ANN Implementation Report

Data Processing

Data Cleansing:

With access to the initial data set, I first began by performing a range of initial data checks starting with the cleansing of the data. For this, I decided to use Microsoft Excel as it is much easier to view and edit a large dataset while still having sufficient tools to perform mathematical calculations.

I first checked to ensure that all the data points were filled with at least some value and that there were no missing dates in the sample. With this confirmed, I moved on to checking that every cell contained a floating-point value which was not the case as several cells contained the character “a”. Having marked these cells down, I then checked for any spurious values in the data set. Here I found several values marked down as ‘-999’, several very large values and a couple of zeros in mean discharge which did not look feasible in the data surrounding. It is unlikely that the river would have run completely dry, especially when its tributaries have not.

With all these missing and spurious values, I then applied linear interpolation to approximate what these values should be as removing them is not feasible in a time-series data set. To do this I used the built-in XLOOKUP function inside Excel. With all the data values correctly filled, I then looked at removing data values over 3 standard deviations from the mean. While this is fairly standard practice for many datasets, I found that applying this suggests that nearly half of the data was an outlier. This is likely because, for rainfall, many of the values were 0 which drags the mean down. I tried up to 5 standard deviations from the mean but felt it was classing too much of the data incorrectly so decided to continue with the data set as was.

Data Pre-processing:

Having completed my initial data cleansing steps, I moved on to preparing my data to be fed into my program. The first step was to select meaningful predictors and predictands to give the ANN the best chance of producing predictions with a low error. I started simply by just using all 8 given inputs on the data set, however, I will be changing the predictors in the future to see if they give different errors. Having decided I was going to use all data inputs, my next step was to create the appropriate time lag which would be to augment the data by one day so we can use the previous day's data to predict the flow of today.

With this complete, I then exported the data to a CSV so I could bring it into Python. Once in Python, the data is stored in a Pandas Array which allows me to split it into a training and testing set. I decided to allocate 25% of my data for testing and 75% for training as this allowed me to use 3 complete years of data which would avoid

seasonal bias. This left 365 days for testing. With the data correctly split, each set was then standardised between 0.1 and 0.9 relative to the dataset's minimum and maximum values. With the data ready, it can now be run through the backpropagation algorithm.

Implementation

Language and Configuration:

I decided to use Python to implement my backpropagation algorithm as I have experience using Numpy and Pandas which are very well suited to this type of problem. In my implementation, I use one class called ‘Neural Network’ which handles the end of the data pre-processing, the backpropagation algorithm and predictions. An advantage to this is I can create multiple instances each with different values for the number of hidden nodes, learning rate and the number of epochs. By default, I will be using 400 epochs to train my model as this gives the MLP sufficient time to learn without risking overtraining. For the models I have produced where the number of epochs is varied, this will be stated. This will allow me to run the program once and come back later once it has generated a variety of results. Initially, I implemented a sigmoid transfer function but plan to implement the tanh(x) function as well, to compare models.

Weights and Bias:

Once the class has been called, random weights and biases are generated and assigned using the get_hidden/get_ouput functions. These, along with the data, are then given to the forward_pass function which begins the process of calculating the weighted sums for each of the hidden nodes and output layer. These weighted sums are stored in an array called node_activations which can then be used in the back_pass function which calculates the current error, before updating them in update_output_weights and update_biases. To allow my program to be adapted instantly to any number of inputs and hidden nodes, the weights are calculated from the top down between the input and hidden layer and then back to the top for the output layer. This allows me to index all of the weights with [j+(i * number of inputs)] where j is the number of hidden nodes and i is the number of inputs. I am then able to repeat this same process backwards in the backpass to update them all.

Prediction:

After the user-defined number of epochs has passed, the function prediction is then run which takes the training portion of the data and runs the forward pass on the remaining data which results in the MLP making a prediction based on the inputs provided. At this stage, each of the expected and predicted values are output to a CSV file which is named with the learning rate and the number of inputs for convenience.

