CAPSTONE PROJECT BY GROUP K

A COMPARISON OF UNI- AND MULTIVARIATE CASES OF SHALLOW AND DEEP NEURAL NETWORKS FOR SHORT-TERM ELECTRICAL LOAD FORECASTING

Doug Smithers (z5135363), Tom Bernstein (z3478832), Daniel Sartor (z5350306).

School of Mathematics and Statistics
UNSW Sydney

April 2024

Submitted in partial fulfilment of the requirements of
the capstone course ZZSC9020

Abstract

Forecasting electrical load is critical for electricity generators and distributors to efficiently dispatch the load, schedule energy transfers, and plan contingencies. Accurate forecasting can significantly lower operational costs, prevent outages, and reduce emissions. Short-term load forecasting (STLF), over a 1-24 hour period, is imperative in managing operations, and STLF will be the focus of this study. Many techniques have been applied to this task. However, the application of neural networks (NNs) is salient, as they do not require the estimation of a complex load model and, crucially, have shown superior forecast accuracy compared to other methods. Since their first use in the late 1980s, advances in NN architectures have offered improved performance and prediction accuracy. This study will compare a simple feed-forward Muli-Layer Perceptron (MLP) NN to a more complex architecture in the Bi-Directional Long Short-Term Memory (BD-LSTM) network to assess the performance gains that come with these advances. In addition, a univariate model will be compared to a multivariate model for each architecture to assess the impact of including additional features. Finally, the model depth of each architecture will be increased to see if performance gains come from allowing the models to find more complex relationships between attributes. While increasing model complexity may come with performance gains, it will also come with increased computational load, which will also be examined. The model results will be compared to predictions by a model supplied by the AEMO Market Management System, which will be used as a baseline for performance. The results from this study have shown that the increased training time required for the complex models prevented them from outperforming the simple MLPs in the timeframe of this project, and more time must be given to training to come to a definitive conclusion. Multivariate models were shown to outperform their univariate equivalents on greater prediction horizons, with the added attributes providing more information for the models to learn relationships. The results showed no clear benefit from increasing model depth. Finally, the results could not match those from the baseline model, and more testing is needed for the applied models to be competitive.

Contents

Chapter 1 Introduction 1

Chapter 2 Literature Review 3
2.1 Pre-Processing and Feature Engineering . . . 3
2.2 Neural Networks . . . 4
2.2.1 Hyperparameter Tuning . . . 4
2.2.2 Simple Networks . . . 5
2.2.3 Deep Networks . . . 5
2.3 Performance Measures . . . 9

Chapter 3 Material and Methods 10
3.1 Software . . . 10
3.2 Description of the Data . . . 11
3.3 Pre-processing Steps . . . 13
3.3.1 Read Data . . . 13
3.3.2 Merge Datasets . . . 13
3.3.3 Impute Missing Values . . . 13
3.3.4 Splitting Timestamp Feature . . . 14
3.3.5 Calculating Daylight . . . 14
3.3.6 Calculating Public Holidays . . . 14
3.3.7 Transforming Periodic Features . . . 14
3.3.8 Writing Feature Set to MongoDB . . . 14
3.3.9 One-Hot Encoding . . . 14
3.3.10 Generate Univariate Inputs . . . 14
3.3.11 Data Scaling . . . 15
3.3.12 Train, Validation, and Test Splitting . . . 15
3.3.13 Feature Reduction . . . 15
3.4 Assumptions . . . 16
3.5 Modelling Methods . . . 17
3.5.1 Model Design . . . 17
3.5.2 Hyperparameter Tuning . . . 19
3.5.3 Model Training . . . 22

Chapter 4 Exploratory Data Analysis 24
4.1 Total Demand Distributions . . . 25
4.2 Temperature Distributions . . . 26
4.3 Correlation Scatter . . . 27
4.4 Periodic Feature Box Plots . . . 28
4.5 Boolean Feature Box Plots . . . 29

Chapter 5 Analysis and Results 31
5.1 Result Summary 31
5.2 Learning Curve Analysis 37

Chapter 6 Discussion 40

Chapter 7 Conclusion and Further Issues 43

References 44

Appendices 47
A.1 Code 47
A.2 Data 47
A.3 Pre-processing Pipeline 49

Chapter 1

Introduction

Forecasting electrical load, typically performed over short- (1-24 hours), medium- (1 day to several months), and long-term prediction horizons (more than a full year ahead) [1], is critical for electricity generators and distributors [2][3]. These stakeholders are particularly reliant on short-term load forecasts (STLF), which are essential for daily operations, and accurate short-term forecasting is required to enable efficient load dispatching, energy transfer scheduling, and contingency planning and has the potential to significantly lower operational costs, prevent outages, improve safety and reduce emissions [4][5]. Therefore, STLF will be the focus of this study.

Load forecasting techniques, both quantitative and qualitative, are typically classified into three major groups: traditional forecasting, modified traditional forecasting, and soft computing techniques [6]. Neural networks (NN), classified as a soft computing technique, have shown to have significant advantages over other methods because they do not require a load model, are not reliant on a functional form of a forecast model, and, importantly, can produce higher forecast accuracy [2]. The advantages presented by utilising NNs for STLF, along with the data science expertise of the team, have led to the decision to focus on this class of models for this project.

Models may be classified as univariate, taking only previous load observations as inputs, or multivariate (i.e. causal), where the load is modelled as a function of other exogenous variables, such as weather or social variables [7]. Demand on the electricity grid strongly correlates to the weather, with variables such as temperature, relative humidity, dew point, dry bulb temperature, wind speed and cloud cover commonly used [8]. However, several models have produced high accuracy using only temperature, as these weather attributes have high multicollinearity [9]. For short-term load forecasting, as weather variables change smoothly over time, univariate models are generally sufficient, with multivariate models often considered impractical due to their increased computational load [10]. The intermittent nature of renewable energy feeding into the grid, particularly wind and solar, is a challenge when producing an STLF [11]. Therefore, wind and solar radiance observations could influence the model's effectiveness. This project is an excellent opportunity to understand the impact of adding exogenous features for STLF models.

Along with a ten-year load dataset across four Australian states, temperature observations over the same period are available. Temperature varies dramatically across each state, and data from only a single station in each state is available. The absence of data from other locations is a potential limitation of this project.

However, the relative change in temperature across the day may be sufficiently well correlated with weather stations across each state.

Numerous other features can be engineered from the available data, which may add value to the models. For example, the classification of work or non-work days and an attribute for the hour of the day may prove useful. In addition, there is a strong correlation between load values for each time step and for the prior hour or the same hour in the previous week [8], and these values can be used as input.

Simple feed-forward NNs have been used for time-series forecasting since the late 1980s [12], with many advancements made since then. This study will offer an opportunity to examine the advantages of these recent developments. This paper will compare a set of feed-forward NNs to a set of more complex bi-directional long short-term memory (BD-LSTM) models. The LSTM model is a variant of the Recurrent NN (RNN), which uses recurrent connections to a context layer, allowing the network to have memory of previous inputs. While traditional RNNs suffer from two significant issues, exploding and vanishing gradients, LSTM networks are a modification of RNNs, employing memory cells to control the gradient flow, allowing them to overcome these problems. The BD-LSTM extends this concept further and uses two separate context layers, allowing information to flow backward and forward at each time step [9].

For each of these architectures, we will compare shallow networks to deeper networks and univariate models to multivariate, giving eight models in total. The aim will be to determine the gains in prediction accuracy that come with increasing model depth and the number of input features. While increased complexity may improve performance, it will also add computational load and, consequently, extend the required training time. These aspects will also be examined here. We will compare the results of each model to a set of predictions from a model created by the AEMO Market Management System. While no specific details on this model were supplied, the results from this model will serve as a baseline to measure predictive performance.

The rest of the paper is organised as follows. Chapter 2 presents a literature review of previous work. Chapter 3 details the materials and methods used. Chapter 4 details the exploratory data analysis. Chapter 5 presents the analysis and results. Chapter 6 provides a discussion of results, and Chapter 7 concludes the study and recommends further actions.

