University of Reading

Department of Computer Science

Using Neural Networks for Time-Series Analysis of UK River Flow Data

Michael Neele

Supervisor: Dr Varun Ojha

A report submitted in partial fulfilment of the requirements of
the University of Reading for the degree of
Bachelor of Science in *Computer Science*

April 29, 2021

Declaration

I, Michael Neele, of the Department of Computer Science, University of Reading, confirm that all the sentences, figures, tables, equations, code snippets, artworks, and illustrations in this report are original and have not been taken from any other person's work, except where the works of others have been explicitly acknowledged, quoted, and referenced. I understand that if failing to do so will be considered a case of plagiarism. Plagiarism is a form of academic misconduct and will be penalised accordingly.

Michael Neele
April 29, 2021

Abstract

Time-series analysis using neural networks has been an active area of study since the introduction of artificial neural networks in the late 1950s, though hydrological applications of such analyses are a relatively more recent development. This report begins with taking a look at some existing research in the field of time-series analysis on both synthetic and real world data. It was found that a convolutional neural network (CNN) and long short-term memory (LSTM) combination forms an ideal model for hydrographical forecasting. The objective of the study was to produce a benchmark model from sunspot observation time-series data and additional models with the capability of forecasting river flow data from three gauging stations in the UK national river flow archive database (Jeffy Knotts, Miller Bridge House, and Colney). Using Keras and TensorFlow to implement the planned model, it was found that the capabilities of the models when trained on the river data was varying. The relative loss in the benchmark predictions was 18.95. The Jeffy Knotts model was the worst performer with a relative loss of 36.98, the Miller Bridge House model was found to perform similarly to the benchmark model with a relative loss of 21.46. The Colney model far outperformed the others with a relative loss as low as 2.58. This demonstrated the capabilities of such CNN-LSTM combination models to accurately predict river flow data in some circumstances.

Keywords: Time-series analysis, Artificial neural network, Long short-term memory, Convolutional neural network, Forecasting.

Report's total word count: 8687

Acknowledgments

I would like to thank Dr Varun Ojha for his continued support and guidance throughout the duration of the project.

Contents

List of Figures

Chapter 1

Introduction

1.1 Background

Time-series analysis is the statistical analysis of numerical data that is collected over time at regular intervals, for example daily airline passengers or stock market closing prices. The potential to analyse and find patterns in the data can lead to improving business decisions, forecasting and preparing for the future, or simply for academic exploration. A time-series may trend over time or follow seasonal patterns, it could even contain both of these aspects, or neither.

The use of artificial neural networks for time-series analysis has been around for as long as the earliest neural network research in the late 1950s. However, there remained relatively limited research into the modelling and forecasting of river flow data up until the early 2000s.

1.2 Project objectives and approach

The aim of the project was to produce an artificial neural network with the ability to analyse and predict river flow rates over time with a similar level of precision to the analysis of a benchmark dataset.

In order to achieve this, the first step was to research neural network models commonly used for time-series analysis. Following the research findings, an algorithm was designed to read the data and perform some minor pre-processing to prepare it for analysis with the neural networks. State-of-the-art data was collected and passed through the program, training a neural network model. After the model was trained, the performance was evaluated by comparing predicted values to the actual values contained within the original dataset. Graphs were produced of the outputs to visualise the performance of the model.

River flow data was then collected to train a similar model to the benchmark. The results were similarly validated and similar graphs produced to compare to the outputs of the benchmark neural network model. Conclusions then could be made regarding the value of such an algorithm when applying the solution to new data.

1.3 Organisation of the report

The report is split into 7 chapters, including the introduction. Each of these chapters is numbered 1 through 7 with numbered subsections of each chapter. Subsection numbers all contain their chapter number as a prefix (e.g. section 5.3 is chapter 5, section 3)

After this introduction, there is a literature review section, containing preliminary reading into existing research related to the project and its objectives.
Following the literature review is the main methodology chapter. This discusses, in detail, the approach taken during the planning and implementation phases of the project.
The results chapter contains the outputs generated from the project implementation of both the benchmark and river flow models. Comparisons are drawn between the outputs for further discussion in the next section.
The discussion and analysis chapter takes a deeper look into the comparisons between the benchmark and river flow outputs, which are used as a basis to determine if the project objectives have been sufficiently met.
Following the discussion is a chapter containing conclusions drawn from the implementation and results. Potential future work and improvements are also discussed, providing direction for future research regarding similar topics.
The final chapter contains a self-reflection regarding my positive and negative experiences during the development of the project.

Chapter 2

Literature Review

2.1 Existing solutions for time-series analysis

2.1.1 ARIMA vs artificial neural network models

There are currently two most prevalent methods used in academia for time-series analysis: Traditional statistics models and artificial neural networks (ANNs). The Box-Jenkins method for autoregressive integrated moving average (ARIMA) models (Box et al., 2015) is one such method of traditional analysis. With this method a time-series is assessed and some preperatory calculations are performed to apply to the model. These values are then used to tune the model to estimate a fit for the time series. Artificial neural networks of many varieties have become more widely used in recent times. These models often use methods such as backpropagation and deep learning to train more complex models for similar analysis.

There has been a lot of research published comparing the traditional methods to ANNs. Adebiyi et al. (2014) explore the differences between multilayer perceptrons trained with backpropagation and a traditional Box-Jenkins ARIMA model. Using the models for the prediction of stock prices, they found that while both were able to reasonably well predict future prices, the ANN model was superior.

Similar research from Chen and Lai (2011) supports these findings through applying similar methods to wind speed forecasting. They concluded again that the ANN model outperformed the ARIMA model when forecasting in the short-term.

2.1.2 Neural network model comparisons

With the establishment that artificial neural networks are consistently superior to traditional methods, there has been some movement away from ARIMA and towards comparing different ANN models for their individual strengths. In Oancea and Ciucu (2014), the authors performed some experiments with daily foreign exchange time-series data. They tested several training algorithms for feed-forward neural networks: backpropagation, resilient propagation (RPROP), and RPROP+. They found the networks to perform similarly in terms of accuracy, with RPROP converging faster than the other models. The authors also tested a recurrent neural network (RNN) which, after training, achieved error rates an order of magnitude lower than the feed-forward models tested.

In a similar vein, Lu et al. (2020) explored using a combination of a convolutional neural network (CNN) and a long short-term memory (LSTM) model for predictions of stock price time-series data. This combination was compared with the performances of several models including a multilayer perceptron (MLP), a standalone CNN, an RNN model, a standalone

LSTM model, and a CNN-RNN combination. The CNN-LSTM combination was found to provide the highest prediction accuracy compared to the other models.

2.1.3 Hybrid models

A paper was published by Zhang (2003) regarding a combination of both the traditional ARIMA method and ANNs to utilise each ones strengths in unison. Using several test datasets: annual sunspot numbers, trapped Canadian lynx and a GBP/USD foreign exchange pair, the results concluded that analysing the data with the ARIMA model followed by the ANN, the hybrid model was able to outperform both individual models in almost all cases. The only exception being the mean absolute deviation, which for the sunspot data, was lower during short term forecasting.

2.2 State-of-the-art data analyses

Both real and synthetic data can be used in time-series analysis. A common form of simulated data is created using a series of differential-delay equations describing physiological control systems (Mackey and Glass, 1977). Known as the Mackey-Glass equations, these are sometimes used to create benchmark data for analysis. Lopez-Caraballo et al. (2016) use data generated by these equations to train an ANN with a particle swarm optimisation algorithm. The ability to tune the input data to be either noiseless or noisy provided valuable on the performance of their algorithm.

Some other time-series data that is commonly used tracks sunspot observations over time, either annually (Zhang, 2003) or monthly (Pala and Atici, 2019). These datasets include observations all the way back to the 18th century, so are quite comprehensive and provide a good overview of the behaviour of the data over a long time. The data contains an 11-year cycle that also provides a good seasonality aspect to the data.

2.3 Hydrological applications of neural networks

Aside from the common application in stock market analysis, another frequently visited area of research comes from hydrology. The analysis of rainfall and river Flow data provides valuable information to governments across the globe to assess flood risks. Improvements in hydrological time-series prediction could save lives and significant cost in the long run, with floods causing an estimated USD 200 billion in damages worldwide in 2020 alone.

Hydrology has seen quite some adoption of ANNs for time-series prediction. A feed-forward neural network trained using backpropagation is used by Campolo et al. (1999) with input data from several rain gauges along the river Tagliamento in Italy. The authors were able to produce reasonable predictions of the water level between 1 and 5 hours in advance. CAMPOLO et al. (2003) takes a similar approach to predicting the water level in the river Arno in Italy. This time, rainfall data was taken in addition to hydroelectric power generation from the river. A similar feed-forward ANN trained using backpropagation was used, again predicting the water level reasonably well between 1 and 6 hours in advance.

Recently, a lot of hydrology research has focused on using LSTM models for forecasting. Le et al. (2019) used daily precipitation data and flood discharge data from several sources around the Da River in Viet Nam. After training an LSTM model with this data, the model demonstrated reliable performance in flood forecasting up to 3 days in advance. Similar methods used in Widiasari et al. (2018) and Li et al. (2020) yielded comparable results, providing reinforcement of LSTM models' viability for hydrological time-series analysis.

2.4 Critique

A significant amount of the research discussed provides good insights into the use of ANNs over traditional ARIMA methods for time-series analysis. Most of the authors support one another's findings and build upon the practice of using these new methods. While ANNs don't outshine ARIMA in absolutely every scenario, it is fair to conclude that their ability to handle non-linear relationships between inputs and outputs gives them a slight edge in chaotic time-series analysis.

A lot of the hydrological research evaluates the use of LSTM models as an ideal approach to river flow and flooding predictions. Most of the research concluded that RNNs outperformed basic feedforward models, and LSTM models performed towards the higher end of the recurrent models.

2.5 Project solution

The method of approach for this project was based on the hydrological research discussed in this chapter. A CNN-LSTM model was evaluated as a valid approach to the problem of analysing river flow time-series data. However, rather than using rainfall data from a number of rainfall gauging stations, it appeared that there was a blind spot in research regarding using exclusively gauged river flow data with the ANN models. As such, it was decided to use gauged daily flow of several independent rivers to train and validate the developed models.

In order to evaluate the performance of the developed models, experiments with a state-of-the-art dataset - monthly sunspot observations - would be performed. This is to provide a benchmark for the performance of the final models as their results can be compared to predictions of long-term cyclical data.