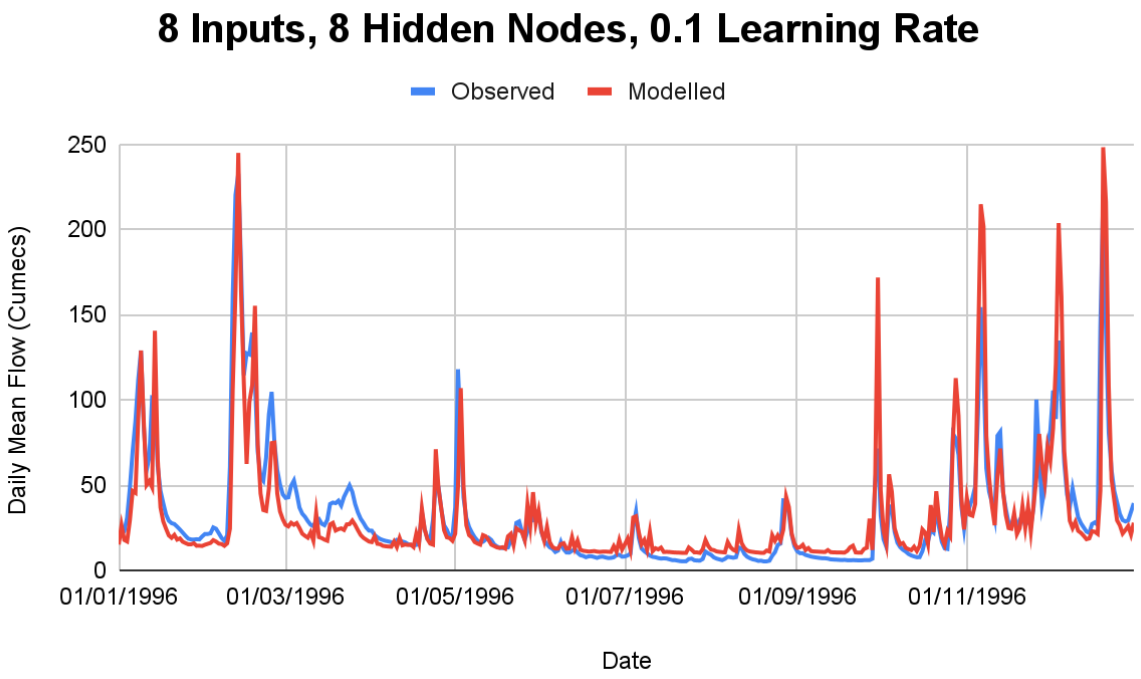
Initial Results

Benchmarking:

With my code working and all the data processed, my next stage is to train and test my MLP. To compare models, I will be using the following error measures which can be seen in many of the tables in the subsequent sections.
RMSE = Root Mean Square Error, MSRE = Mean Squared Relative Error
CE = Coefficient of Efficiency, Rsqr = Coefficient of Determination
This will allow a very direct comparison of models directly related to how accurate they are at predicting Daily Mean Flow.

Results:

Initial configuration: 8 inputs, 8 hidden nodes, learning rate = 0.1 and a sigmoid transfer function.
To start with I am going to perform an initial run with the configuration above to get some benchmark values. This configuration produces the following result:
(Fig. 1)



RMSE = 16.9956
My initial thought when viewing this line graph is that my MLP captures the general trend of the mean daily flow pretty accurately. Especially in the middle months of the year where daily mean flow is low (as are values for rainfall), I would say the model matches the observed trendline well, though the model is slightly optimistic and predicts higher values than we actually see. Especially, in the later months where we begin to see some much higher values for mean daily flow (and rainfall), the model tends to take these inputs far out of proportion, in some cases predicting also double the observed value. However, this is the first attempt with no improvements and will

have a look at ways I can try to reduce the error. Here we have initially used RMSE error as it is easy to calculate but I will also look at other ways of calculating error.

Altering Learning Rate and Hidden Nodes:

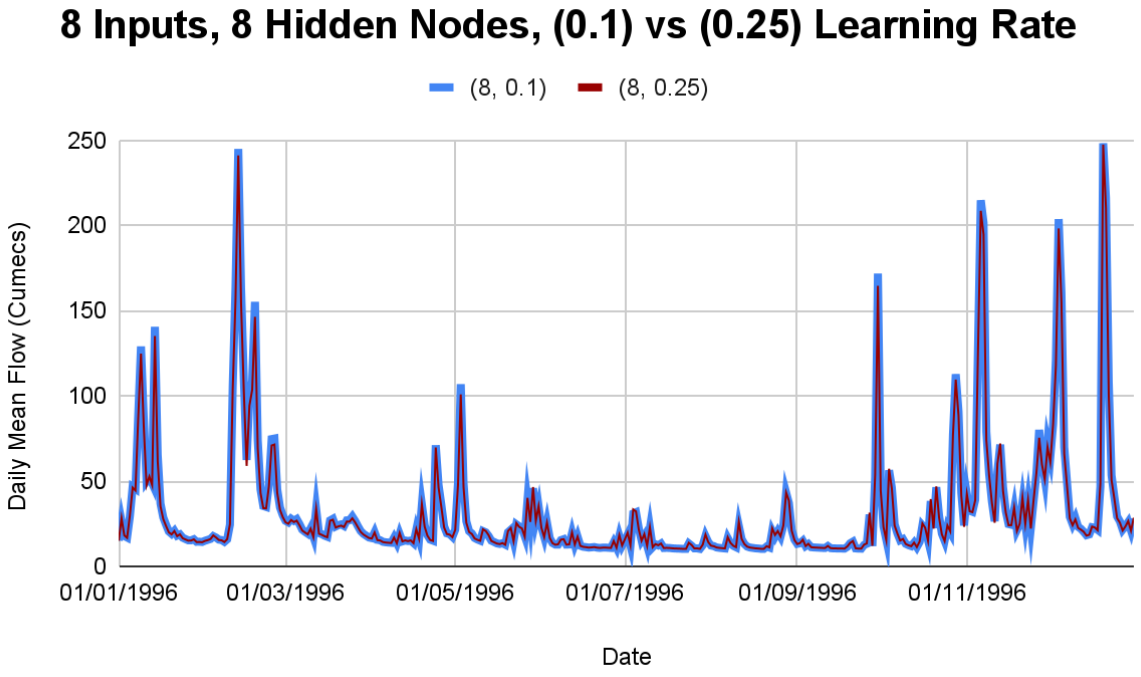
Having got a benchmark model, I am now going to try a range of values for rho and hidden nodes to see how this improves or degrades my model. These results are summarised in Figure 2.