Chapter 2

Literature Review

2.1 Pre-Processing and Feature Engineering

How features are coded influences an NN's ability to map inputs to outputs and affects the way it learns [13]. Two main properties have an impact. The first is the adjacency property, which dictates that adjacent values in the original space should be represented by adjacent values in the code space. The second is periodicity, which says that the starting and ending values in the original space should have similar values in the code space. As an example, [14] used a polar coordinate system to represent the hour-of-day parameter using Equation 2.1.1, where h is the new feature vector corresponding to hour of the day, t is the hour of the day (from 1 to 24) and c is the total number of hours in the day. For any feature engineering for this project, these principles will be adhered to, where applicable.

$$h = \sin\left(\frac{2\pi t}{c}\right) \tag{2.1.1}$$

Four components affect a time series: trends, cyclical variations, seasonal variations and random variation. A comparison of several artificial intelligence-based STLF techniques [8] identified additional features that could be engineered from input data that contains only hourly temperature, similar to the data supplied for this project. As the load heavily correlates with temperature, it is subject to seasonal variations. Seasonally clustering the observations was shown to improve model performance in this study.

Additionally, the load profile over each hour through each day of the week, which directly relates to human activity, was also examined in [8].[1] [8] showed a repeating pattern of high use on weekends. This pattern will vary greatly depending on the part of the world from which and the season during which the data are sourced. It is important to consider this in our project, and it may be beneficial to include an attribute classifying the day of the week or a classification of working days. It may also be beneficial to consider non-typical days, such as special events or public holidays. Further, [8] detailed the high autocorrelation with previous load data and noted that previous observations (e.g., the load observation for the previous hour or the same hour in the previous week) are useful predictors.

[1] This relationship appears in our data and is presented in Section 4

2.2 Neural Networks

When comparing NN architectures, deep learning techniques have been shown to have advantages over shallow techniques, with superior generalisation and accuracy, but often at the expense of computational cost due to their complex network structure [5]. Of the many deep learning techniques applied across the vast spectrum of data modelling, Long short-term memory (LSTM), a type of Recurrent NN (RNN) and Gated Recurrent Unit (GRU) have gained popularity for time-series problems due to their efficient feature extraction and high prediction performances [14]. Of the variants of LSTM architectures, bi-directional LSTM networks have been compared favourably to other methods [9][15]. Another promising, although lesser studied, technique is the One-Dimensional Convolutional Neural Network (1D-CNN), which alleviates the issue of long RNN training times through the use of causal convolutional layers, with no recurrent connections [14].

2.2.1 Hyperparameter Tuning

The authors in [8] found that the predictive performance of a neural network was heavily dependent on the chosen activation functions and training algorithms. As there is no general rule for choosing these parameters, it will be essential to ensure that they are well-researched for each of the applied model architectures, and experimentation will be required to achieve the optimal result.

A comprehensive guide to hyperparameter (learning rate, batch size, momentum and weight decay) tuning is given in [16], which pays particular attention to the examination of training validation and test loss curves for evidence of under- or over-fitting. The goal of training should be to obtain maximum performance while minimising computation time. The study showed that optimal architecture and hyperparameter settings could be determined in the early stages of training, saving the need for a grid or random search, which is computationally expensive. The article implements the techniques of Cyclical Learning Rates (CLR) and Learning Rate Range Testing (LRRT), first introduced in [17] and then revised in [18].

CLR involves specifying a lower and upper bound for the learning rate and a step size. For each cycle, the learning rate is linearly increased from the lower to the upper bound and then linearly back down again. The LRRT involves starting training with a small learning rate and slowly increasing it linearly in a pre-training run. This process allows for determining how well the network trains over a range of learning rates to find the maximum learning rate that provides stable training. This value will be used as the upper bound of the CLR. The minimum learning rate is often chosen to be a factor of 3 to 4 times less than the maximum. Using large learning rates with CLR allows the network to reach super-convergence [19] (i.e. minimisation of loss over a few iterations). Further, [18] propose a modification of CLR, dubbed *1cycle*, where one cycle is smaller than the total number of epochs, allowing the learning rate to decrease below the initial rate for the final iterations.

The author in [18] showed that, along with large learning rates, other regularisation techniques, such as small batch sizes, weight decay and dropout, must be balanced for each dataset and architecture. In particular, small weight decay values (below 10^{-4}) must be used in concert with large learning rates. The weight decay value should remain constant throughout training. The article suggests using large batch sizes, which allow large learning rates when using *1cycle*. Larger momentum

values will speed up training and help the network escape saddle points but can impact final convergence. A test of values between 0.9 and 0.99 should be performed, with the best value set as a constant.

2.2.2 Simple Networks

2.2.2.1 Feed-Forward Neural Networks

Neural networks have become a popular option for STLF due to their ability to model complex and non-linear relationships, which are challenging for conventional techniques. A simple feed-forward NN consisting of an input layer, a single hidden layer containing a finite number of hidden neurons, and an output layer has the power to act as a universal approximator, regardless of the chosen activation function [20][21]. That is, no matter the complexity of the function being modelled, the NN can approximate it. In multi-step-ahead forecasting, the dimension of the output layer relates to the forecasting horizon. Feed-forward NNs have been implemented for time-series problems since the late 1980s and early 1990s [12].

In [22], the authors propose an NN with a single hidden layer to forecast 24 hours ahead using load and temperature data as input. [23] presents one of the original applications of a feed-forward NN in producing hourly load forecasts. In a comparison of many early studies using feed-forward NNs for STLF, [7] found that a single hidden layer was sufficient in most applications. Most networks compared in this study utilised sigmoidal activation functions for the hidden layers and a linear function for the output layer to give the predicted output. In [24], a comparison of deep feed-forward NNs was performed, with one case using tanh activation functions for the hidden layers and the other using sigmoid, with the former providing better prediction accuracy. For a simple feed-forward MLP network, [25] first applied a local learning technique designed to take into account the differences in input and output spaces caused by seasonal or cyclical variations, reported that the use of a single hidden neuron with a sigmoid activation function produced comparable results to models with larger numbers of hidden neurons. It is noted that the simplification of the network with local training may be required to achieve such results with such simple model architectures. For feed-forward NN models, the use of the Adam optimiser, which adapts the learning rate for each parameter as the learning progresses, has been regularly used [9][21][26] and has been shown to as having superior performance to stochastic gradient descent, which retains consistent gradients throughout training [7].

More recently, advances in NN architectures have seen more complex feed-forward models applied to STLF. For example, [26] used an approach inspired by deep Residual Networks (ResNets) to predict day-ahead forecasts using a much deeper architecture than the models mentioned above. [27] introduced the Deep Belief Network (DBN), which is composed of stacked layers of Restricted Boltzmann Machines (RBMs), which address the problems of slow learning and training becoming stuck in local minima.

2.2.3 Deep Networks

2.2.3.1 Long Short-Term Memory Networks

Simple Recurrent Neural Networks (RNNs) use time series data. They are commonly used for temporal problems and utilise a context layer that enables the

network to have memory of prior inputs, which influences the current input and output. This memory allows them to extract time-varying patterns in data [28]. However, RNNs can suffer from two well-known problems: exploding gradients and vanishing gradients. Vanishing gradients occur when the gradient becomes so small that the weight parameters update until they become insignificant (i.e. 0), stopping the network from learning. Conversely, when the gradients grow too large, the model weights explode, making the model unstable. Long Short-Term Memory (LSTM) models, first introduced in [29], were developed to overcome these problems. They maintain the same topological structure as an Elman RNN (a common type of simple RNN). However, the network's inner cells' composition differs [21]. Each cell implements a gated system, allowing the control of gradient flow.

The Bi-Directional LSTM (BD-LSTM) further develops this concept and uses two context layers, allowing information to proceed forward and backward through the network. Inputs are read from two directions, one from past to future and the other from future to past. This allows the network to preserve information from both the past and future at any point in time, giving it a long-range context. The BD-LSTM network structure used by [9] is given in 2.1, with the LSTM cell composition shown in 2.2.

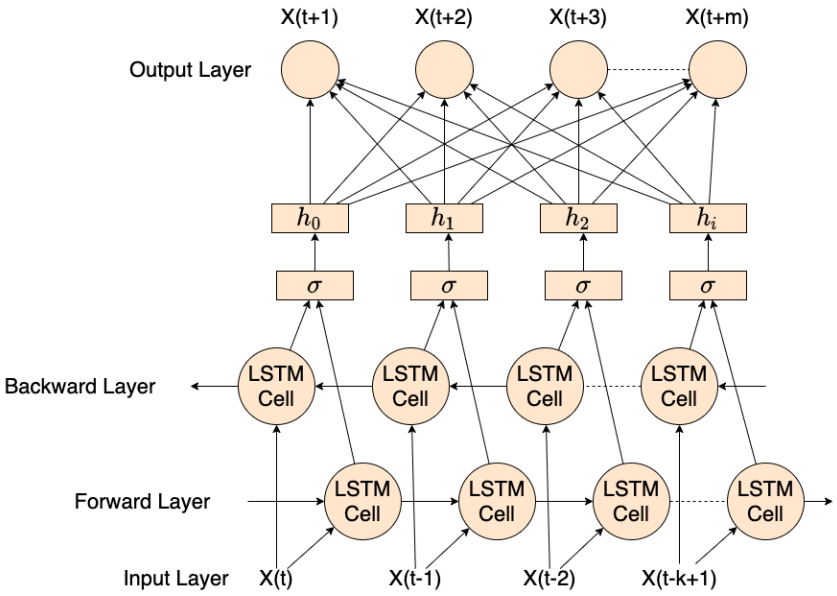

Figure 2.1: The BD-LSTM network structure [9].