2.6 Summary

A number of research papers were analysed and discussed, many of them finding that newer artificial neural network models outperform more traditional methods of time-series analysis. Research centred around hydrology has seen a general shift towards using ANNs, and in very recent years, has been refined towards the use of long short-term memory architectures. A possible missing piece of hydrological research was identified, analysing time-series data of river flow alone. This formed the basis of the solution for the project. The decision for the use of sunspot data as a benchmark in the project was influenced by its use in multiple papers for general model evaluation. This data can be used along with river flow data to analyse the performance of a CNN-LSTM model similar to one used for hydrology research.

Chapter 3

Methodology

3.1 Design

With the objectives of the project established and problem-related literature explored, the next phase of the project development was its design. The goal of this stage was to lay out the foundations for prototyping and implementation of the final application. This included selecting development frameworks, planning and designing the overall application algorithm, planning the neural network model to be used, and deciding which outputs to produce.

3.1.1 Languages and libraries

In order to produce an application that covered all of the objectives, existing languages and frameworks were assessed for their suitability.

Python

The programming language selected for development of the project was Python version 3.8.6 (Python, 2020) released in September 2020. Python is commonly used for machine learning applications as it contains an extensive set of libraries data manipulation, machine learning, graph plotting, and more. It also provides an ideal environment for both object oriented programming and procedural programming.

TensorFlow

TensorFlow (Abadi et al., 2015) is an open-source machine learning library originally developed by Google. The version used for the project was TensorFlow 2.3.1 released in September 2020. It provides all of the necessary framework for creating deep learning neural network models and was a suitable fit for all of the project objectives.

Keras

Built on top of TensorFlow is the Keras API (Chollet et al., 2015). Keras provides a layer of abstraction when using TensorFlow, allowing for easier implementation of neural network models and more manageable code implementations. The version used for the project was Keras 2.4.0, released in June 2020.

Additional libraries

Additional libraries that were used for the implementation include NumPy (Harris et al., 2020) - a library that provides multidimensional array implementations for scientific computing, matplotlib (Hunter, 2007) - a plotting library that provides an API for graphing and visualising data, and Python's CSV library (Python, n.d.) - providing basic import and export functionality for spreadsheets and databases.

3.1.2 Artificial neural network architecture

The model produced for the project was a combination of a CNN and an LSTM.

Convolutional neural network

Convolutional Neural Networks are used in situations where there is a dependency between the input data and allows for a reduction in the number of parameters that have to be processed as opposed to a fully connected multilayer perceptron (MLP) network which would become too complex. CNNs were introduced specifically for image analysis but the same technique can also be used on time-series data.

Convolution of a 1D time-series input is a special case of the 2D convolution process with the input being a vector **x** of dimension $1 \times n$ and the kernel a vector **k** of dimension $1 \times m$. The process produces a vector output **o** of dimension $1 \times p$ where $p = (n - (m - s))$, based on stride value s. The output can be subjected to a bias **b**.

For an input series of length n, a kernel length m, stride 0, and no bias applied, the convolution equation can be given as:

$$y(a) = \sum_{i=0}^{m-1} x(a+i)k(i) \tag{3.1}$$

where $0 \leq a \leq (n - m)$.

However, if the output is not dependent on future inputs, as is the case for the river flows, then the convolution equation is given by:

$$y(a) = \sum_{i=0}^{m-1} x(a-i)k(i) \tag{3.2}$$

where $(m - 1) \leq a \leq (n - 1)$.

Long short-term memory

LSTM networks are a specific form of recurrent neural network (RNN). An RNN contains a feedback loop where the next evaluation step in a particular neuron learns from the previous step by combining outputs with inputs, thus creating a form of "memory". However, this has some limitations when it comes to carrying out time-series forecasting. Not only is setting the weights challenging, it does not address potential longer term dependencies within time-series.

LSTMs are an evolution of the RNN model, developed to overcome these issues by controlling the way in which weights are managed, how the various inputs from the feedback loops are used over time and when the historical information is discarded. This allows for the creation of longer term memory. It achieves this by creating a grouping of layers interacting with each other in a specific configuration.

3.1.3 Goals

The main objective of the project was to design and develop an algorithm that is capable of performing time-series analysis of river flow data and generating predictions of short-term future values with an accuracy similar to those obtained from a well-known sunspot observation benchmark dataset. In order to achieve this, the program was designed to work procedurally. This means that it is a simple flow process that has a clearly defined process and will end after completing all of the computation in the defined algorithm.

3.2 Data collection and preparation

Two types of data were collected for the time-series analysis neural network models; a state-of-the-art dataset used to create an initial model and produce benchmark results, and UK river flow data to produce another set of outputs for comparison with the benchmark results to evaluate the capability of a similar model to analyse unexplored data.

3.2.1 Sunspots

The benchmark dataset used was a time-series of the average number of sunspots observed monthly between January 1st 1749 up until January 31st 2021. Collected from kaggle (SILSO data/image, Royal Observatory of Belgium, Brussels, n.d.), it contains 3,264 entries, with average sunspot observation lying between 0 and 398.2 in any given month.

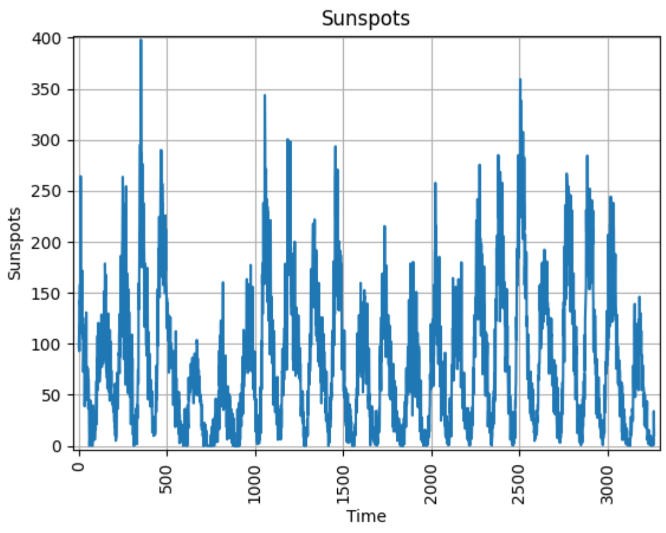


Figure 3.1: Monthly Mean Total Sunspots

Figure 3.1 shows a plot of the sunspots data. There is evidence of an 11-year cycle of peaks and troughs within the data and no trend. As all the data is accounted for in the dataset, there was no reason to modify it in any way.

3.2.2 River flow

The hydrological data used by the models was collected from the UK National River Flow Archive (National River Flow Archive, n.d.*d*). The archives contain data collected from a number of gauging stations across the UK measuring the flow rate (in cubic metres per second of water) of rivers.

Miller Bridge House gauging station

The first dataset collected is from the Miller Bridge House gauging station on the river Rothay (National River Flow Archive, n.d.*b*). This dataset contains 15,978 entries collected daily between January 1st 1976 and September 30th 2019.

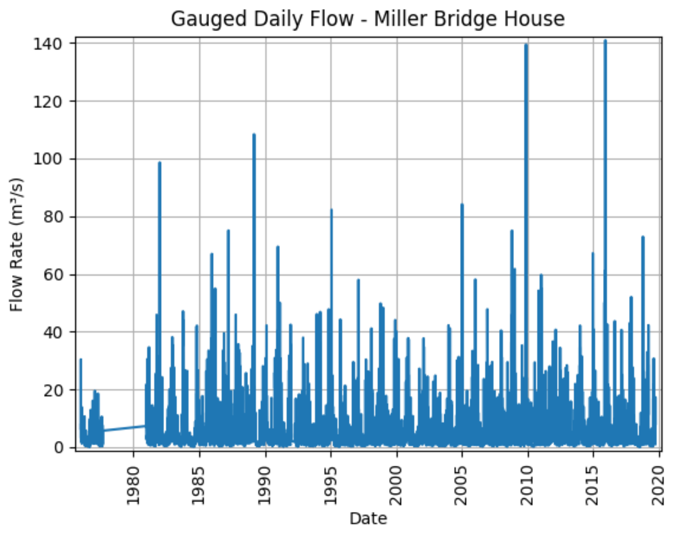


Figure 3.2: River Rothay - Miller Bridge House

In Figure 3.2 it can be seen that there is again, as with the sunspots data (Figure 3.1), a cyclical structure to the measurements. As the measurements are taken daily at gauging stations, this cycle is annual and reflects the changes of weather patterns throughout the months of the year. Figure 3.2 also shows that there is a long period of missing data between September 1st 1977 and December 31st 1980. In order to avoid any potential major impact to the performance of the neural network models, all datapoints from the beginning of the measurement (January 1st 1976) until the end of the aforementioned missing period (December 31st 1980) were removed. This leaves the dataset with a total of 14,152 entries beginning on January 1st 1981 and ending on September 30th 2019.

Jeffy Knotts gauging station

The second dataset collected is from the Jeffy Knotts gauging station on the river Brathay (National River Flow Archive, n.d.*a*). As with the Miller Bridge House time-series, this dataset also contains 15,978 entries collected between January 1st 1976 and September 30th 2019.

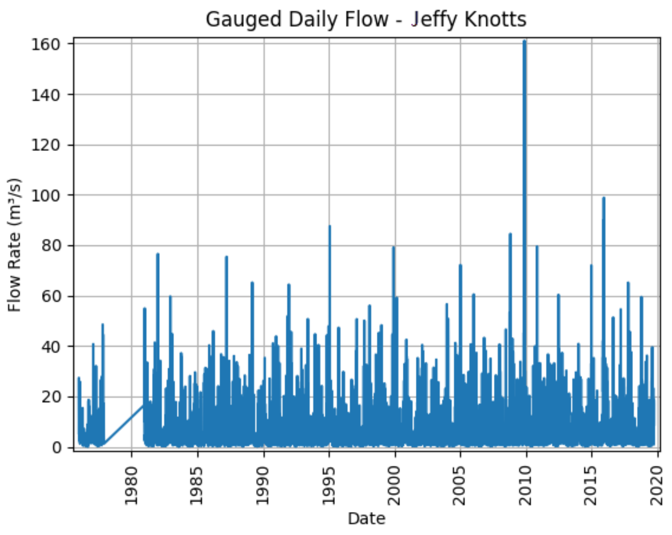


Figure 3.3: River Brathay - Jeffy Knotts

Figure 3.3 shows data very similar to the Miller Bridge House data (Figure 3.2). This is to be expected as the rivers are in the same geographic region, so the two rivers will experience much of the same weather events. Again the graph shows a period of missing data, this time between December 1st 1977 and December 30th 1980. There were several additional smaller pockets of missing data scattered throughout the time-series up until the 13th of June 1985. Taking similar precautionary measures to the previous dataset, all data preceding June 13th 1985 was removed to minimise impact to the neural network models. This leaves the dataset with a total of 12,528 entries beginning on June 13th 1985 and ending on September 30th 2019.

