to outputs must equal the proportion of inputs to output in the used neural network structure.

**Train (part two):**

Text

Description automatically generated

For the second part of the train method, it covers the full algorithm of the multi-layer perceptron. It allows the neural network to make a guess with the given data. Passes the correct value and node layer values in matrix form to the back-propagation algorithm to find its deltas. These deltas are then returned to the update weight method to generate the new weights. These new weights are set to the MLP class’s respective attributes.

The guess and answers are appended to the predictions and observations lists.

Whenever an epoch is finished the guesses and answers are used as arguments to the assessment models. And those results are stored in the other lists initialised at the start of the method. See Train (part one).

If simulated annealing is used, the learning rate is decremented as mentioned in the simulated annealing section. This learning rate will eventually slow the system down to reduce the amount of rapid progress through the weight space. By the time is slows down rapidly, the more likely the algorithm has found the area surrounding the global minima. But this is really tested through trial and error to find the best learning rate boundary.

The next section of code initiates the validation method to check if the model is over-fitting. This method is initiated every 50 epochs. More on this will be explained in the next section.

Lastly, the data obtained after training is stored in several text files, that are later copied to excel for plotting.

**Validation:**

Text

Description automatically generated

This method is formatted in the same exact way as the train method but with a few exceptions.

The validation method takes data from the validation data set.

Also, when validation is called from the training set, a forward pass to the validation set is provided without any backpropagation. This is to test the program to see if the results are over-fitting.

We test if the results are over-fitting if the validation set’s root mean square error has worsened to a certain gradient since the last time it has been called. This code is found within the last part of the train method. If so, the training set is finished. If not, the training is continued until the last epoch is finished.

**Test:**

Text

Description automatically generated

This code is essentially a repeat of the validation set but it is called after training is finished, not during like the validation set. This is to test if the neural network can predict values to a suitable degree that is not under-fitted or over-fitted. These results are again put into specific text files to be plotted in excel.

Text

Description automatically generated**Assessment Models (part one):**

This part of the assessment models sections contains the mean square error and R-Squared models.

Means squared Error is used to calculate the amount of error between the predicted and observed values and shows us the average error.

A picture containing text, clock, gauge

Description automatically generated

R-Squared is also known as the coefficient of determination and it measures the dispersion between observed and predicted values throughout the assessment.

Text

Description automatically generated

**Assessment Models (part two):**

Text

Description automatically generated

Coefficient of efficiency is also used to measure how much knowledge the model has to predict the data. It ranges from -infinity to 1.

Text

Description automatically generatedNext, the mean square relative error is used to calculate the error relative to the observed value, which gives a fairer indicator of how well the model is performing.

A picture containing text

Description automatically generatedText

Description automatically generatedLastly, root mean square error is an assessment model that records the overall match between observed and modelled data.

Text

Description automatically generated**Record data to text files:**

This method is called upon when an epoch is finished. The lists of all data points stored in the lists are written to certain text files. Looking over this code, I can easily make use of one small function that performs writing to data. I might implement this later in development.

Shape

Description automatically generated with medium confidence**Main method:**

Lastly, these lines of code run first in the program. It starts off if assigning a certain neural network structure, in this case, 9 inputs , 18 hidden and 1 output node. It then calls the train function to obtain the best weights to predict the model. This last for 150 epochs. Then we test the found weights to an unseen data set in the test method. This is just an example of epochs and neural network structure, so its not likely the best option to use for this use-case.

**Training and network Selection (20%):**

**Evaluation of final model (20%):**

**Comparison with other models (10%):**

**Overall:**

**References:**

*Table

Description automatically generatedTable

Description automatically generatedTable

Description automatically generatedGraphical user interface, application

Description automatically generated with medium confidenceFigure 1: Figure 2:*

*Figure 3: Figure 4:*

*Table

Description automatically generatedFigure 5:*

Chart, line chart

Description automatically generated

A picture containing graphical user interface

Description automatically generatedText

Description automatically generated