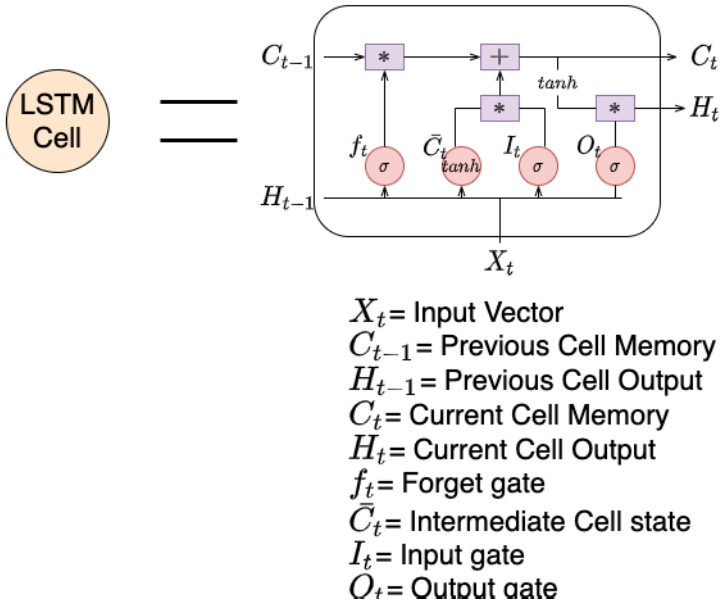

Figure 2.2: The composition of a LSTM cell, as depicted in [9]

In [30], an RNN was shown to have less error than a feed-forward NN on all metrics and all domains. In [9], a comparison of shallow and deep learning techniques for STLF was performed, including three variants of LSTM models: traditional LSTM, BD-LSTM, and Encoder-Decoder LSTM (ED-LSTM), as well as a One-Dimensional Convolutional Neural Network (1D-CNN). LSTM networks utilise memory cells and gates, allowing long-term dependencies to be better captured in temporal sequences. Simple RNNs have limited ability to learn long-term dependencies and can suffer from the issue of vanishing or exploding gradients. Therefore, LSTM models, which are variants of RNNs, are advantageous for STLF. This study's three LSTM models outperformed shallow learning models and simple RNNs, with ED-LTSM and BD-LTSM giving the best results.

For this project, we decided to study a variety of NN architectures rather than multiple versions of the same principle. Therefore, the BD-LSTM method, first introduced in [31], will be investigated further. For this method, [9] used a combination of five input neurons, one hidden layer with ten LSTM cells each for the forward and backward layers, and ten output neurons (one for each step-ahead prediction), with Rectified Linear Unit (ReLu) activation functions applied across the board.

2.2.3.2 One-Dimensional Convolutional Neural Networks

The CNN, first introduced in [32][33], has become prominent in character and image classification tasks. Recently, they have shown promise for time-series problems [34][35]. A CNN uses locally shared parameters with a kernel of fixed length k sliding over the input x with a predefined stride and padding value. The convolution operator uses element-wise multiplication over the input groups of width k and sums the results, producing a feature map. The model can comprise multiple filters, with the number used determining the model depth. The padding and stride values determine the spatial dimensions of the feature maps.

While the 1D-CNN tested by [9] was unable to match the performance of the LSTM networks used in their study, [14] introduced the use of Video Pixel Networks (VPN) in 1D-CNNs to handle time series applications, specifically for STLF, which may offer an improvement over the model used by [9]. A VPN was first introduced by [36] and is designed to estimate the discrete join function distribution of pixel values used in video data. The VPN enables local stochastic transitions from one pixel to the next without introducing independence assumptions. The 1D-CNN architecture proposed by [14] is shown in 2.3.

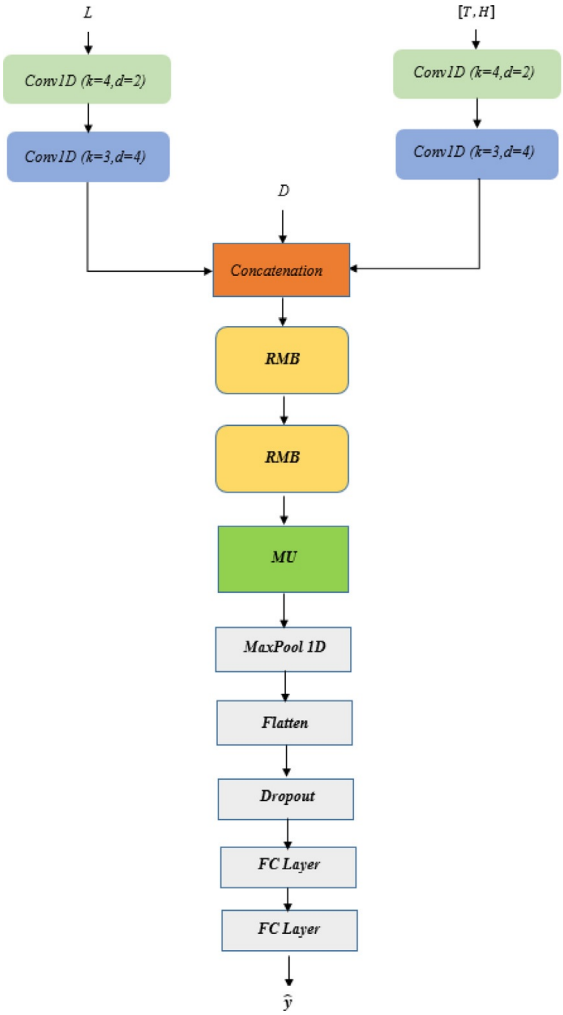

Figure 2.3: The 1D-CNN model architecture proposed by [14].

This network utilises Residual Multiplicative Blocks (RMBs; see Figure 2.4), an essential ingredient in VPNs, and a Multiplicative Unit (MB; see Figure 2.5), initially used for video processing problems. The MU used in [14] comprises gates merged into a convolutional layer and implements some slight modifications to the original method used by [36], resulting in performance improvements.

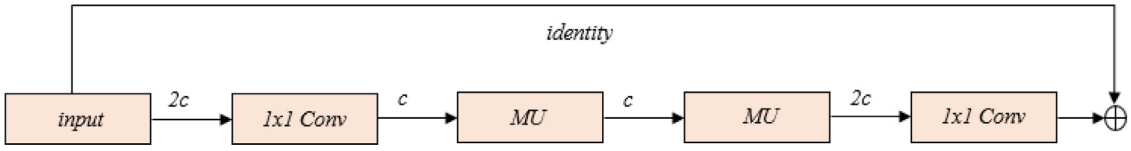

Figure 2.4: The RMB module structure used in [14].

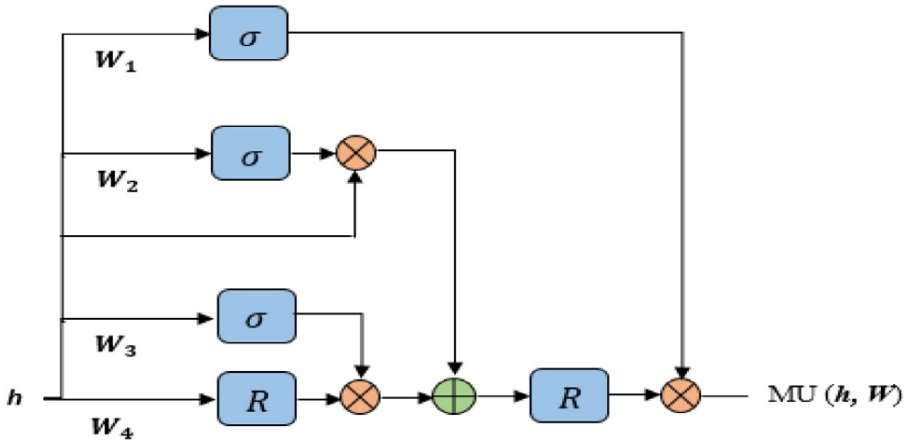

Figure 2.5: The MU structure used by [14].