(Fig. 2) - X = number of hidden nodes, Y = learning rate

X,Y	4, 0.01	4, 0.1	4, 0.25	8, 0.01	8, 0.1	8, 0.25	10, 0.01	12, 0.1	16, 0.1
RMSE	17.501	17.071	16.787	19.089	16.996	16.691	46.178	18.078	231.384
MSRE	0.578	0.258	0.305	0.746	0.257	0.273	2.350	0.265	434.099
CE	0.748	0.760	0.768	0.700	0.762	0.770	-0.758	0.731	-43.131
RSqr	0.767	0.796	0.779	0.736	0.790	0.788	0.737	0.769	0.724

Having run 9 different iterations with a range of hidden nodes and learning rates, we can see the error of the predicted values summarised in Figure 2. Looking at these error values we can immediately see that having a large number of hidden nodes was detrimental to the model and performed very poorly in nearly all cases. This is especially true when using 16 where the error values predict almost no correlation.

(Fig. 3)



Coincidently my initial values for rho and hidden nodes were actually near-optimal for this configuration. Although (8, 0.1) performed well in all error calculations, as you can see from Fig. 2 it was beaten slightly by the (8, 0.25) configuration which was

the best performing in these initial runs. Comparing my initial values to the best model ((8, 0.25) according to the error functions) in Figure 3, we can see the best model tends to be more sympathetic to the more extreme values seen later on in the time series, however, does not massively alter the overall trend shown as both have relatively similar error values.

Changing Predictors

Configuration:

Having performed several iterations using all 8 inputs, my next stage is to try changing the predictors I am inputting into my model. I am going to try 2 different sets of inputs to see how they change the error of the model we can produce. For both of these sets, I will be combining all rainfall data on a given day into a single input which is the weighted moving average of rainfall over the past 3 days. To calculate this, yesterday's average rainfall is multiplied by 0.5, 2 days ago is multiplied by 0.3 and 3 days prior is multiplied by 0.2. To implement a weighted moving average I had to add a 3-day time lag to the data set to allow enough data at the start of the series. This weighted moving average will be combined with yesterday's flow at Skelton for one set and I will be removing it from the other set to see whether yesterday's flow at Skelton helps to predict today's mean flow.

4 inputs (WMA + Daily mean flows (no Skelton)):

(Fig. 4) - X = number of hidden nodes, Y = learning rate

X, Y	2, 0.1	4, 0.01	4, 0.25	4, 0.3	4, 0.1	5, 0.4	6, 0.1	8, 0.1	8, 0.4
RMSE	20.337	19.923	20.370	20.601	19.412	20.406	19.000	18.568	19.740
MSRE	0.492	0.502	0.487	0.502	0.418	0.494	0.387	0.360	0.431
CE	0.488	0.772	0.495	0.471	0.577	0.490	0.611	0.649	0.554
RSqr	0.676	0.791	0.676	0.669	0.704	0.675	0.715	0.727	0.695

Here I have used just 4 inputs and again attempted a range of different learning rates and hidden nodes. From Figure 4, we can see that there are clearly 2 superior models. Despite this, when evaluating the models, both come up short of the RMSE benchmark of 16.691 set in Figure 2. This set of results also comes up short when evaluating MRSE and Rsqr, however, performs marginally better when looking at CE. While error calculations give a good indication that neither model is better than the one seen before, upon visual inspection you can see that the (8, 0.1) struggles to stay as close to the observed trend line throughout hence we see a higher value for RMSE. Although it does not follow the trend as well, for the larger values of mean discharge, it is not as reactive and maps the peaks much closer which is what I believe gives it the lowest RMSE and MSRE. In the case of the (4, 0.01) model, we get a model which is very tidy and sits close to the observed trend line, as is certainly much tidier around March than the one seen in Figure 2. However, where this model

loses out is its disproportionate peaks i.e. modelling 71m^3/s as 166m^3/s, well over 2x the observed value.