Colney gauging station

The final river flow time-series dataset collected is from the Colney gauging station on the river Yare (National River Flow Archive, n.d.*c*). The data collected spans 21,915 entries and runs from October 1st 1959 until September 30th 2019.

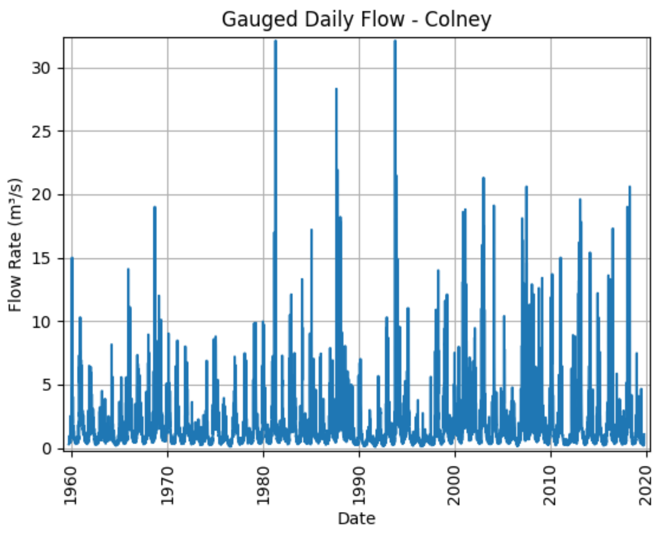


Figure 3.4: River Yare - Colney

The data shown in Figure 3.4 contains a similar cycle of peaks and troughs to the other river flow time-series datasets. This is again due to the UKs annual weather cycle. There were no periods of missing data for this dataset so the final dataset used for the implementation was the original with 21,915 entries measured between October 1st 1959 and September 30th 2019.

3.3 Algorithm process

After collecting the time-series datasets, the procedural flow of the algorithm was designed. Figure 3.5 shows the stages that the application executes when analysing the collected time-series datasets.

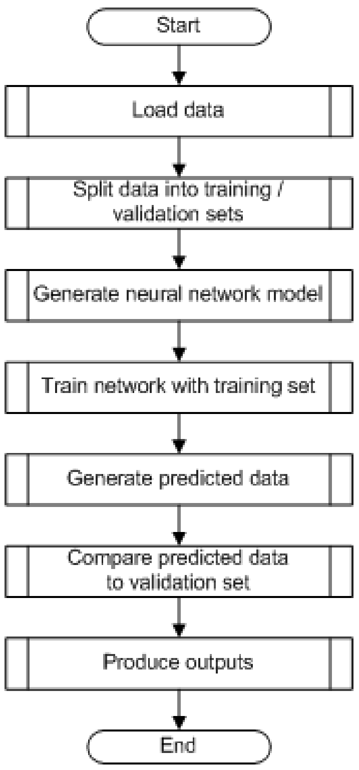

Start
Load data
Split data into training / validation sets
Generate neural network model
Train network with training set
Generate predicted data
Compare predicted data to validation set
Produce outputs
End

Figure 3.5: Algorithm Flow Chart

3.3.1 Loading data

The first stage of the algorithm is loading data. Figure 3.6 shows the code used to load in the sunspots time-series dataset.

```
time_step = []
sunspots = []

with open("State of the Art Data/Sunspots.csv") as csvfile:
    reader = csv.reader(csvfile, delimiter=',')
    next(reader) # Skip header line
    for row in reader:
        sunspots.append(float(row[2]))  # Average number of sunspots over period
        time_step.append(int(row[0]))   # index used as time series step

# Convert series lists to numpy arrays
series = np.array(sunspots)
time = np.array(time_step)
```

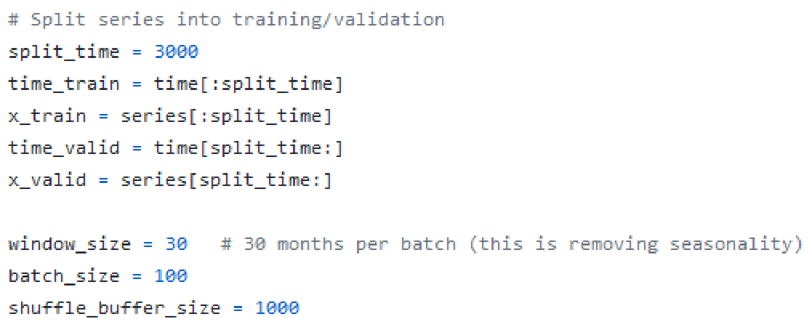
Figure 3.6: Loading Data

The csv library is first used to access `Sunspots.csv`. Once the file has been opened, a reader is created that will iterate over each row of the file and separate the data looking for a comma as a delimiter. After the first line in the file (containing only column names) is skipped by the reader each row in the file is read and the sunspots measurement column is appended to the `sunspots` list, with the csv file's row ID (numerical order) appended to the `time_step` list.

After the entire file has been read through and the two lists populated, the `sunspots` and `time_step` lists are converted to numpy arrays (`series`) and `time` respectively. These will be used as the data for the remainder of the algorithm.

3.3.2 Data pre-processing

The second stage of the algorithm is to split the data into training and validation (testing) sets. Figure 3.7 below shows the process used to split the `series` and `time` arrays into data to be used for training and testing.



```
# Split series into training/validation
split_time = 3000
time_train = time[:split_time]
x_train = series[:split_time]
time_valid = time[split_time:]
x_valid = series[split_time:]

window_size = 30   # 30 months per batch (this is removing seasonality)
batch_size = 100
shuffle_buffer_size = 1000
```

Figure 3.7: Splitting Data into Training and Validation Sets

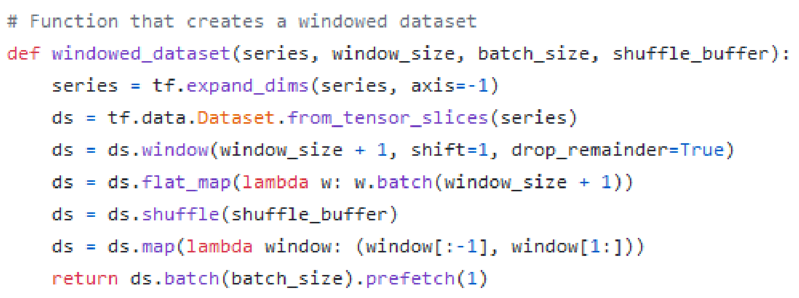
First, a time to split the dataset (`split_time`) is selected. In this case, the sunspots data consists of 3,264 individual entries, so splitting the array at index 3000 will produce a roughly 90%/10% split for training and validation.

To actually split the data, two new variables are created for training of the neural network models (`time_train` and `x_train`) and these are populated with the data up until the `split_time` from their respective arrays created earlier (`time` and `series` respectively). The same thing is done for the testing datasets in reverse, all of the data in the original arrays after the defined split time are added to `time_valid` and `x_valid`.

Some additional variable are then also assigned (`window_size`, `batch_size` and `Shuffle_buffer_size`). `window_size` will be used to determine how many data points in series will be used in each chunk for training. The model will be learning to take the previous 30 months, and predict the following month. `window_size` being reasonably small also acts to remove seasonality from the original data as the seasonal pattern in the sunspots data is made up of a roughly 11-year cycle (roughly 132 months).

`batch_size` and `shuffle_buffer_size` are picked somewhat arbitrarily. `batch_size` is the number of windows to be used simultaneously by the neural network when training, thus limiting the memory load, and using a `shuffle_buffer_size` larger than the total data points (in this case 3,000 for training) divided by the window size (in this case 30) will include all windows in the training process.

These three variables, along with the training dataset are then passed into the `windowed_dataset` function (Figure 3.8) to create the final labelled data used for training.



```python
# Function that creates a windowed dataset
def windowed_dataset(series, window_size, batch_size, shuffle_buffer):
    series = tf.expand_dims(series, axis=-1)
    ds = tf.data.Dataset.from_tensor_slices(series)
    ds = ds.window(window_size + 1, shift=1, drop_remainder=True)
    ds = ds.flat_map(lambda w: w.batch(window_size + 1))
    ds = ds.shuffle(shuffle_buffer)
    ds = ds.map(lambda window: (window[:-1], window[1:]))
    return ds.batch(batch_size).prefetch(1)
```

Figure 3.8: Creating a Windowed Dataset

This function goes through several steps to produce the final batches of data used for training. The first line simply adds an additional dimension to the series. This is to facilitate the use of LSTMs, which require a 3-dimensional input. Following this, a `Dataset` object is created from the information in the series.

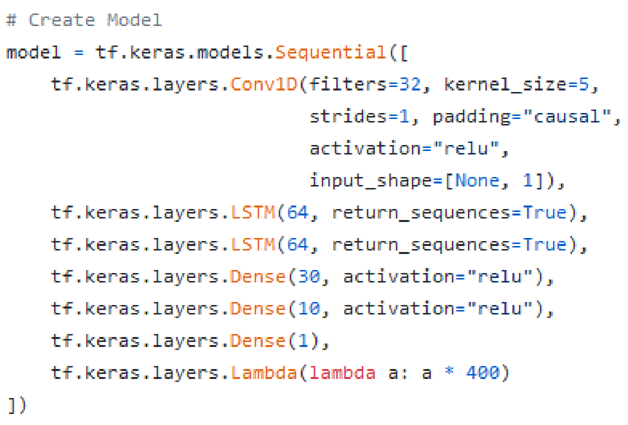
The dataset is then adjusted to be made up of windows. `window_size + 1` is used here to simultaneously create the windows and label the data. Each window will consist of `window_size` consecutive data points and the next data point will be used as the target value for the neural network. This windowed dataset is then flattened to remove the additional dimensionality added through the windowing process.

Following this, the dataset is shuffled. This produces a dataset with `shuffle_buffer` entries. In this case, `shuffle_buffer` is greater than the total number of windows in the dataset and thus, a dataset with all entries in a random order will be produced. The idea of shuffling the data is to discourage overfitting in the training process as each batch will be randomly constructed and general.

Finally, the dataset is remapped to separate the window data and their respective labels and the entire dataset is split into `batch_size` batches. This produces a final 3-dimensional dataset split into batches of 2-dimensional elements containing window data and their labels.