2.3 Performance Measures

The mean squared percentage error (MAPE) is commonly used for time series predictions, often with the addition of a regularisation penalty [21]. The calculation is provided in Equation 2.3.1, where L_A is the actual load, L_F is the forecast load, N is the number of hours, and i is the time step index.

$$MAPE = \frac{100}{N} \sum_{i=1}^{N} \frac{|L_A^i - L_F^i|}{L_A^i} \tag{2.3.1}$$

Authors in [7] argued that while the MAPE has become the industry standard, it is not enough to assess model performance and that the loss function associated with the forecasting errors should be used in model evaluation, as is detailed extensively in [16].

Chapter 3

Material and Methods

3.1 Software

In the execution of this project, various software tools were utilised to perform analyses, manage data, and facilitate the computational aspects. The primary tools we used were as follows.

Python. *Python* served as the primary tool for data analysis in this project, chosen for its comprehensive selection of machine learning packages and accessible entry level. Additionally, given the project's emphasis on constructing neural networks to forecast demand, the *TensorFlow* framework was employed, offering robust support for *Python*. Other *Python* packages used included *Matplotlib* and *seaborn* for visualisations, *Schemdraw* for diagrams, *Pandas* for managing tables, and *NumPy* and *SciPy* for statistical operations.

MongoDB. *MongoDB* was utilised as the core database to maintain one central data source throughout data analysis and model building. As a scalable, cloud-hosted No-SQL database, *MongoDB* adeptly manages the project's substantial data volume. Its No-SQL architecture supports a wide range of data structures, proving particularly effective for storing tensors.

Bash. *Bash* was employed for the initial loading of raw data into *MongoDB*. It was chosen for its minimal overhead and compatibility across all operating systems.

Git. *Git* was used to manage and synchronise changes across team members, including *Python* code and *Markdown* meeting agendas and minutes. This version control system is essential for coordinating team efforts, allowing for seamless collaboration, historical tracking of project changes, and the ability to revert to previous versions if necessary, ensuring consistency and accuracy in the development process.

GitHub. *GitHub* was used to remotely host the repository for the project, providing redundancy against code loss or corruption. This platform not only ensures secure storage of all project artifacts but also facilitates easy access and collaboration among team members, regardless of their location, enhancing the robustness and reliability of the development process.

Overleaf. All reports were written in LaTeXand coordinated through *Overleaf*. LaTeXis a powerful markup language that enables professional reports to be easily produced. It also integrates well with Python, as *Pandas* tables can be saved as TEX files and read directly into reports. *Overleaf* extends these capabilities to collaborative efforts, allowing the team to write and review our documents simultaneously.

3.2 Description of the Data

This section provides a detailed overview of the data utilised in the project, including the storage of data in *MongoDB*, comprehensive data dictionaries, and high-level statistical summaries of the data sets, such as file sizes and quantities.

This project will use two input data sources that provide total demand (TD) data and temperature (Temp) data for several Australian states spanning 2010 to 2021. Summary statistics for the two datasets can be found in the table below.

Dataset	Records	Variables	Size in memory (MB)
total_demand	786,051	3	26.47
temperature	778177	4	21.5

Table 3.1: Summary statistics of source files. Size in memory refers to the total storage size for the dataset in MongoDB.

These source datasets were stored as separate collections in a *MongoDB* database. High-level data dictionaries, including field definitions and field statistics, have been included below.

Field Name	Description	Data Type	Example	Statistics
DATETIME	The UTC date-time of the demand recording in 30 min increments	DATETIME	2010-01-01 00:00:00	**Maximum**: 2021-03-18 00:00:00 **Minimum**: 2010-01-01 00:00:00
TOTALDEMAND	The total demand for a given state in the 30 min time interval	Float	5561.21	**Range**: [21.89, 14579.86] **Mean**: 5194.14 **Std. Dev.**: 2606.72
state	The Australian state the reading corresponds to	String	QLD	

Table 3.2: Data dictionary and summary statistics for total demand dataset

Field Name	Description	Data Type	Example	Statistics
DATETIME	The UTC date-time of the demand recording in 30 min increments	DATETIME	2010-01-01 00:00:00	**Maximum**: 2021-03-18 00:00:00 **Minimum**: 2010-01-01 00:00:00
TEMPERATURE	The recorded temperature for that location	Float	5561.21	**Range**: [-1.3, 44.7] **Mean**: 18.7 **Std. Dev.**: 6.0
state	The Australian state the reading corresponds to	String	QLD	
LOCATION	Location description of the reading	Enumerated String	Bankstown	

Table 3.3: Data dictionary and summary statistics for temperature dataset

3.3 Pre-processing Steps

A data pre-processing pipeline was used to clean, aggregate, enrich, and transform raw data sources stored in *MongoDB* into a collection of features used by the neural network models. The key steps were as follows (see Appendix A.3 for the process flow diagram for this pipeline).

Read Data. We ingested the total_demand and temperature datasets from *MongoDB* collections and converted them to *dataframes*.

Merge Datasets. We joined the temperature data with the demand data to create a consolidated feature set. This step also included removing the location information from the temperature source, as it was not used for building the models.

Impute missing values. We used an iterative imputation method to fill any missing values. This imputation is particularly important, as missing values may be introduced into the temperature readings due to the merge.

Split Datetime. We separated the timestamps into their components for modelling. These components included the year, month, and day of the week.

Calculate Daylight. We enriched the data using the *astral Python* package and calculated whether a record was taken during daylight.

Calculate Public Holiday. We enriched the data using the *holidays Python* package and calculated whether a record was taken during a public holiday.

Transform periodic features: We performed a polar transformation on periodic features (e.g., the month, day of the week).

Combine Features. We consolidated all engineered features into a single dataset.

Dependent Variables Added. We added the total demand data for 1 and 24 hours into the future to be used as the predicted variables when training the neural network for short-term forward prediction of the total_demand.

Write Features. We wrote the processed and cleaned feature dataset to a collection in *MongoDB*.

3.3.1 Read Data

To leverage *Python*'s extensive data processing capabilities and robust package support, the data was extracted from *MongoDB* using the *pymongo* API and converted into a DataFrame for manipulation.

3.3.2 Merge Datasets

The local temperature is a key variable when predicting the total demand, necessitating the merging of temperature and demand datasets. The location field was also removed from the temperature dataset, as it is not necessary for demand prediction in its current form [37].

3.3.3 Impute Missing Values

To build an effective model, the dataset must first be cleaned. This process includes imputing missing values. In this investigation, iterative imputation is employed, which utilises regression techniques to estimate missing values based on the relationships with other variables in the dataset, thereby preserving the kurtosis of the data. Maintaining the kurtosis is crucial as it ensures that the tail-heaviness or the peakedness of the distribution remains consistent, which is vital for the accuracy of statistical models and the validity of inferred conclusions [38].

3.3.4 Splitting Timestamp Feature

In the pre-processing pipeline, the date-time feature was split into its components to enhance the accuracy of the machine learning models, providing detailed temporal insights that allowed for more precise capture of seasonal and cyclic trends. These components include the year, month, day of the month, day of the week, whether it is a weekday, and the period of day[1].

3.3.5 Calculating Daylight

To further enrich the feature set, the timestamps were utilised with the *astral* package to determine whether a given record was taken during daylight hours. Daylight is a key factor in predicting demand, as daylight typically corresponds with reduced power usage in homes due to people being at work. Please note that this calculation incorporated the 'state' field to account for variations in daylight hours by location.

3.3.6 Calculating Public Holidays

Similarly, the *holidays* package was used to determine whether a timestamp fell on a public holiday. This was also a key feature in helping to account for spikes in demand caused by community events or increased home power usage on public holidays. This calculation also incorporated the 'state' field since the public holiday schedule differs by state.

3.3.7 Transforming Periodic Features

The pre-processing pipeline employed a polar coordinate transformation to encode periodic date-time features, ensuring that models effectively capture the cyclical and seasonal patterns. As described in Section 2.1, this method enhances adjacency and periodicity, which is crucial for accurate and efficient model training. The logic for this transformation is described below.

3.3.8 Writing Feature Set to MongoDB

Finally, after applying all transformations and cleaning operations, the processed features were stored in *MongoDB*, ready for integration into the modelling process. This was also performed using the *MongoDB* API.

3.3.9 One-Hot Encoding

As our *MongoDB* cluster had limited storage, the final pre-processing stage was performed locally with *Python*. This firstly involved one-hot encoding of the state variable using *scikit-learn*. The original string column of state names was then removed.