5 inputs (WMA + Daily mean flows):

(Fig. 5) - X = number of hidden nodes, Y = learning rate

RMSE = Root Mean Square Error, MSRE = Mean Squared Relative Error

CE = Coefficient of Efficiency, Rsqr = Coefficient of Determination

X,Y	3, 0.01	3, 0.03	3, 0.1	5, 0.01	5, 0.08	5, 0.1	5, 0.25	10, 0.1	10, 0.25
RMSE	17.574	18.765	19.294	19.296	20.882	19.681	19.164	20.787	19.173
MSRE	0.716	0.529	0.345	0.638	0.346	0.335	0.391	0.344	0.355
CE	0.745	0.710	0.693	0.693	0.641	0.681	0.697	0.644	0.697
RSqr	0.766	0.777	0.746	0.785	0.765	0.754	0.723	0.761	0.737

In this second case where I have used Skelton flow as an input, we can see once again that we have not improved the model from Figure 2. In this instance, our best model according to error functions is (3, 0.01), however, does not beat any of our previous RMSE or CE scores meaning it is not an improvement on anything we have produced before. In practice, this produces a chart that has a very similar overall trend when compared to the observed values but is just not as tight as the one seen in Figure 2. Yet again, this model also has a tendency to overpredict the larger values hence a higher RMSE/MSRE and a lower RSqr. At least on the settings I tried, I can conclude that my model does not appear to substantially benefit from knowing yesterday's flow at Skelton, however, there may be other factors at play.

Evaluation of Altering Predictors:

(Fig. 6) - X = number of hidden nodes, Y = learning rate, Z = number of inputs

X,Y,Z	3, 0.01, 5	4, 0.01, 5	8, 0.25, 8
RMSE	17.574	19.923	16.691
MSRE	0.716	0.502	0.273
CE	0.745	0.772	0.770
RSqr	0.766	0.791	0.788

I have now tested my MLP with a number of learning rates, hidden nodes over a set of 3 inputs. In Figure 6, you are able to see the best result from each set of inputs. These cells are colour coded with green being the best result for that measure of error, amber the second-best and red the worst. It is clear from this that the model using 5 inputs is by far the worst but the other two are somewhat close when categorised. Despite this, calculating the percentage difference between them is very telling, with less than a 1% difference in CE and RSqr compared to 17% and 59% for RMSE and MSRE respectively. With this in mind, it is fair to conclude that the (8, 0.25, 8) model is the best produced so far by quite a long way. As stated before it does tend to overpredict, however, generally holds the trend well.

Changing the Transfer Function

Configuration:

While I have so far produced a decent model for Daily Mean Flow at Skelton, I am now going to try to improve my MLP by using a different transfer function. For all the results produced so far, I have been using the Sigmoid transfer function so will now replace this with the Hyperbolic Tangent Function tanh(x). With this implemented, I will repeat the process above to see if there are any improvements to be made.

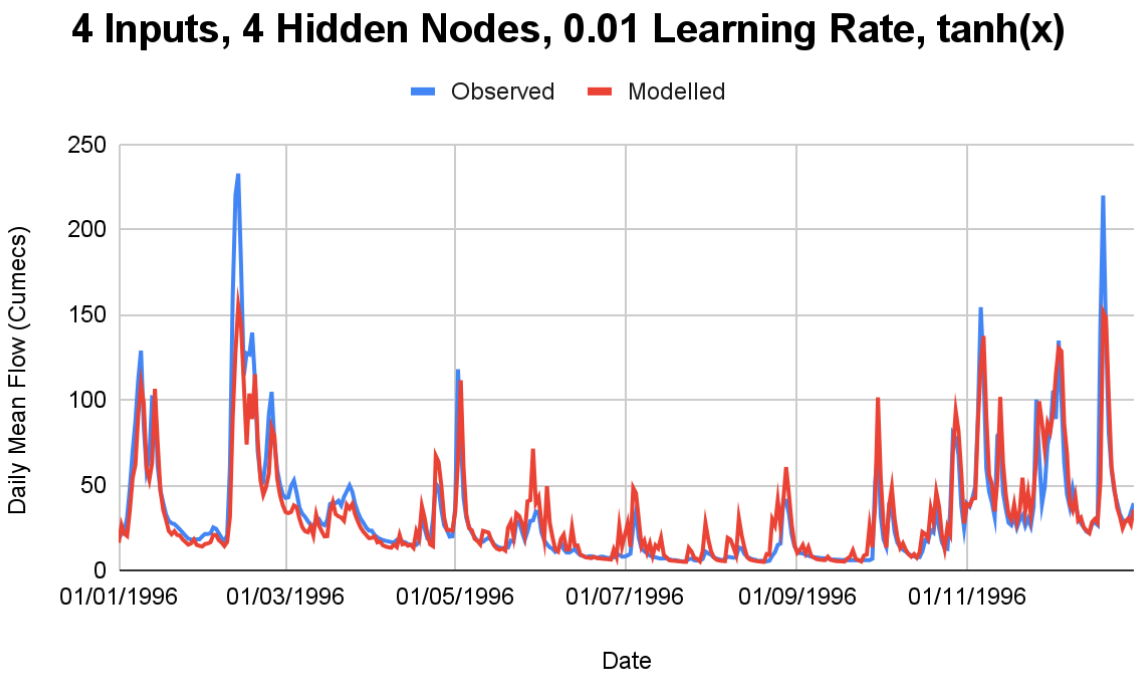
4 inputs (WMA + Daily mean flows (no Skelton)):