3.3.3 Model generation

Step 3 of the implemented algorithm is generating the neural network. Figure 3.9 below shows the construction of the neural network model as it was used for the sunspot data.



```python
# Create Model
model = tf.keras.models.Sequential([
    tf.keras.layers.Conv1D(filters=32, kernel_size=5,
                           strides=1, padding="causal",
                           activation="relu",
                           input_shape=[None, 1]),
    tf.keras.layers.LSTM(64, return_sequences=True),
    tf.keras.layers.LSTM(64, return_sequences=True),
    tf.keras.layers.Dense(30, activation="relu"),
    tf.keras.layers.Dense(10, activation="relu"),
    tf.keras.layers.Dense(1),
    tf.keras.layers.Lambda(lambda a: a * 400)
])
```

Figure 3.9: Creating A Neural Network Model

A single sequential model has been created, this will calculate the outputs of each layer one after the other, with each layer being provided one input (the output from the previous layer) and producing one output (to be fed into the next layer).

The first layer in the model is a 1-dimensional convolutional layer. The CNN is fed with a dataset split into batches of `batch_size` samples, each consisting of `window_size` timesteps, as shaped by the `windowed_dataset` function (Figure 3.8). The CNN is set to have a kernel size of 5 with no bias applied to the model. The padding is set to 'causal' to ensure that the output of any input sequence is not dependent on any future values. The activation function applied is the standard rectified linear unit (ReLU), which is defined by:

$$g(x) = max(x, 0) \tag{3.3}$$

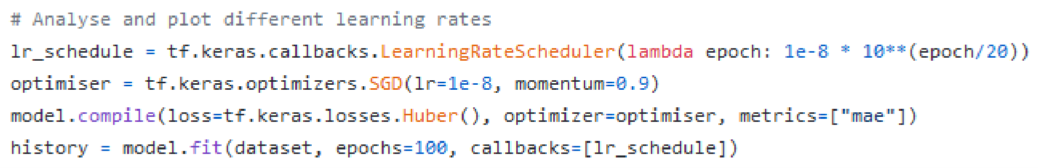
This layer is followed by two stacked LSTM layers, each comprising of 64 units with the default `activation` function `tanh` and default `recurrent_activation` function `sigmoid` to control the feedback loops. The return sequence is set to `True` to ensure the first layer provides the correct output as input into the second LSTM layer.

Finally, there are three densely connected neuron layers.

The `Lambda` layer at the end simply multiplies the output of the network by 400. This is to re-scale the results to the same range as the original data (0-400).

Finding learning rate

Before the model can be trained with the data, a good learning rate needs to be determined. This parameter tells the neural network how significantly the internal weights should be changed during training so that it can converge towards an area of minimum loss. The code used to find the ideal learning rate is shown in Figure 3.10 below.



```
# Analyse and plot different learning rates
lr_schedule = tf.keras.callbacks.LearningRateScheduler(lambda epoch: 1e-8 * 10**(epoch/20))
optimiser = tf.keras.optimizers.SGD(lr=1e-8, momentum=0.9)
model.compile(loss=tf.keras.losses.Huber(), optimizer=optimiser, metrics=["mae"])
history = model.fit(dataset, epochs=100, callbacks=[lr_schedule])
```

Figure 3.10: Finding the Optimal Model Learning Rate

The first step of finding the optimal learning rate is to create a `LearningRateScheduler`. This allows the learning rate of the model to be changed after every epoch. In this case, the learning rate begins at 1e-8 and over the course of 100 epochs, increases to 1e-3. The model is then compiled and trained. After every epoch, the model uses the `LearningRateScheduler` that was set up to dynamically adjust the learning rate to the newly calculated value. At the end of 100 epochs, the graph in Figure 3.11 can be produced.

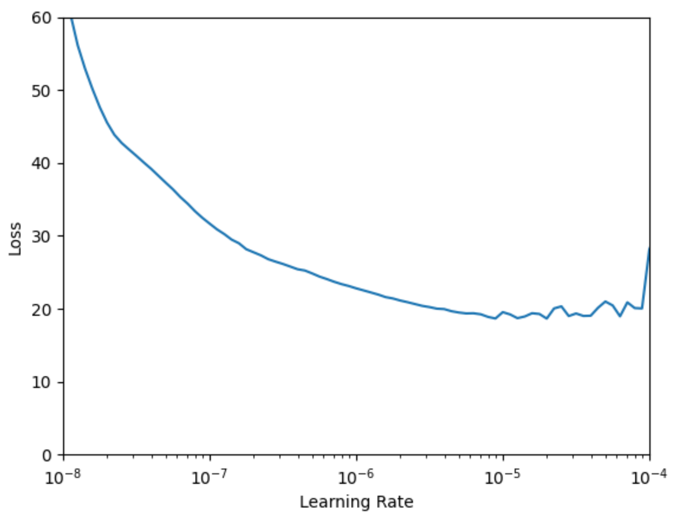


Figure 3.11: Sunspots Loss at Different Learning Rates

This graph shows that as the learning rate increases, the loss generally decreases. After the learning rate reaches roughly 1e-5, it begins to become unstable as it continues changing. 1e-5 is therefore used as the learning rate as it is the highest learning rate that will produce stable outputs.

3.3.4 Model training

Now that the model has been created and the ideal learning rate has been established, the model can be trained extensively on the training data. Figure 3.12 shows how the model is trained in the implementation.

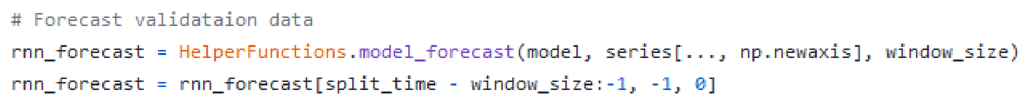
```
# Train model
optimiser = tf.keras.optimizers.SGD(lr=1e-5, momentum=0.9)
model.compile(loss=tf.keras.losses.Huber(), optimizer=optimiser, metrics=["mae"])
history = model.fit(dataset, epochs=500)
```

Figure 3.12: Training the Neural Network

The training model uses a stochastic gradient descent (SGD) optimiser which updates model parameters for every training sample introduced every epoch. This allows the model to converge reasonably well towards local minima, but also gives it a chance to "jump" to other solutions with potentially better local minima or a global minimum. The model is compiled using this optimiser and will keep track of the loss over time using the Huber loss function. Huber loss is an alternative to squared error loss and is discussed further in the results section. Additionally, the model outputs the mean average error (MAE) to the console at the end of each epoch. This was used primarily for monitoring training progress during application runtime. With the model compiled, it is finally trained on the windowed dataset for 500 epochs. The `history` variable is used to track the loss over time during training.

3.3.5 Model prediction

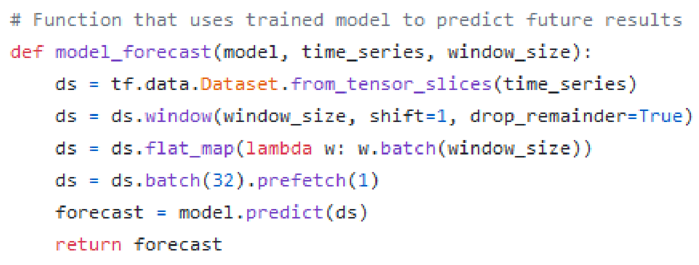
After the model was trained for the desired amount of time, it was used to forecast the datapoints contained in the validation array. Figure 3.13 shows briefly how this forcasting was carried out.



```
# Forecast validataion data
rnn_forecast = HelperFunctions.model_forecast(model, series[..., np.newaxis], window_size)
rnn_forecast = rnn_forecast[split_time - window_size:-1, -1, 0]
```

Figure 3.13: Forecasting Validation Data with the Trained Model

First the model and the entire time-series is passed into the `model_forecast` function (Figure 3.14). The output of this function is then split such that it only includes data relevant to the validation set (i.e. after `split_time`).



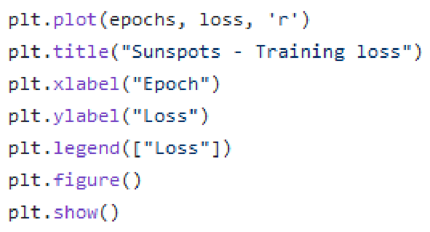
```
# Function that uses trained model to predict future results
def model_forecast(model, time_series, window_size):
    ds = tf.data.Dataset.from_tensor_slices(time_series)
    ds = ds.window(window_size, shift=1, drop_remainder=True)
    ds = ds.flat_map(lambda w: w.batch(window_size))
    ds = ds.batch(32).prefetch(1)
    forecast = model.predict(ds)
    return forecast
```

Figure 3.14: Implementation of the *model_forecast* Function

The `model_forecast` function is similar in implementation to the `windowed_dataset` function (Figure 3.8). A `Dataset` object is first created from the time series data, followed by window creation. This data is then flattened and batched together. All of these similar steps are taken because the data should be in a similar format when forecasting as when training. With the batched windowed data, the model is used to predict, for each window, the next value in the time series. This final forecasted dataset is then returned by the function.

3.3.6 Output generation

All of the output graphs were generated using the matplotlib library. Figure 3.15 shows an example of using the library to generate a graph.



```
plt.plot(epochs, loss, 'r')
plt.title("Sunspots - Training loss")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend(["Loss"])
plt.figure()
plt.show()
```

Figure 3.15: Plotting Results

The graph produced by the above code is shown in Figure 3.16. The code plots, in red, `epochs` on the x axis against `loss` on the y axis. The title of the plot is set to "Sunspots - Training loss" and has labels for both the axes. A legend has been added to show what the coloured line represents (this is more relevant with multiple lines on the same graph). Finally, the Figure is compiled, and shown on the screen.

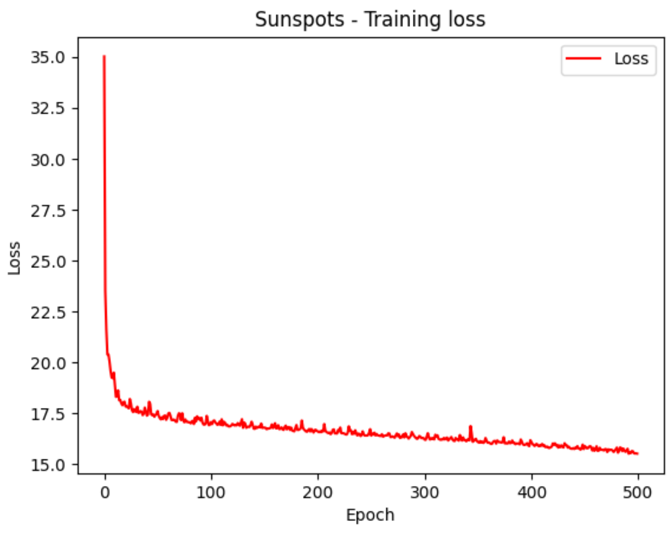


Figure 3.16: Sunspots Training Loss Graph

3.4 Summary

In this section, an overview of the technologies used, along with a description of the project goals were discussed. The process used for data collection and initial preparation shows how state-of-the-art sunspot data was gathered, along with the project's three river flow time-series datasets. Some minor pre-processing was carried out, removing missing values from the Miller Bridge House, and Jeffy Knotts gauging station datasets.