3.3.10 Generate Univariate Inputs

The second local process was generating the univariate columns. For this, a loop was used to join the table 48 times with an offset copy of itself in which the timestamp was increased in 30-minute intervals. These joins added 48 new variables to the dataset that showed the total demand for the past 24 hours, inclusive. As this created empty values in the newest and oldest records in the dataset, all records with a null value were removed from the table. The univariate approach was adopted because demand exhibits continuity over short intervals, making its immediate history

[1] This corresponds to splitting a single day into 48 periods to represent the data at its 30-minute granularity.

a strong predictor of near-future conditions. By utilising these specific time lags, the model can effectively capture daily and hourly fluctuations, which is crucial for accurate forecasting in scenarios where routine human activities and natural cycles heavily influence demand patterns.

3.3.11 Data Scaling

The final transformation of the data before training the models was standardising each variable using Function 3.3.1, where x is a single value, μ is the variable's mean, and s is the variable's standard deviation. This transformation centres the variable's distribution around zero and changes the individual values to their standard deviations from the mean without changing the shape of the data. By placing all variables on the same scale, this standardisation stops any variable from dominating the model by having a larger magnitude.

$$f_{\text{scale}}(x) = \frac{x - \mu}{s} \tag{3.3.1}$$

3.3.12 Train, Validation, and Test Splitting

The data were split into train, validation, and test sets to develop and test the models. As the data are for time-series modelling, they were sorted chronologically before splitting. The most recent 20 per cent of the data were reserved for testing. To ensure the validity of the process, these data remained "unseen" by the models or modeller until the final tests. The older 80 per cent of the data were then split five times into training and validation sets for time-series cross-validation. For this process, the oldest $\frac{1}{6}$ of the data are used to forecast the second oldest $\frac{1}{6}$. Then the oldest $\frac{1}{3}$ are used to forecast the third oldest $\frac{1}{6}$. This pattern repeats until the oldest $\frac{5}{6}$ of the data are used to forecast the most recent $\frac{1}{6}$. The mean validation MAPE of these rolling forecast origins is then used to determine which model variant performs best [39]. The train-validation data were used for model development and hyperparameter tuning. After this, the models were retrained on the entire train set (i.e. 80% of the data) and validated on the test set. The results reported below were subsequently generated from the test set.

3.3.13 Feature Reduction

After engineering various features and consolidating the training data, an attempt was made with L1 regularisation to reduce the number of features used in the final models. This reduction would benefit the models by decreasing training time and reducing the data required to utilise them. The subsequent results are outlined in Table 3.4. A deep multivariate MLP with L1 regularisation factor $\lambda_{L1} = 0.1$ on the kernel weights was trained on the NSW data for 2000 epochs. The subsequent model was used to exclude the least impactful variables from the model. The kernel weights relating to each variable throughout the neural network were summed, and variables with sums falling below ϵ were excluded. The ϵ values 10^{-3} and 10^{-4} were reviewed. The results showed that removing low-impact variables was preferable to simply using L1 regularisation, as it decreased training time and improved the MAPE. However, the acceleration was only modest, and the validation loss deteriorated,

compared to the unregularised model, more than mild improvements in training time could justify. As such, all features were included in the final model tests.

Model	Training Minutes	Count of Variables	Validation MAPE
$\lambda_{L1} = 0.1$	5:09	66	4.68
$\lambda_{L1} = 0.0$	5:09	66	2.58
$\epsilon > 10^{-3}$	4:54	51	2.80
$\epsilon > 10^{-2}$	4:43	19	3.01

Table 3.4: The feature reduction results of the four models reviewed. The λ_{L1} values represent the weight of the L1 regularisation, with $\lambda_{L1} = 0$ indicating no regularisation. The ϵ values represent the minimum sum of coefficients for a variable to be included in the model.

3.4 Assumptions

When conducting feature engineering, three key assumptions were made to obtain a valid set of features. The first assumption is that the temperature data from a specific location within the city was applied to the entire state. This assumption is considered valid because a significant portion of the demand originates from the city. The second is that daylight calculations were based on the capital city's data. However, it is recognised that temperature variations can be substantial across different areas of the state. The third is that the identification of public holidays was based on the schedule of the capital city, despite known variations in holiday observance across different regions.

Assumptions were also made during the modelling. Given the size of the dataset, model architecture experimentation, including hyperparameter grid-search cross-validation, utilised only the data from NSW. This step dramatically accelerated experimentation, which was necessary given the time constraints of the project. However, it meant that the process assumed that the patterns within the NSW data would apply to all states. There are good reasons, such as environmental and economic differences, to doubt this assumption, but it was necessary given the current limitations. Similarly, the grid search for the hyperparameters was only performed on the shallow multivariate versions of the MLP and BD-LSTM. The generalisability of these results are unclear, but we needed to assume this would be sufficient due the the considerable time constraints. For the same reason, we also assumed that Adam optimisation would be optimal and conducted our tests accordingly. This, of course, violates the "no free lunch" theorem. The final assumption made during modelling is that issues of model complexity and comprehensibility would be irrelevant to our report. This is often not the case. However, as we are only reviewing neural networks, which are always somewhat complex and never readily comprehensible, this consideration was deemed superfluous to our modelling. Subsequently, only runtime performance, goodness of fit, and forecast accuracy were considered.

3.5 Modelling Methods

3.5.1 Model Design

Two broad models were used in this report: multilayer perceptrons (MLPs) and bidirectional long short-term models (BD-LSTMs).[2] Four variants were created for both of these models: univariate, multivariate, deep, and shallow, with eight models ultimately being instantiated and tested. The univariate models took 49 inputs. These covered the total demand values from the time of the record to 24 hours beforehand, inclusive, in half-hour intervals. The multivariate models, taking 70 inputs, included these 49 inputs, temperature, and various engineered features (see Section 3.3.1). The univariate models both had one hidden layer.[3] The deep MLPs had ten hidden layers, whereas the deep BD-LSTMs had two bidirectional layers. With one exception, all hidden layers had 32 units. The deep BD-LSTM had 64 units in its first layer, making this model both wider and deeper. All BD-LSTM hidden layers used a tanh activation function. The shallow MLPs used relu in its hidden layer, and the deep MLP alternated between relu and linear. All output layers used a linear activation function and produced two outputs covering the one-hour-ahead and 24-hours-ahead prediction horizons.

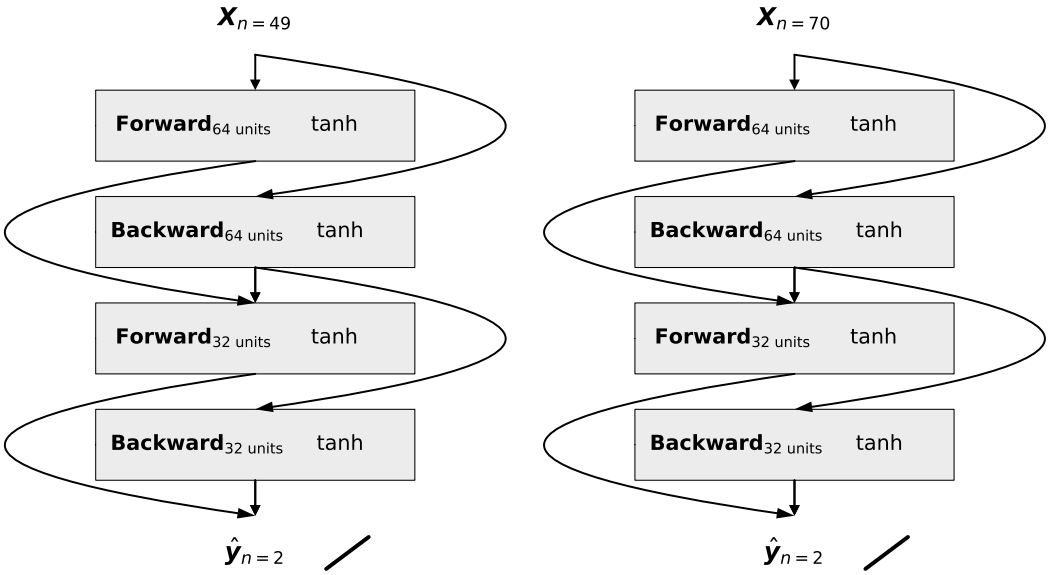

Figure 3.1: The architecture of the uni- and multivariate deep BD-LSTMs. The deep variants have two hidden bidirectional LSTM units. The univariate model takes 49 inputs and delivers two outputs, and the multivariate model takes 70 inputs and delivers two outputs. Each bidirectional unit has two layers: one parsing the data forward and the other backward. Both directional layers take inputs from the layer before the bidirectional unit and pass their outputs to the layer following the bidirectional unit. All hidden layers use a tanh activation function. The output uses a linear or identity activation.