(Fig. 7) - X = number of hidden nodes, Y = learning rate

X,Y	2, 0.01	2, 0.1	4, 0.01	4, 0.08	4, 0.1	4, 0.25	8, 0.01	8, 0.1	8, 0.25
RMSE	23.067	16.836	15.018	17.685	17.944	42.640	15.274	20.064	43.174
MSRE	0.230	0.522	0.242	0.655	0.762	0.893	0.317	1.105	0.894
CE	0.561	0.766	0.814	0.742	0.735	-0.499	0.808	0.668	-0.536
RSqr	0.802	0.771	0.818	0.748	0.750	0.321	0.817	0.685	0.281

Having implemented the transfer function, we can immediately see a difference in the results produced by the MLP. From Figure 7, we can quickly identify that (4, 0.01) is the strongest model in this batch. The low RMSE and MSRE combined with a strong CE suggest that this model has a very good knowledge of what it is trying to predict. This is also the best model we have produced so far and can be seen in Figure 8.

(Fig. 8)



Here, we can see a really strong correlation between the observed and modelled trend line which is expected as the high RSqr score would suggest a very high coincidence of the shape. It should also be noted that this model has now begun to underpredict the high peaks in February and November/December and while they are now predicting decently under the observed, this is an improvement on the predicted values that were two times the observed.It is also interesting to see that we also produced another model which was very similar, highlighted in yellow. While this model does not map the trend quite as nicely on the whole, when graphed in Excel, I could see that it handled the peaks better than (4, 0.01) getting even closer to the actual values.

Another model of interest produced here is (2, 0.01) as it has the lowest MSRE of the batch and a very strong RSqr (highlighted in orange). Despite this, it ranks 3rd last in both RMSE and EC, only being beaten by models (4, 0.25) and (8, 0.25) which appeared to get stuck bouncing between the sides of a valley as a result of their higher learning rate. This suggests the model has a strong coincidence of the general shape but does not map the values particularly well. When graphed in Excel, this model produced a graph that had an excellent idea of the trend but was very subdue and seemed reluctant to predict anything over 80 regardless of the input.

5 inputs (WMA + Daily mean flows):

(Fig. 9) - X = number of hidden nodes, Y = learning rate

X,Y	3, 0.01	3, 0.1	5, 0.01	5, 0.08	5, 0.1	5, 0.25	10, 0.01	10, 0.1	10, 0.25
RMSE	15.335	19.434	15.461	18.684	18.912	133.062	15.619	19.521	46.142
MSRE	0.322	0.674	0.328	0.653	0.670	143.543	0.363	0.968	0.948
CE	0.806	0.689	0.803	0.712	0.705	-13.594	0.799	0.686	-0.755
RSqr	0.809	0.709	0.809	0.716	0.711	0.380	0.810	0.689	0.338

Once again with the tanh(x)function, we can see an overall improvement when looking at the results produced by our error functions (See Figure 9). For this batch, there are 3 really strong models produced, (3, 0.01), (5, 0.01) and (10, 0.01). While (3, 0.01) is the strongest, it is not better than the best model produced when only using 4 inputs, once again reinforcing my idea that yesterday's flow at Skelton is not a particularly useful input to this MLP.

When comparing these 3 models, we can see they all score nearly identically when applying RSqr error suggesting all 3 have a strong shape, but the values predicted by (3, 0.01) are more in agreement with the observed values. It should also be noted that like in the case of using 3 inputs, a learning rate of 0.01 seems to be more optimal when using the tanh(x) function on this dataset.

8 inputs (WMA + Daily mean flows (no Skelton)):