Following this, the flow of the program algorithm have been described and discussed in order, using examples where appropriate of the implementation as used in creating a benchmark from the state-of-the-art sunspot data. The steps of the algorithm discussed were:

1. Loading the data
2. Splitting the data into training and validation datasets
3. Generating the neural network model
4. Identifying the ideal learning rate of the model
5. Training the model
6. Predicting validation data using the fully trained model
7. Generating output graphs

All of the same algorithm steps were also applied to each of the river flow datasets. The next chapter takes a look at the results generated by the algorithm for each of the datasets.

The complete project implementation can be found at https://csgitlab.reading.ac.uk/vn013754/mike-final-year-project.

Chapter 4

Results

After the framework was set up to generate and train the neural network models for each of the datasets, the benchmark results were created and results of the river flow time-series datasets were gathered for comparison.

4.1 Huber loss

When identifying the optimal learning rate of the datasets and during training of the neural network models, logs of the loss over time were collected. In order to gauge the performance of the neural networks their outputs were evaluated with a loss function known as the Huber loss function (Huber, 1964), as defined by Eq. 4.1 where y is the actual measured value, $f(x)$ is the predicted value, and δ is a threshold value. For all measurements undertaken by the implemented algorithm, $\delta = 1.0$.

$$L_\delta(y, f(x)) = \begin{cases} \frac{1}{2}(y - f(x))^2 & \text{for} |y - f(x)| \leq \delta, \\ \delta\, |y - f(x)| - \frac{1}{2}\delta^2 & \text{otherwise.} \end{cases} \tag{4.1}$$

The Huber loss equation is an ideal alternative to squared error loss as values above the threshold are represented linearly, thus the loss is less sensitive to outliers.

4.2 Benchmark results

The state-of-the-art sunspots dataset was used to generate a set of benchmark results. The outputs and performance of the benchmark model will be compared with those of the gauging station models to determine whether or not the models were able to perform similarly. Throughout the overall algorithm process, three distinct measurements were calculated and graphed.

An initial measurement to determine the ideal learning rate of the model was performed. This process was discussed in detail in section 3.3.3, and by analysing Figure 3.11 it was determined that the ideal learning rate for the sunspots model was 1e-5.

Figure 3.16, shown in section 3.3.6 shows that very quickly into the training process, the loss dropped significantly to around 17.5 after only a few epochs of training time. During the remainder of the 500 epochs, the loss slowly decreased to around 15.6. Figure 4.1 shows a zoomed version of this training loss graph that only includes the final 300 epochs. This graph shows that even after the 500 epochs training time, the rate at which the loss is decreasing hasn't slowed significantly, and through further training, could still be improved.

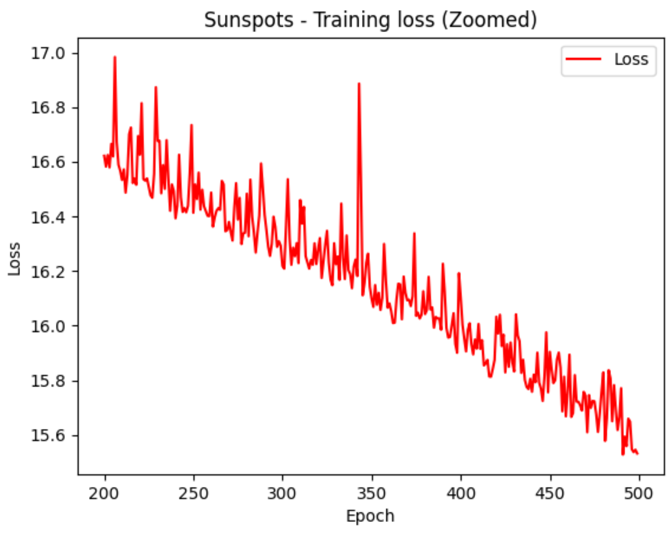


Figure 4.1: Zoomed Huber Loss During Training - Sunspots

The final graph produced through the algorithm process shows the validation data and overlays it with the model's predictions of the time-series. Figure 4.2 shows that for the state-of-the-art dataset, the benchmark model was able to easily follow the general direction of the data and wasn't significantly affected by the noise present in the measurements.

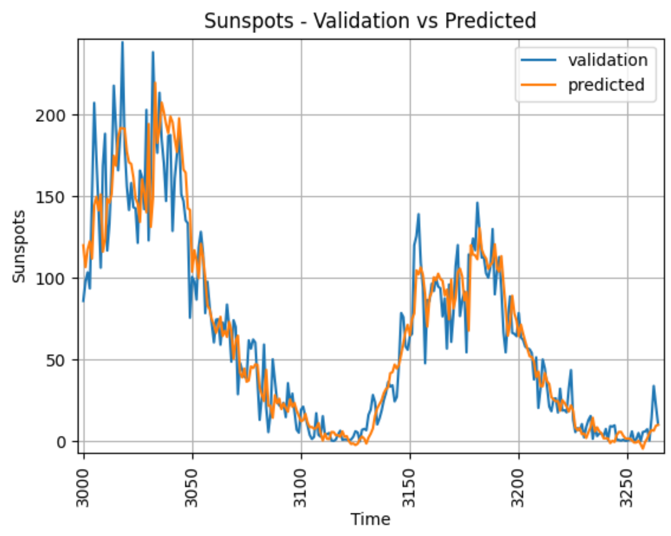


Figure 4.2: Predictions vs. Validation Data - Sunspots

4.3 River flow results

The data collected from the individual gauging stations was passed through the same algorithm as the sunspot data, some minor changes were made to the hyperparameters of the neural network models compared to the benchmark to facilitate the overall differences in the datasets.

4.3.1 Miller Bridge House gauging station

The Miller Bridge House gauging station dataset used by the program consists of 14,152 entries. With a cyclical structure lasting a year, it was decided that the `window_size` parameter would be set to 365. Additionally the `batch_size` was changed to 40. The neural network model was kept the same as the benchmark.

The first step of the training process was again to determine the optimum learning rate. Figure 4.3 shows the Huber loss for different learning rates. By analysing this graph, it was determined that a learning rate of 4e-5 was ideal for the model.

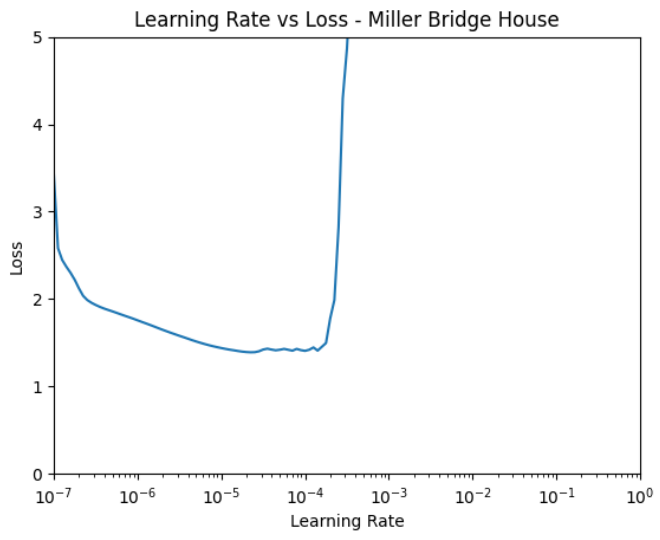


Figure 4.3: Learning Rate vs. Huber Loss - Miller Bridge House

The model was trained for 1,000 epochs, the course of which is demonstrated in Figure 4.4. As can be seen in the graph, the calculated Huber loss very rapidly declined during the early period of training and slowed significantly within the first 50 epochs. After approximately 500 epochs however, the Huber loss of the model started accelerating lower again with increased variance between epochs. This demonstrates that even after 1,000 epochs there was still plenty of room for the model to improve its performance.

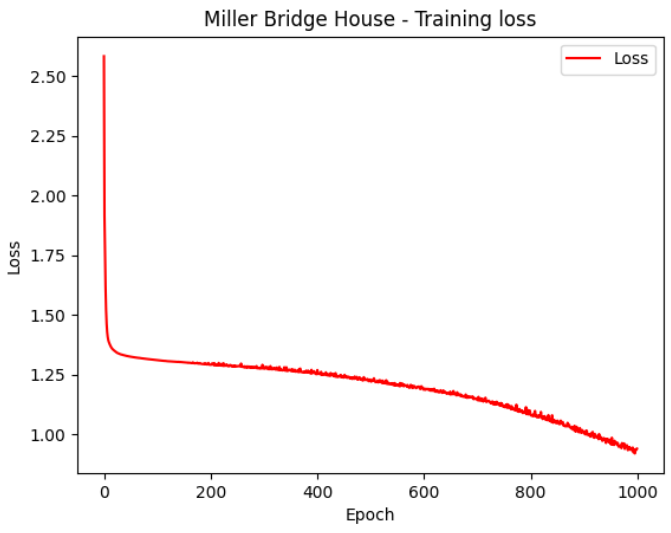


Figure 4.4: Huber Loss During Training - Miller Bridge House

Figure 4.5 shows a close up of the last 50 epochs in the training loss graph. There is still a swift downward trend even in the last 50 epochs of training reiterating that the model can continue to improve.