[2]The team also hoped to include a one-directional convolutional neural network, but time and technical constraints ultimately made this infeasible.

[3]The BD-LSTM technically had two hidden layers, as the bidirectional layer consists of two layers: one parsing the data forward and the other backward.

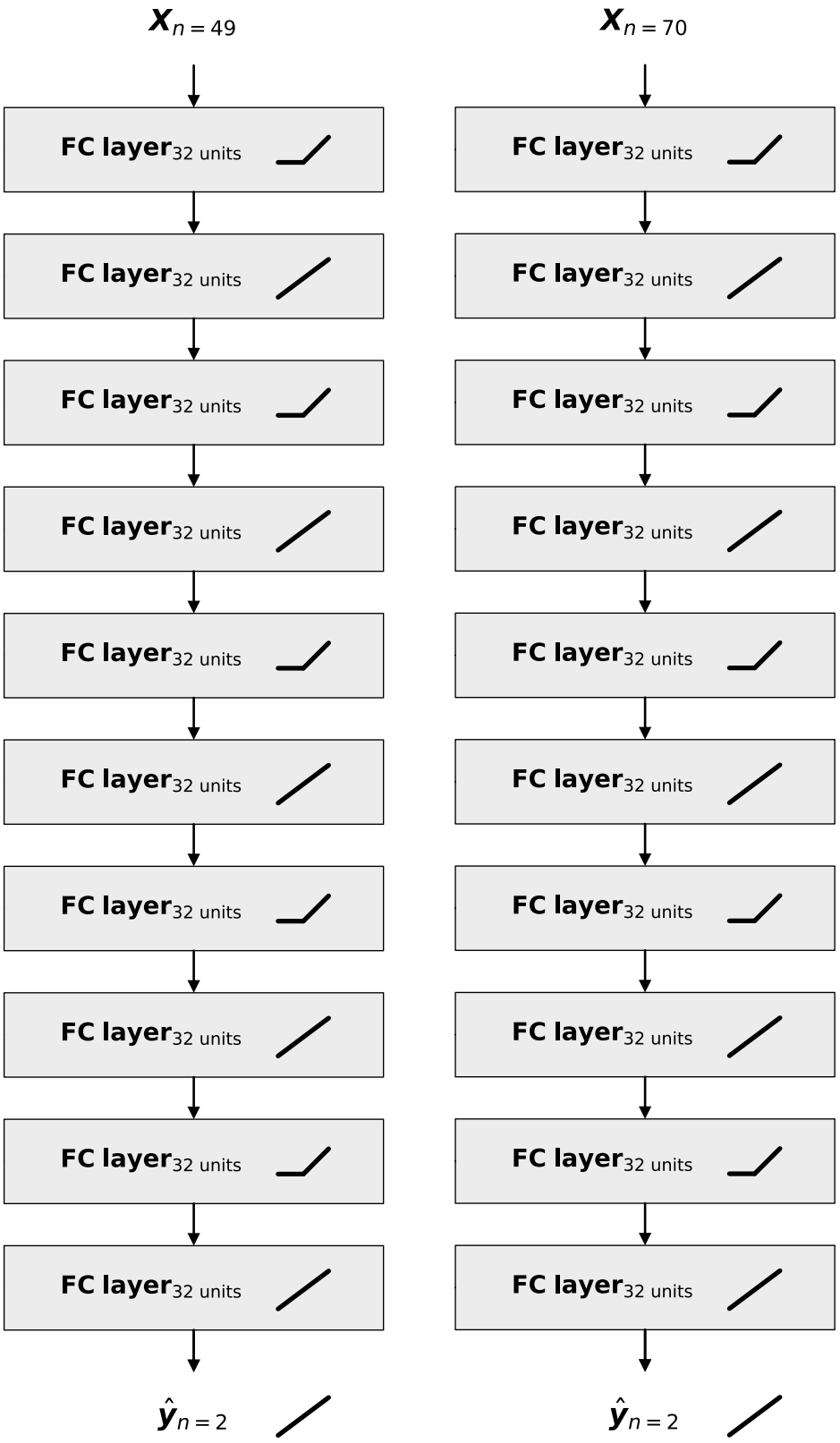

Figure 3.2: The architecture of the uni- and multivariate deep MLPs. The deep variants have ten hidden layers. The univariate model takes 49 inputs and delivers two outputs, and the multivariate model takes 70 inputs and delivers two outputs. The hidden layers alternate between relu and linear or identity activation functions. The output uses a linear or identity activation.

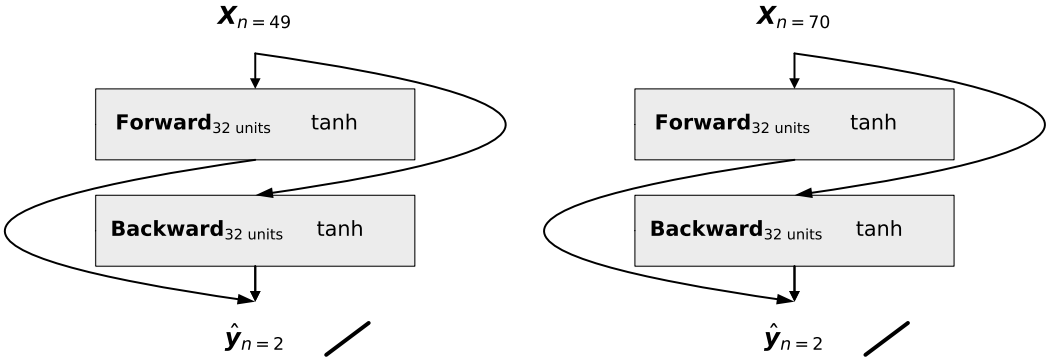

Figure 3.3: The architecture of the uni- and multivariate shallow BD-LSTMs. The shallow variants have one hidden bidirectional LSTM unit. The univariate model takes 49 inputs and delivers two outputs, and the multivariate model takes 70 inputs and delivers two outputs. The bidirectional unit has two layers: one parsing the data forward and the other backward. Both directional layers take inputs from the input layer and pass their outputs to directly to the output layer. Both hidden layers use a tanh activation function. The output uses a linear or identity activation.

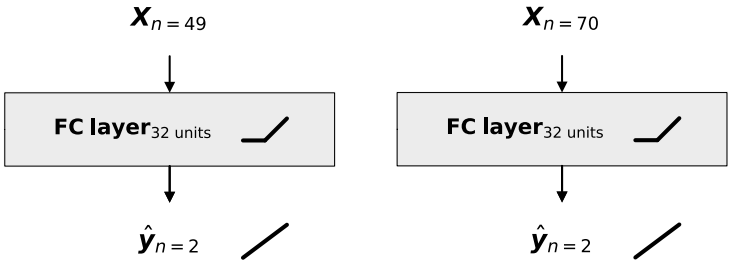

Figure 3.4: The architecture of the uni- and multivariate shallow MLPs. The deep variants have one hidden layers. The univariate model takes 49 inputs and delivers two outputs, and the multivariate model takes 70 inputs and delivers two outputs. The hidden layer has a relu activation functions. The output uses a linear or identity activation.

3.5.2 Hyperparameter Tuning

Given the project's time constraints, we used larger batch sizes and learning rate scheduling that included a large learning rate to accelerate training [16]. Beyond acceleration, larger batch sizes and learning rates contribute to model regularisation, making these adjustments broadly beneficial. For the batch size, we tested models using 2^8, 2^9, and 2^{10} (see Table 3.5). The learning rate was scheduled to adjust between epochs (i.e. the step size was set at one epoch), alternating between a large and a small rate. To find an appropriate learning rate schedule, an experiment

was run using five-fold cross-validation to compare these four scheduling functions, where the function's output provides the learning rate for the current epoch.

$$f_1 = \text{IF epoch}\%2 = 1 \text{ THEN } 10^{-4} \text{ ELSE } 10^{-3} \tag{3.5.1}$$

$$f_2 = \text{IF epoch}\%2 = 1 \text{ OR epoch} > \frac{\text{epoch}}{2} \text{ THEN } 10^{-4} \text{ ELSE } 10^{-3} \tag{3.5.2}$$

$$f_3 = 10^{-4} \tag{3.5.3}$$

$$f_4 = \text{IF epoch}\%2 = 1 \text{ OR epoch} > \text{epoch} - 100 \text{ THEN } 10^{-4} \text{ ELSE } 10^{-3} \tag{3.5.4}$$

The first function alternates the learning rate at every epoch. The second alternates for only the first half of all epochs before only using the lower rate. The third only uses the lower rate. The fourth alternates until the last 100 epochs, for which it only uses the lower rate. The results, shown in Figure 3.5, indicate that f_4 performs best. However, when reviewing the training histories (see Figure 3.6), we can see that, beginning around epoch 1900, the training curve for f_4 improves significantly while its validation curve deteriorates. This divergence indicates overfitting [40]. Fortunately, f_1 performed similarly well and showed no signs of misspecification. As such, f_1 was used in all subsequent modelling.