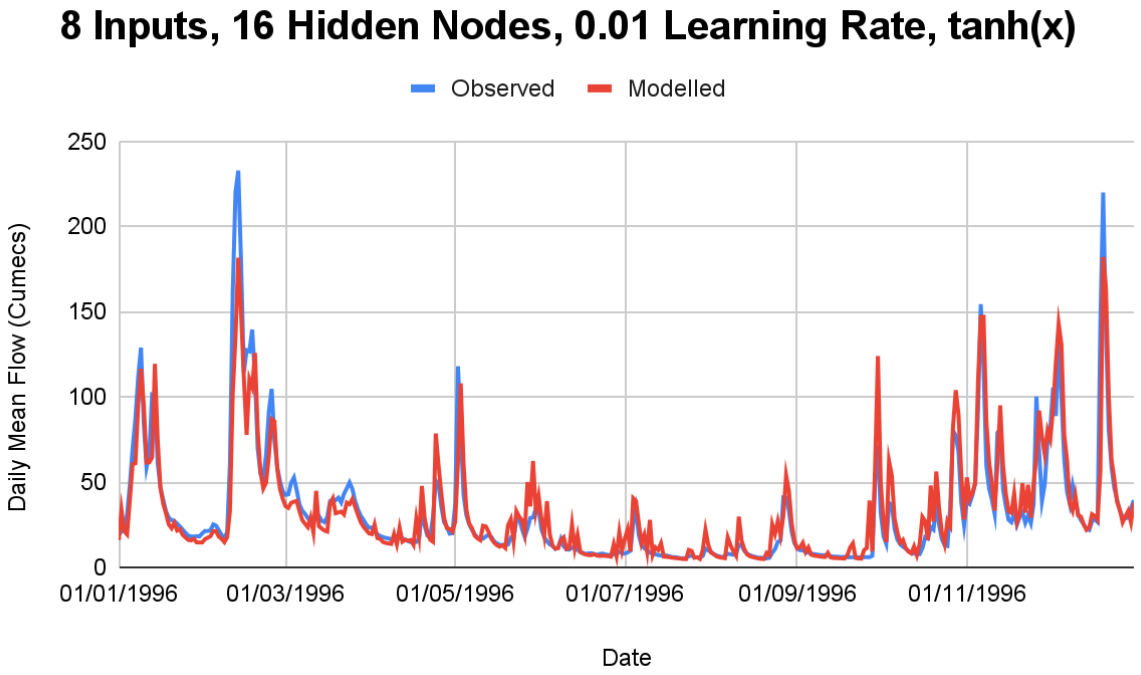
Having evaluated 4 and 5 inputs, I now returned to 8 inputs which provided me with the best results when I was altering the predictors. Once again, the tanh(x) transfer functions performed well and produced strong inputs all around. Not only this, but the best RMSE values jumped down from 15.335 to 13.652 when using more inputs and more hidden nodes. This can be seen in Figure 10.

(Fig. 10) - X = number of hidden nodes, Y = learning rate

X,Y	4, 0.01	4, 0.1	8, 0.01	8, 0.1	10, 0.01	12, 0.01	12, 0.1	16, 0.01	16, 0.25
RMSE	14.214	16.143	14.265	17.753	13.961	14.249	39.112	13.652	60.169
MSRE	0.228	0.354	0.232	0.372	0.269	0.267	0.700	0.248	1.655
CE	0.833	0.785	0.832	0.740	0.839	0.833	-0.261	0.846	-1.984
RSqr	0.836	0.800	0.835	0.762	0.840	0.835	0.033	0.847	0.195

This table seems to suggest that a smaller learning rate (highlighted in orange) when using the tanhx(x) transfer function produces much better results than the same number of nodes with a higher learning value.

(Fig. 11)



All the models highlighted in orange produce error values far less than our initial model, however, the model (16, 0.01) is the best we have produced so far and is plotted in Figure 11. It is clear that it matches the overall trend very accurately, deals well with flow values between 100-150 and has a good attempt at modelling the values greater than 200 (though does still fall slightly short).

Evaluation of Changing Transfer Functions:

(Fig. 12) - X = number of hidden nodes, Y = learning rate, Z = number of inputs

X,Y,Z,Func	16, 0.01, 8, tanh(x)	8, 0.25, 8, Sigmoid	Improvement (%)
RMSE	13.652	16.691	20.031
MSRE	0.248	0.273	9.597
CE	0.846	0.770	9.405
RSqr	0.847	0.788	7.217

Having completed tests with all 3 sets of predictors using the tanh(x) transfer function, it is clear from Figure 12 that there has been a significant performance improvement across all error measures. This works out to an 11.6% average improvement per measure. The most significant of these is in RMSE at 20% which is very clearly visible when comparing Figure 11 to Figures 1 and 3 which were the initial plots my MLP produced. While this model is good, other improvements can be made to my algorithm to see if I can reduce it further. Overall, I would say the tanh(x) transfer function is much better suited to my data set and algorithm as it produces much more accurate predictions across the board regardless of the configuration used.

Improvements

Having performed several tests with a variety of tests on my MLP I have been able to produce a decent model for Daily Mean Flow at Skelton. The next stage is to implement several improvements to the model to try to reduce the error in the results being produced. To do this, I am going to use the strongest model I have produced so far and see the impacts these improvements have.

Adding Momentum:

When adding momentum to my (16, 0.01, 8, tanh(x)) model, I did not notice a particularly big reduction in error when compared to the standard model. During my initial test, I found purely adding momentum had little to no impact on the final error value. However, when using two identical configurations with momentum turned on and off, it was clear that the model using momentum was far quicker at getting down towards the global minima. In general, this is an improvement as it allows me to run fewer epochs while still getting down to a small error value. This comparison can be seen in Figure 13.