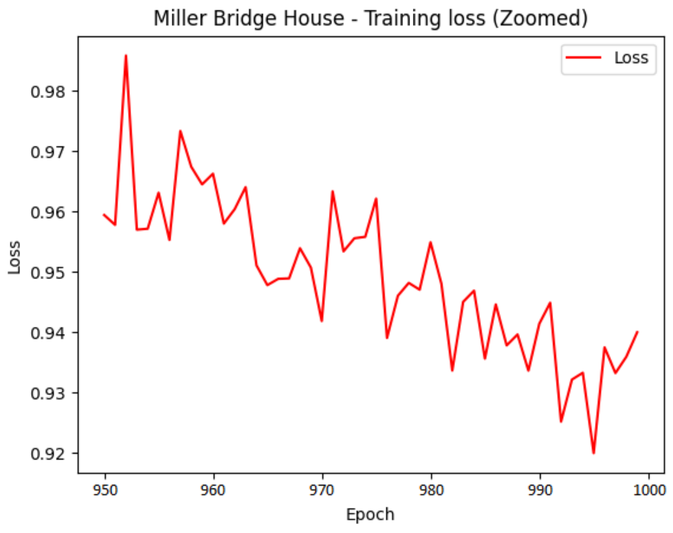


Figure 4.5: Zoomed Huber Loss During Training - Miller Bridge House

Figure 4.6 shows the validation data in blue overlayed with the predicted values in orange. While in general, the predictions are keeping up with the real measurements, there are a couple of anomalous results. At the end of 2015 and at the end of 2018, the model has predicted a negative flow rate. These values do not make a sense to have been generated as outputs, and indicate that measures should perhaps be taken to ensure that the model does not calculate negative results.

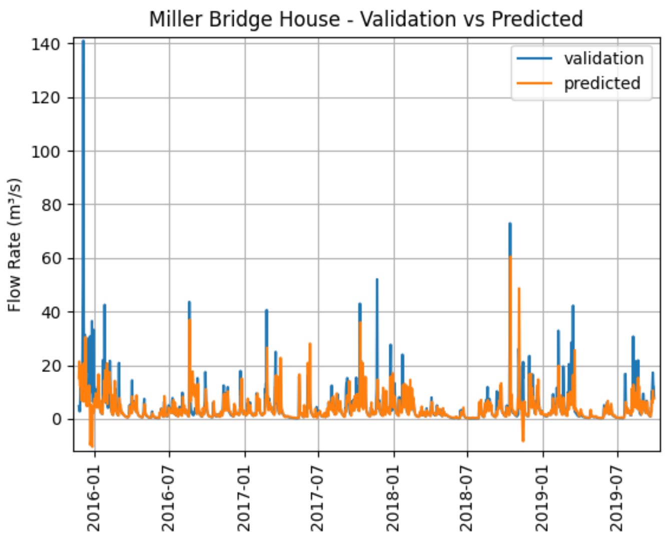


Figure 4.6: Predictions vs. Validation Data - Miller Bridge House

Finally, Figure 4.7 shows the final 6 months of predicted data. This graph demonstrates clearly that the model is capable of following the direction and predicting accurately the actual data when the interday values are not wildly fluctuating, but even with large spikes in measurements, the model recovers quickly and maintains reasonably accurate predicted values.

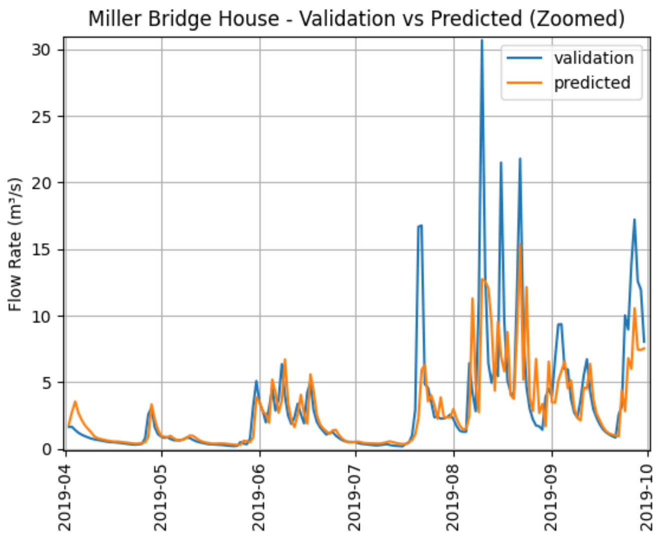


Figure 4.7: Zoomed Predictions vs. Validation Data - Miller Bridge House

4.3.2 Jeffy Knotts gauging station

The Jeffy Knotts gauging station dataset used by the program consists of 12,528 entries. As with the Miller Bridge House gauging station data, there is an annual cyclical seasonality within the time-series. For this dataset's model however, the `window_size` parameter was set to 72, with a `batch_size` of 30. This again acts to remove the seasonality from the dataset and aid the training process.

An additional modification made to the model construction was the removal of the 30 unit Dense layer, and the second LSTM layer was modified to have only 30 units rather than the original 64. This was discovered to provide slightly improved training conditions for this dataset. The remaining steps in the algorithm remained the same, and the initial learning rate was identified as 1e-5 using Figure 4.8.

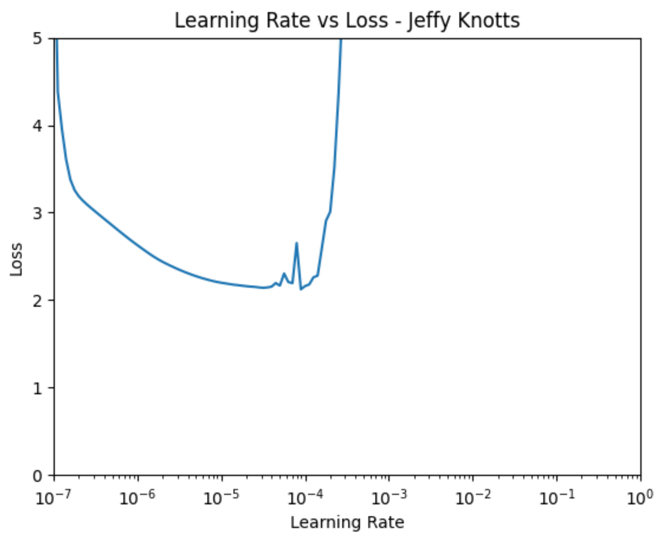


Figure 4.8: Learning Rate vs. Huber Loss - Jeffy Knotts

The loss reduction during training of this model as shown in Figure 4.9 started with a sharp decline during the early training epochs, then maintained a steady downward trajectory throughout the remaining epochs. Again, this implies that the model was capable of being trained for a longer period and that it would continue to improve after 1,000 epochs.

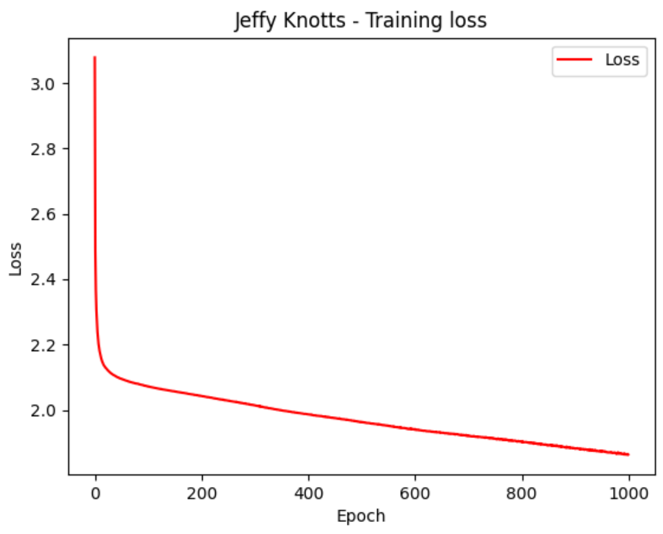


Figure 4.9: Huber Loss During Training - Jeffy Knotts

The zoomed loss graph in Figure 4.10 shows the same loss reduction over the last 50 epochs. The actual decline over these final few epochs is not exceptionally fast, though the steady improvement is still clearly continuing and the model would likely have facilitated further improvement with more training epochs.

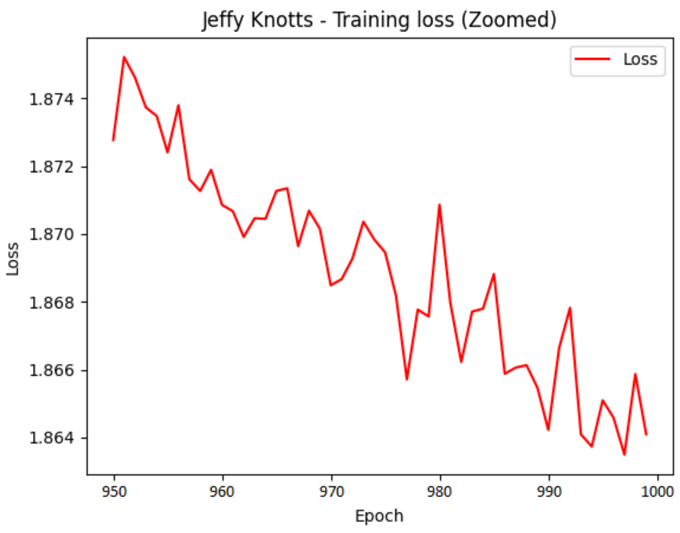


Figure 4.10: Zoomed Huber Loss During Training - Jeffy Knotts

Again, an output of the validation data in blue overlayed with the forecasted results in orange is shown in Figure 4.11. The model appears to have performed rather poorly at following the erratic movement of the measurements, often not reaching even half way up the individual peaks.

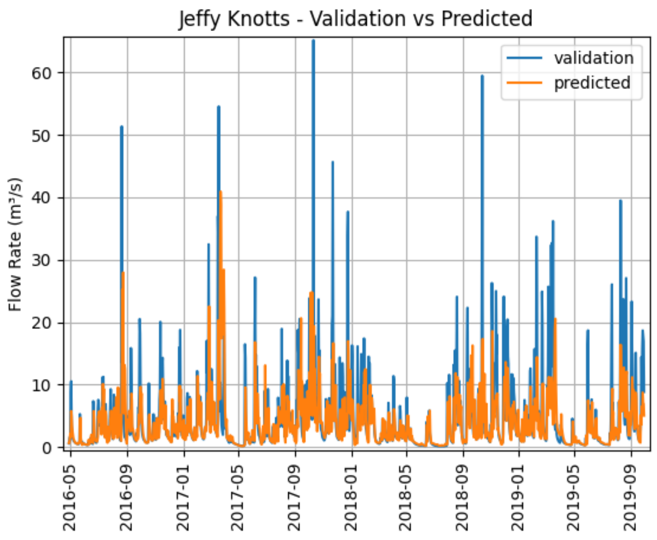


Figure 4.11: Predictions vs. Validation Data - Jeffy Knotts

The closeup of the graph in Figure 4.12 shows that the Jeffy Knotts neural network model is quite good at making predictions when the measurements are relatively calm, but struggles when significant spikes are introduced.