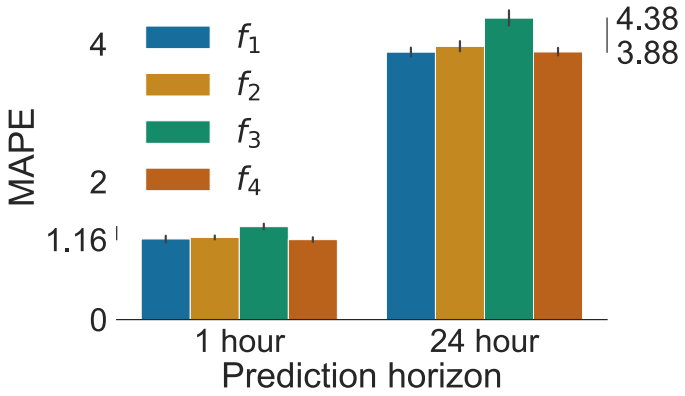

Figure 3.5: Results of the learning rate scheduling experiment showing the MAPE (CI: 95%) for both prediction horizons for each function.

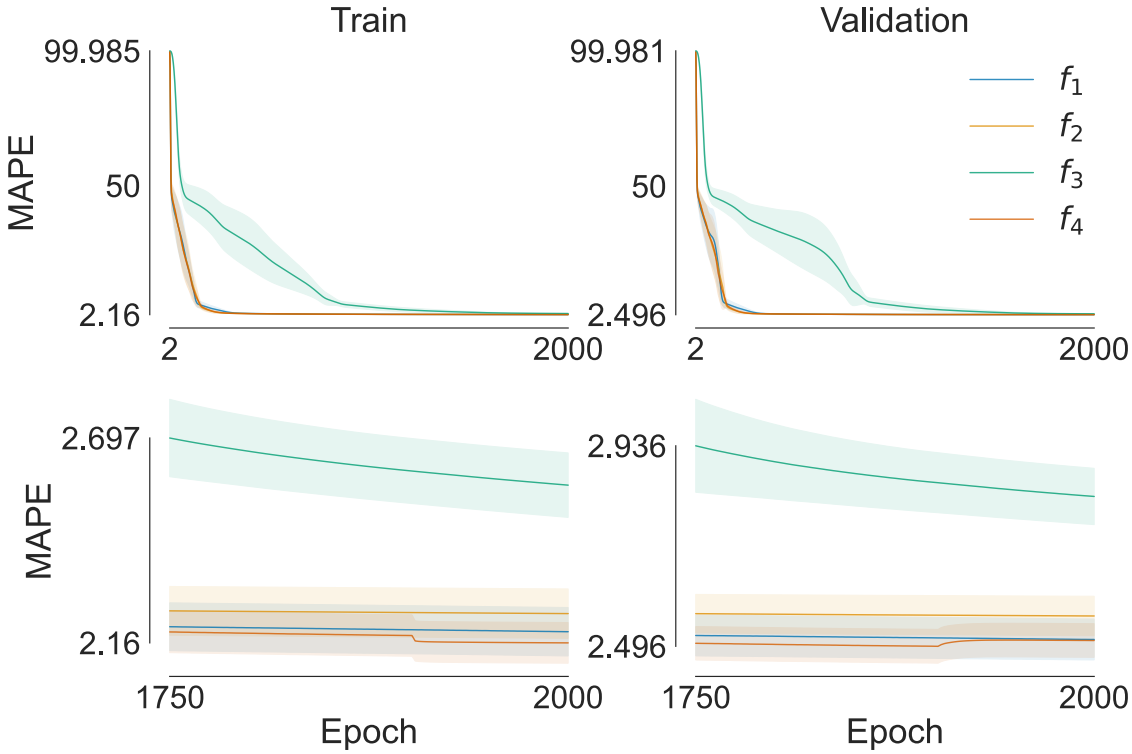

Figure 3.6: Training histories from the learning rate scheduling experiment showing the test and validation MAPE (CI: 95%). Due to their significance, the final 250 epochs are shown in greater detail.

For the remaining hyperparameter tuning, a grid search with five-fold cross-validation was used to select the optimal hyperparameters. A shallow multivariate version of both models was used to estimate the optimal hyperparameter combination for each group of models. A parameter grid (Table 3.5) covering 27 combinations of batch size, weight decay, and momentum (β_1) was searched. The combination resulting in the best mean results from the cross-validation was selected. For both MLPs and BD-LSTMs, a batch size of 2^8 and weight decay of 10^{-5} were optimal. The MLP performed better with a momentum of 0.95, whereas the BD-LSTM performed better with $\beta_1 = 0.8$.

Hyperparameter	Values Searched
Batch Size	$[2^8, 2^9, 2^{10}]$
Optimizer Weight Decay	$[10^{-3}, 10^{-4}, 10^{-5}]$
Optimizer β_1	$[0.8, 0.9, 0.95]$

Table 3.5: The hyperparameter matrix used in the grid search. The optimiser values were passed to the Adam optimiser, and the batch size was passed to the models' fit methods.

Model	Batch Size	Optimizer Weight Decay	Optimizer β_1
MLP	2^8	10^{-5}	0.95
BD-LSTM	2^8	10^{-5}	0.80

Table 3.6: The optimal hyperparameters for each model found in the grid search. The optimiser values were passed to the Adam optimiser, and the batch size was passed to the models' fit methods.

3.5.3 Model Training

To train the models before producing the final results, we initially ran each model for approximately two hours. As BD-LSTMs train on the data recurrently, they take magnitudes longer to complete the same number of epochs as an MLP (see Table 3.7). For instance, the univariate shallow MLP completed 10,500 epochs in slightly under two hours. In a comparable amount of time, the univariate shallow BD-LSTM completed 105 epochs. The slowest model was the multivariate deep BD-LSTM, which only completed 21 epochs within two hours. Once this first round of training was completed, the learning curves were analysed, and it was determined that the MLPs were sufficiently trained. The remaining training was then shared between the team to accelerate the process. Unfortunately, however, we ultimately lacked sufficient time to train the models to optimality (see Section 5.2). Total training times and epochs are listed in Table 3.8.

Label	Epochs	Training Hours
Multivariate Deep BD-LSTM	21	1:59
Univariate Deep BD-LSTM	30	2:12
Multivariate Shallow BD-LSTM	73	2:08
Univariate Shallow BD-LSTM	105	2:11
Univariate Deep MLP	3,950	1:44
Multivariate Deep MLP	4,000	1:48
Multivariate Shallow MLP	10,000	2:01
Univariate Shallow MLP	10,500	1:57

Table 3.7: The number of training epochs completed within approximately two hours. MLPs can undergo magnitudes more training than BD-LSTMs within the same time frame. We hypothesise that this had a significant impact on the test results.

Model	Epochs	Total Training Hours
Multivariate Deep BD-LSTM	189	9:53
Univariate Deep BD-LSTM	150	7:07
Multivariate Shallow BD-LSTM	511	9:57
Univariate Shallow BD-LSTM	315	4:38
Univariate Deep MLP	3950	1:44
Multivariate Deep MLP	4000	1:48
Multivariate Shallow MLP	10000	2:01
Univariate Shallow MLP	10500	1:57

Table 3.8: The number of epochs completed and hours of training performed for each model. Models completing significantly more than two hours of training were run on a better suited machine to accelerate the process.

Chapter 4

Exploratory Data Analysis

The exploratory data analysis focused on identifying key features significantly impacting the total demand. Visual analysis and correlation assessments were performed to determine key predictors for inclusion in the machine learning models. This process was critical in enhancing the accuracy and efficiency of the predictive outcomes.