(Fig. 13) - Error after 300 epochs

X = number of hidden nodes, Y = learning rate, Z = (1 = Momentum Active)

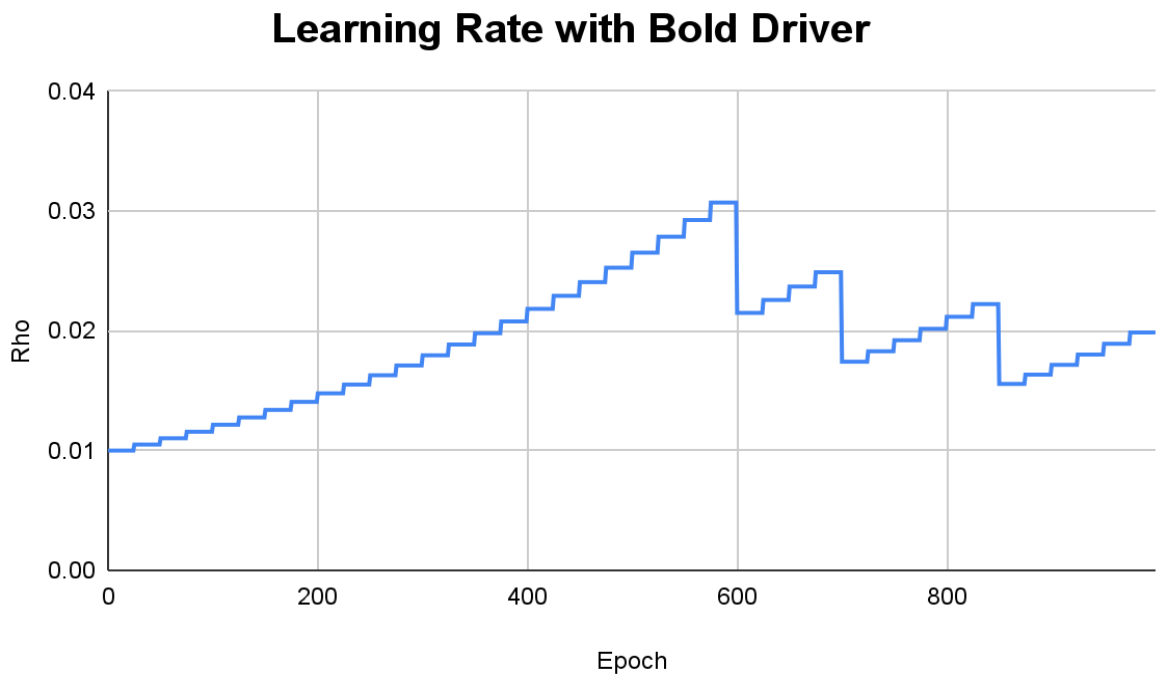
X,Y, Z	4, 0.01, 1	4, 0.01, 0	8, 0.01, 1	8, 0.01, 0	12, 0.01, 1	12, 0.01, 0	16, 0.01, 1	16, 0.01, 0
MSE	0.546559	0.565159	0.523819	0.546362	0.519552	0.521531	0.481520	0.501234

Adding Bold Driver:

The next improvement I implemented into my program was bold driver. This will allow my MLP to update its learning rate (rho) automatically as it runs through its training cycle. To implement this, I set a variable to keep track of the MSE which could then be compared to the current MSE every 25 epochs. If the current MSE is less than the stored MSE (i.e error has decreased), the learning rate is multiplied by 1.05x and the current weights/biases are saved to a variable. If the error has increased, then the weights and biases are reset to their values 25 epochs ago and the learning rate is multiplied by 0.70x to decrease it and the algorithm continues. For this implementation I put boundaries on the learning rate of [0.01, 0.5] and to avoid a snowballing effect the learning rate is not updated every epoch.

When running a model with both momentum and bold driver active, I managed to produce almost the identical model as can be seen in Figure 11. Looking at Figure 14, we can see the learning rate change over 1000 epochs. Especially as the MSE starts to get really low, approaching 600 epochs, we can see how bold driver regresses the model and decreases the learning rate to try to find an optimal learning rate.

(Fig. 14)



Comparison to Simple Multiple Linear Regression Model

While developing an MLP to solve complex problems can be very useful, there is little point in developing such a complex model if a simple model can predict with a similar degree of accuracy. To check, my next stage was to develop a Simple Multiple Linear Regression (SMLR) model to compare to my best MLP. To ensure a

fair comparison, I used the same 8 inputs as I did for the MLP instance that gave me my best results. With all the data brought into Excel and the regression performed, I used the generated coefficients for each input (Figure 16) with the *SUMPRODUCT* function on each row of data to generate a predicted value for each row of test data. The data generated is then graphed against the observed data in Figure 17.