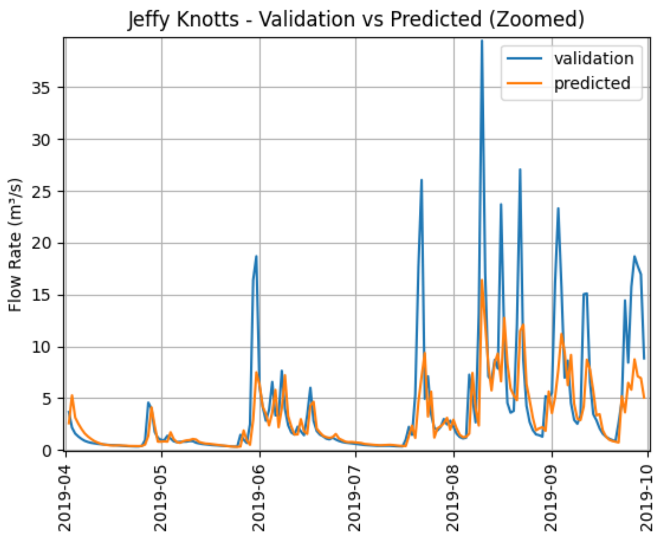


Figure 4.12: Zoomed Predictions vs. Validation Data - Jeffy Knotts

4.3.3 Colney gauging station

The final model produced was for the Colney gauging station dataset. This dataset consists of 21,915 entries and as with the other river flow time-series, follows the same annual cycle of peaks and troughs. For this model, a `window_size` of 72 was used, along with a `batch_size` of 40. The construction of the model was the same as the benchmark model.

Figure 4.13 shows the Huber loss for different learning rates of the model. Using this graph, a learning rate of 1e-3 was identified as ideal.

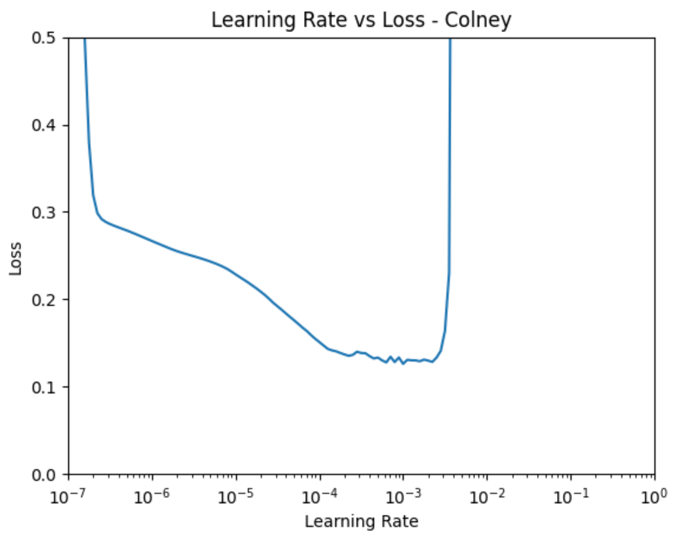


Figure 4.13: Learning Rate vs. Huber Loss - Colney

The loss reduction of this model, shown in Figure 4.14 follows a peculiar shape. It started off as usual, quickly dropping in the first few epochs. Then the loss reduction began accelerating again until about half way through the training process, where the decline in loss became less rapid again. As with the other models, this model was still reducing its loss after the full 1,000 epochs of training time and could likely have continued to improve with more training time.

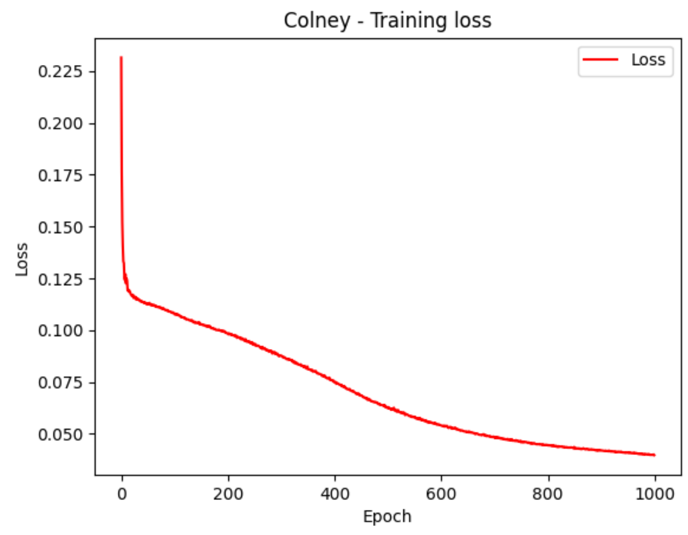


Figure 4.14: Huber Loss During Training - Colney

Towards the end of the model training period, seen in Figure 4.15, there is still a 2% reduction in the loss value over the final 50 epochs.

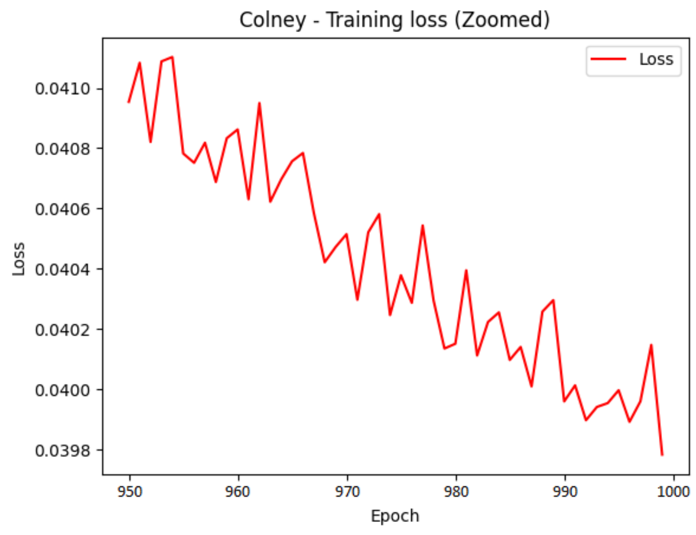


Figure 4.15: Zoomed Huber Loss During Training - Colney

This model performed exceptionally well when predicting the validation data. Figure 4.16 shows that the predictions closely follow the annual peaks and troughs in the data measurements, and even the spikes are reasonably well handled by the model.

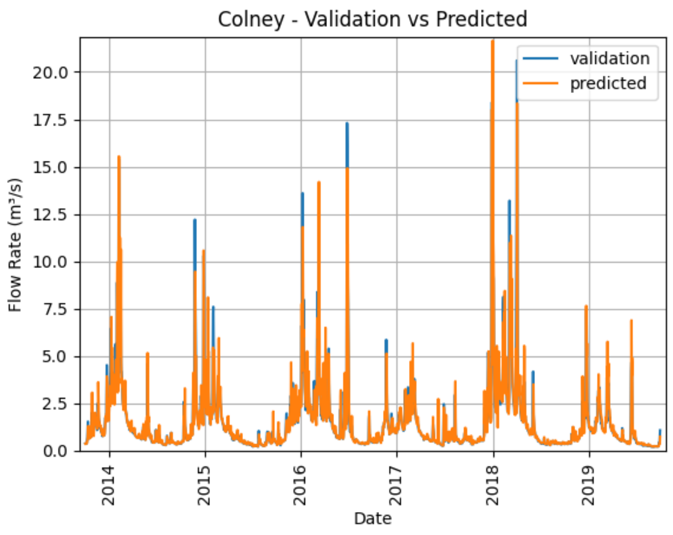


Figure 4.16: Predictions vs. Validation Data - Colney

The zoomed graph in Figure 4.17 show clearly how closely the predictions actually match the measured data, following the calm fluctuations until a sharp spike. A small overestimate after the spike was predicted by the model, but a couple of time-steps further along, the predictions were back on track with the measurements.

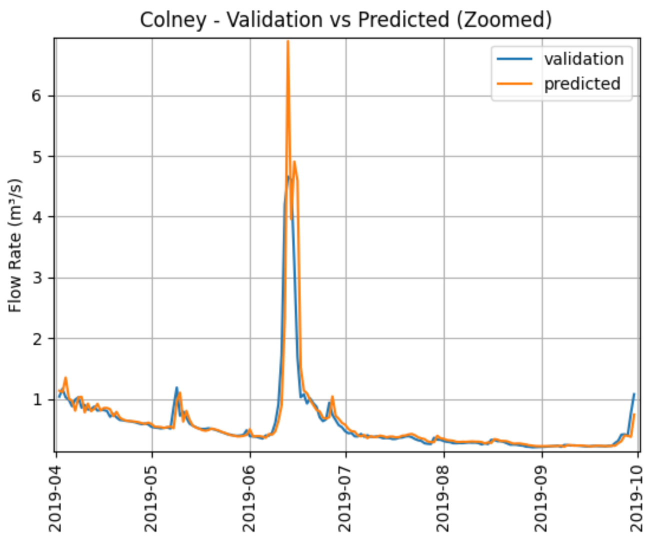


Figure 4.17: Zoomed Predictions vs. Validation Data - Colney

4.4 Summary

Using the state-of-the-art sunspot time-series data, a set of benchmark outputs were calculated using the neural network model. The model was able to recognise patterns in the sunspot dataset and learned to predict with a reasonably accuracy, the value following a 30 time-step window. Each of the river flow datasets was able to be applied to a similar model in the same way. All of the saw a significant improvement after training for 1,000 epochs, and were all capable of improving further after the end of their training periods. Predictions of the validation data were made by each of the models with reasonable success, and with the exception of two impossible negative flow rate calculations from the Miller Bridge House model, the predictions made were all within a reasonable range of the actual values.

Comparisons of performance of the benchmark model and the river flow models will be explored in the following chapter.

Chapter 5

Discussion and Analysis

5.1 Relative Loss

By dividing the Huber loss of the model by the average value of the data contained in a time-series, a relative loss value can be calculated. The relative loss for each of the time-series neural network models produced in this project is shown in Table 5.1

Model	Relative Loss
Sunspots	18.95
Miller Bridge House	21.46
Jeffy Knotts	36.98
Colney	2.58

Table 5.1: Relative Loss for each Model

5.2 Comparison of benchmark and river flow results

With all of the results collected and compiled, the performance of the river flow time-series models can be compared to that of the benchmark model.

5.2.1 Jeffy Knotts

Starting with the worst performing of the three models, the Jeffy Knotts model. This model used a modified neural network architecture, removing one of the Dense neural network layers, and reducing the capacity of the second LSTM layer. While these changes had a positive impact on the training conditions of the model for the dataset, this impact did not offset the seeming incapability for the model to sufficiently learn the data.

Compared to the sunspot data with a relative loss of 18.95, the Jeffy Knotts model's relative loss of 36.98 is substantially worse. This was evident in the model's prediction outputs (Figure 4.11, Figure 4.12) where the model was not able to accurately predict data revolving around the measurement spikes.

The benchmark predictions (Figure 4.2) were not completely capable of following all of the movement of the observations, but the model was more than capable of adjusting to the fast changing datapoints nonetheless.

5.2.2 Miller Bridge House

The Miller Bridge House model was better performing than the Jeffy Knotts model. With a relative loss value of 21.46, this model more closely matched the benchmark model's 18.95. This, again, is evident when comparing the model's predicted output performance (Figure 4.6, Figure 4.7) to those of the benchmark model. The Miller Bridge House model was able to closely map a generous portion of the sudden spikes in the data. Significantly more so than the Jeffy Knotts model was capable of.

The strange negative outputs from the model could potentially have been negated by using a ReLU activation function on the final Dense layer of the network. Though, it is likely that with only two of the values being predicted as negative, this likely didn't significantly impact the overall relative accuracy of the model.

5.2.3 Colney

The final model to compare is for the Colney gauging station. The performance of this model was generally outstanding. With a relative loss of only 2.58, it was clear in the predicted output graphs (Figure 4.16, Figure 4.17) that the performance of the neural network far exceeded either of the other two river flow models, and even outclassed the benchmark results.

The graphs show that the model was able to closely map the validation data all the way through, and could just as effectively predict the areas of sharp movement as it could the more calm periods of data. In direct comparison to this model, the benchmark predictions seem much more passive and more akin to a smoothing function than accurate predictions.

5.3 Significance of results

The main objective introduced by this project was to build models for use with time-series river flow data, that was capable of producing predictions similar in accuracy to those produced from a state-of-the-art benchmark model. While the Jeffy Knotts gauging station data was unable to produce a model capable of accurate predictions, the Miller Bridge House model generated predictions with a comparable accuracy to the benchmark. Furthering this, the Colney gauging station data was able to be used by an almost identical model to the benchmark to calculate highly accurate predictions.

In addition to demonstrating that such a model can perform similarly to a state-of-the-art example, the ability for the models to predict values with river flow measurements alone and no extra information regarding rainfall or water runoff shows that for some rivers, it may be possible to extrapolate future information just with prior observations of the river's behaviour.

5.4 Summary

This chapter took a look at some comparisons of the performance of the river flow neural network models against the benchmark models. It was concluded that the worst performing model was the Jeffy Knotts gauging station model, performing significantly worse than the benchmark. The Miller Bridge House gauging station model performed only slightly worse than the benchmark model when predicting future values, and the best performing model was the Colney gauging station model, which dramatically outperformed the benchmark resutls.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

The main objective of the project was to develop a neural network model capable of the time-series analysis of river flow data. After an in-depth analysis of existing solutions, the conclusion was met that an ideal solution would come in the form of a CNN-LSTM model. A benchmark model was created using a state-of-the-art dataset to predict monthly sunspot observations, and results were collected from the model's outputs. After collecting river flow data from three gauging stations in the UK: Colney, Miller Bridge House, and Jeffy Knotts, similar models to the benchmark were then created and predictions were made for each of the datasets.
The final outputs of each of the models showed that it was possible to attain results similar to, or better than the benchmark predictions, using river flow data. The Jeffy Knotts model was the worst performing model, and was unable to predict values as well as the benchmark. The Miller Bridge House model was able to produce results at a level similar to the benchmark sunspots model, and the Colney model was able to predict values with a relatively high accuracy, outclassing the benchmark model.

The models' performances have met the requirements set in the initial objective and it has been demonstrated that for some geographical regions, river flow analysis is able to be performed even without additional information about rainfall or water runoff.

6.2 Future Work

There is significant scope for the furthering of this research in the future. The first step may come in the form of longer training times of the data. The models used for this project were only trained for a maximum of 1,000 epochs, but still showed signs of significant improvement after the end of the training period. In a similar vein, future research can look at further experimenting with the hyperparameters of the models. Basic hyperparameters were tuned using trial and error here, but the introduction of more advanced methods such as the use of genetic algorithms could prove fruitful in improving the predictive accuracy of the models.

Extensions of the algorithm could be made to include anomaly detection calculations that can identify flood events from the data and predictions, and with the possible introduction of additional data inputs, such as rainfall, the models could prove to be effective advanced warning and risk management tools.

Chapter 7

Reflection

I thoroughly enjoyed the process of researching, planning, and developing this project. The most rewarding parts of the assignment were definitely in learning how to use new tools such as Keras, TensorFlow, and NumPy to produce real predictions of real world data. I was able to further my confidence using Python as well in the process of developing the project. Writing up the report using LaTeX was an enjoyable experience, I had never written a report with the tool before and overcoming the learning curve was a bit of a hurdle. In the end though, I quite enjoy using LaTeX and, to me, the results it produces are favourable over other word processors.

I think that I managed to complete all of my initially planned objectives as they were laid out in the PID, but was unable to successfully implement some suggested extensions to the code, including an autoencoder for anomaly detection. I think the thing I had the most problem with during the project was time management. I wasn't able to follow the original schedule of the implementation as it was laid out in the PID, and I think I should have allocated more time to the initial development of the project.

Obviously working during COVID has had an additional impact on everyone, but I found that the lack of a change in environment throughout the past year had a negative impact on my overall productivity and motivation.

All in all, I consider the completion of this project a positive experience, and I look fondly towards the future.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y. and Zheng, X. (2015), 'TensorFlow: Large-scale machine learning on heterogeneous systems'. Software available from tensorflow.org.
URL: *http://tensorflow.org/*

Adebiyi, A. A., Adewumi, A. O. and Ayo, C. K. (2014), 'Comparison of arima and artificial neural networks models for stock price prediction', *Journal of Applied Mathematics* **2014**(614342).
URL: *https://doi.org/10.1155/2014/614342*

Box, G. E., Jenkins, G. M., Reinsel, G. C. and Ljung, G. M. (2015), *Time series analysis: forecasting and control*, John Wiley & Sons.

Campolo, M., Andreussi, P. and Soldati, A. (1999), 'River flood forecasting with a neural network model', *Water Resources Research* **35**(4), 1191–1197.
URL: *https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/1998WR900086*

CAMPOLO, M., SOLDATI, A. and ANDREUSSI, P. (2003), 'Artificial neural network approach to flood forecasting in the river arno', *Hydrological Sciences Journal* **48**(3), 381–398.
URL: *https://doi.org/10.1623/hysj.48.3.381.45286*

Chen, L. and Lai, X. (2011), Comparison between arima and ann models used in short-term wind speed forecasting, *in* '2011 Asia-Pacific Power and Energy Engineering Conference', pp. 1–4.
URL: *https://ieeexplore.ieee.org/document/5748446*

Chollet, F. et al. (2015), 'Keras', `https://keras.io`.

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C. and Oliphant, T. E. (2020), 'Array programming with NumPy', *Nature* **585**(7825), 357–362.
URL: *https://doi.org/10.1038/s41586-020-2649-2*

Huber, P. J. (1964), 'Robust Estimation of a Location Parameter', *The Annals of Mathematical Statistics* **35**(1), 73 – 101.
URL: *https://doi.org/10.1214/aoms/1177703732*

Hunter, J. D. (2007), 'Matplotlib: A 2d graphics environment', *Computing in Science & Engineering* **9**(3), 90–95.

Le, X.-H., Ho, H. V., Lee, G. and Jung, S. (2019), 'Application of long short-term memory (lstm) neural network for flood forecasting', *Water* **11**(7).
URL: *https://www.mdpi.com/2073-4441/11/7/1387*

Li, Z., Kiaghadi, A. and Dawson, C. (2020), 'Exploring the best sequence lstm modeling architecture for flood prediction', *Neural Computing and Applications* pp. 1–10.

Lopez-Caraballo, C. H., Salfate, I., Lazzus, J. A., Rojas, P., Rivera, M. and Palma-Chilla, L. (2016), 'Mackey-glass noisy chaotic time series prediction by a swarm-optimized neural network', *Journal of Physics: Conference Series* **720**(012002).

Lu, W., Li, J., Li, Y., Sun, A. and Wang, J. (2020), 'A cnn-lstm-based model to forecast stock prices', *Complexity* **2020**(6622927), 1–10.
URL: *https://doi.org/10.1155/2020/6622927*

Mackey, M. and Glass, L. (1977), 'Oscillation and chaos in physiological control systems', *Science* **197**(4300), 287–289.
URL: *https://science.sciencemag.org/content/197/4300/287*

National River Flow Archive (n.d.*a*), 'Gauged daily flow - brathay at jeffy knotts'. (Accessed 2021-04-20).
URL: *https://nrfa.ceh.ac.uk/data/station/meanflow/73014*

National River Flow Archive (n.d.*b*), 'Gauged daily flow - rothay at miller bridge house'. (Accessed 2021-04-20).
URL: *https://nrfa.ceh.ac.uk/data/station/info/73013*

National River Flow Archive (n.d.*c*), 'Gauged daily flow - yare at colney'. (Accessed 2021-04-20).
URL: *https://nrfa.ceh.ac.uk/data/station/meanflow/34001*

National River Flow Archive (n.d.*d*), 'National river flow archive - home'.
URL: *https://nrfa.ceh.ac.uk/*

Oancea, B. and Ciucu, T. C. (2014), 'Time series forecasting using neural networks'.
URL: *https://arxiv.org/abs/1401.1333*

Pala, Z. and Atici, R. (2019), 'Forecasting sunspot time series using deep learning methods', *Solar Physics* **294**.

Python (2020), 'Python 3.8.6'.
URL: *https://www.python.org/downloads/release/python-386/*

Python (n.d.), 'Python csv'.
URL: *https://docs.python.org/3/library/csv.html*

SILSO data/image, Royal Observatory of Belgium, Brussels (n.d.), 'Monthly mean total sunspot number, from 1749/01/01 to 2017/08/31'. Database from SIDC - Solar Influences Data Analysis Center - the solar physics research department of the Royal Observatory of Belgium. (Accessed 2021-04-20).
URL: *https://www.kaggle.com/robervalt/sunspots*

Widiasari, I. R., Nugoho, L. E., Widyawan and Efendi, R. (2018), Context-based hydrology time series data for a flood prediction model using lstm, *in* '2018 5th International Conference on Information Technology, Computer, and Electrical Engineering (ICITACEE)', pp. 385–390.

Zhang, G. (2003), 'Time series forecasting using a hybrid arima and neural network model', *Neurocomputing* **50**, 159–175.
URL: *https://www.sciencedirect.com/science/article/pii/S0925231201007020*