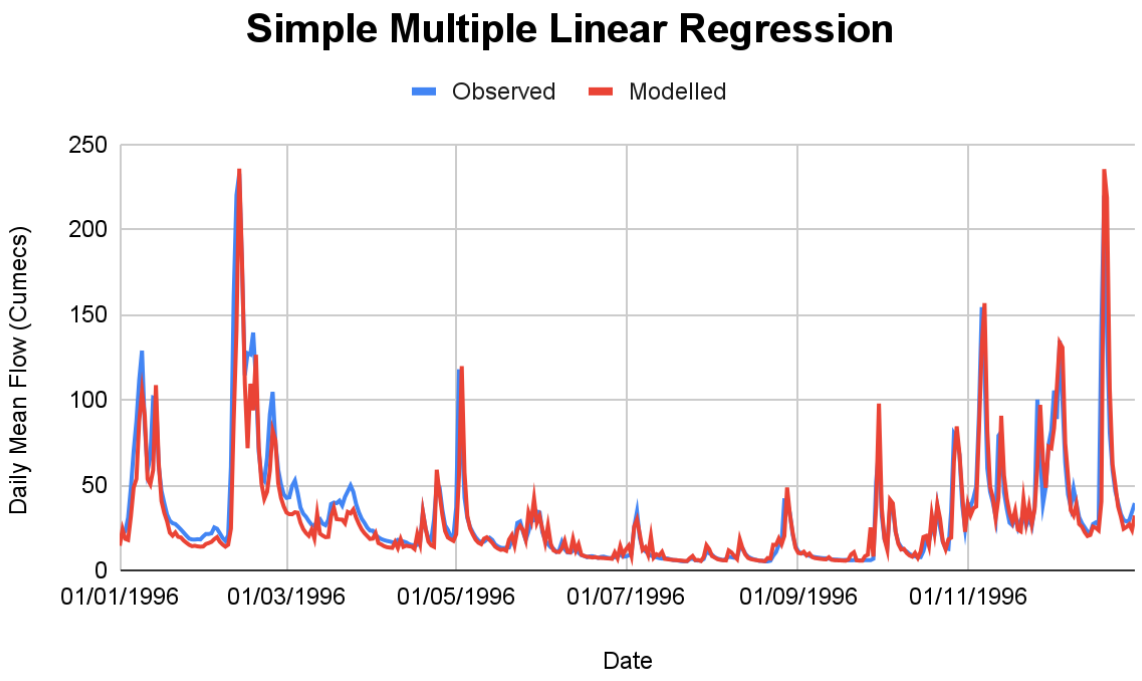
(Fig. 15)

Regression Statistics	
Multiple R	0.947026151
R Square	**0.896858531**
Adjusted R Square	0.896361293
Standard Error	17.93477363
Observations	1095

(Fig. 16)

Coefficients	
Arkengarthdale	0.17858187
East Cowton	0.02453028
Malham Tarn	0.07898257
Snaizeholme	0.43729087
Crakehill	0.51075162
Skip Bridge	1.84799208
Westwick	0.5615137

(Fig. 17)



As can be seen from Figure 17, the SMLR produces a very strong model for Daily Mean River Flow at Skelton. While this is obvious from Figure 17, looking at Figure 15, the Regression Statistics which are calculated by Excel, we can see this model scores an R Square value of 0.8968 (or 89.68%). In the context of the problem, this equates to ~90% of the variance seen in daily mean flow being as a result of the inputs. The result of this is a very accurate prediction which when graphed, we see that the modelled trend line matches the observed trend line very closely. While this

model is very good, when compared to the MLP I have produced, it does not score as well for RMSE, scoring 13.691 compared to the MLP 13.652.

Evaluation

Overall, the best model for predicting the Daily Mean River Flow I can produce can be seen in Figure 11. This instance of the neural network uses 8 inputs, 16 hidden nodes, the tanh transfer function and a learning rate of 0.01 which produced an RMSE of 13.652, MSRE of 0.248, CE of 0.846 and an RSqr of 0.847. Although these error values produced are very good, the whole point of using an MLP is to solve problems that cannot be solved easily by another, simpler model. However, as can be seen in the comparison to a simple linear regression, I would probably conclude that it is not worth taking the time to write such a complex program when a much simpler approach can be used to get a result that is nearly as good. This is especially true when you consider the time it took me to be able to find the optimal settings for this dataset.

I would say that my AI Model is accurate enough to be applicable when trying to predict tomorrow's mean flow at Skelton. Especially as the York/Yorkshire area is prone to flooding, being able to predict tomorrow's flow, could give advanced warning. However, it should be noted that my model does tend to underpredict slightly when there are high levels of discharge and rainfall which is arguably less useful as you would rather the model predicted high and was wrong, especially in the case of a flood event.

To further improve my ability to predict daily mean flow I would try to implement different training algorithms such as line searching and conjugate gradients. On top of this, I would also look at how different inputs could be used and weighted, such as weighting the rainfall at Malham Tarn as a tributary in this rainfall catchment area does not contribute to the river flow until past the Skelton measuring point.

Overall, I think my model does a very good job at predicting daily mean flow at Skelton, however, could be improved further and the use of a more simple linear regression model also outputs good predictions for this problem.