4.1 Total Demand Distributions

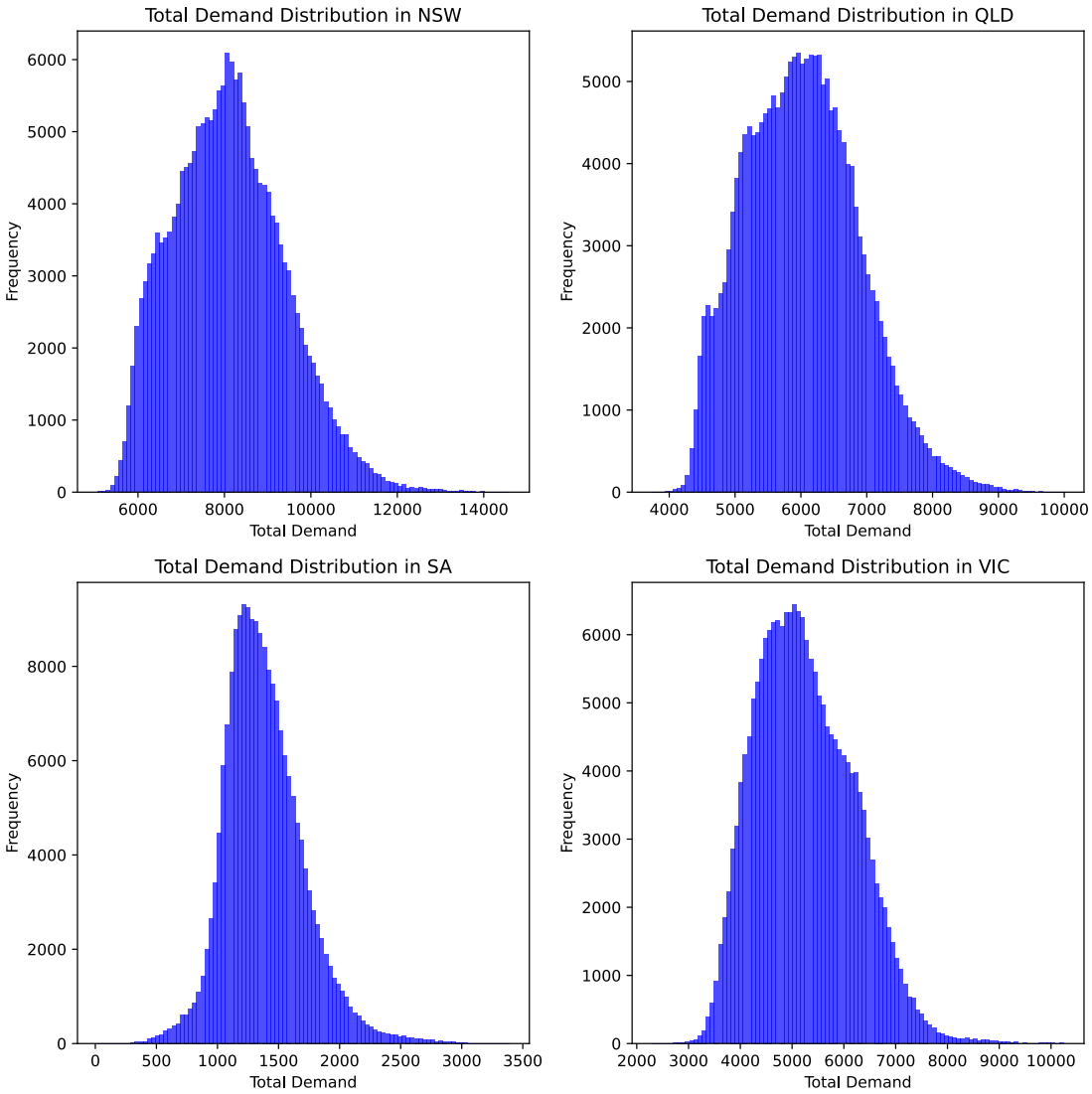

Figure 4.1: Histograms of total demand for NSW, QLD, SA and VIC

Figure 4.1 shows that each state's demand data is approximately normal. This stability helped facilitate faster and more stable convergence during training since the activation functions typically perform better with inputs that avoid extreme values. Each distribution displays a leptokurtic shape with some right skew, indicating that demand typically concentrates around the mean with some high-demand outliers.

4.2 Temperature Distributions

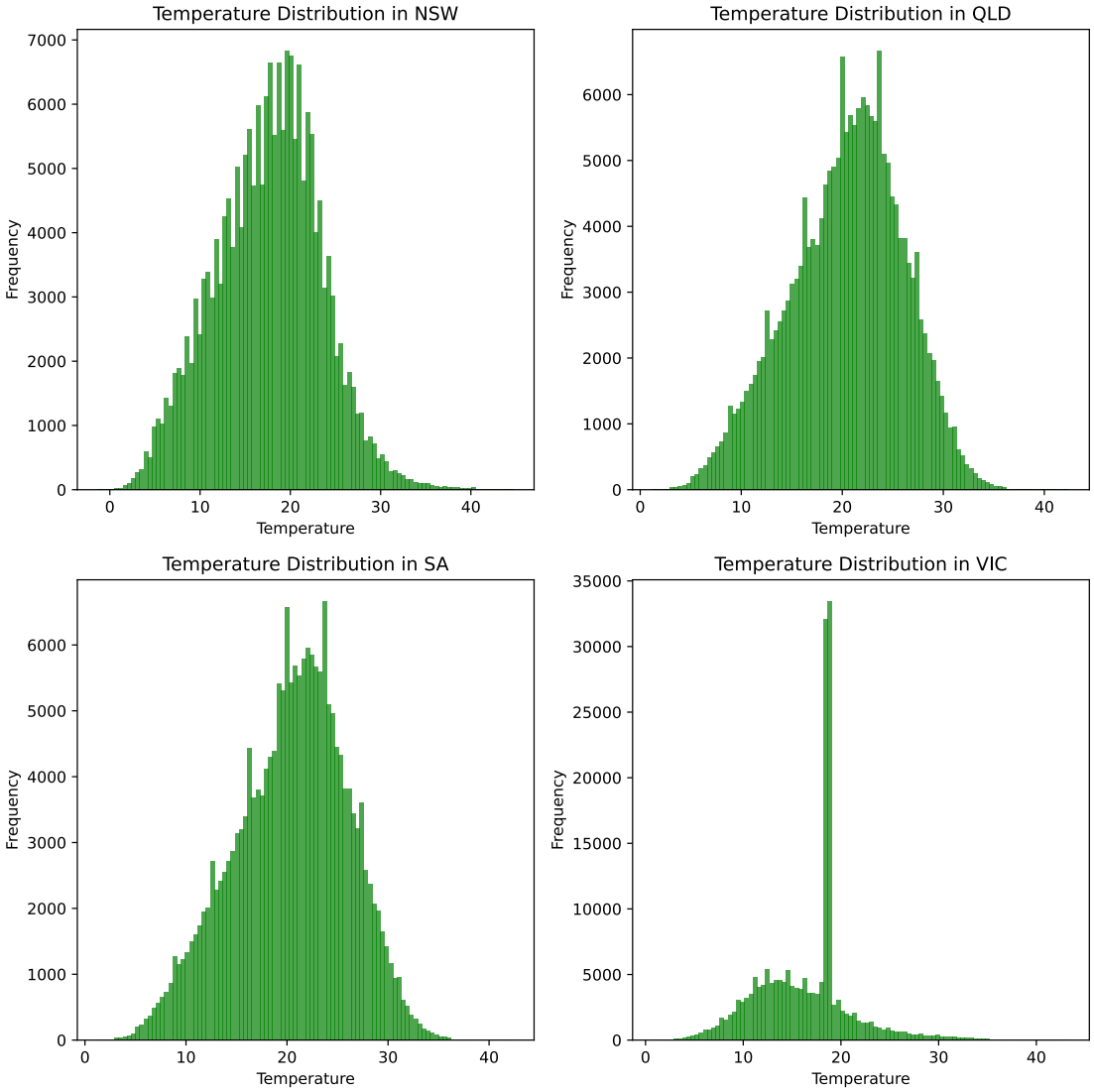

Figure 4.2: Histograms of temperature for NSW, QLD, SA and VIC

The temperature distributions in Figure 4.2 follow a similar leptokurtic approximately normal distribution. The distributions also exhibit several peaks that can be attributed to the lesser granularity of the temperature data, resulting in common values occurring more frequently. In the VIC dataset, a substantial peak occurs around the mean. This peak resulted from imputing the missing values that entered the dataset when merging with the temperature and total demand data.

4.3 Correlation Scatter

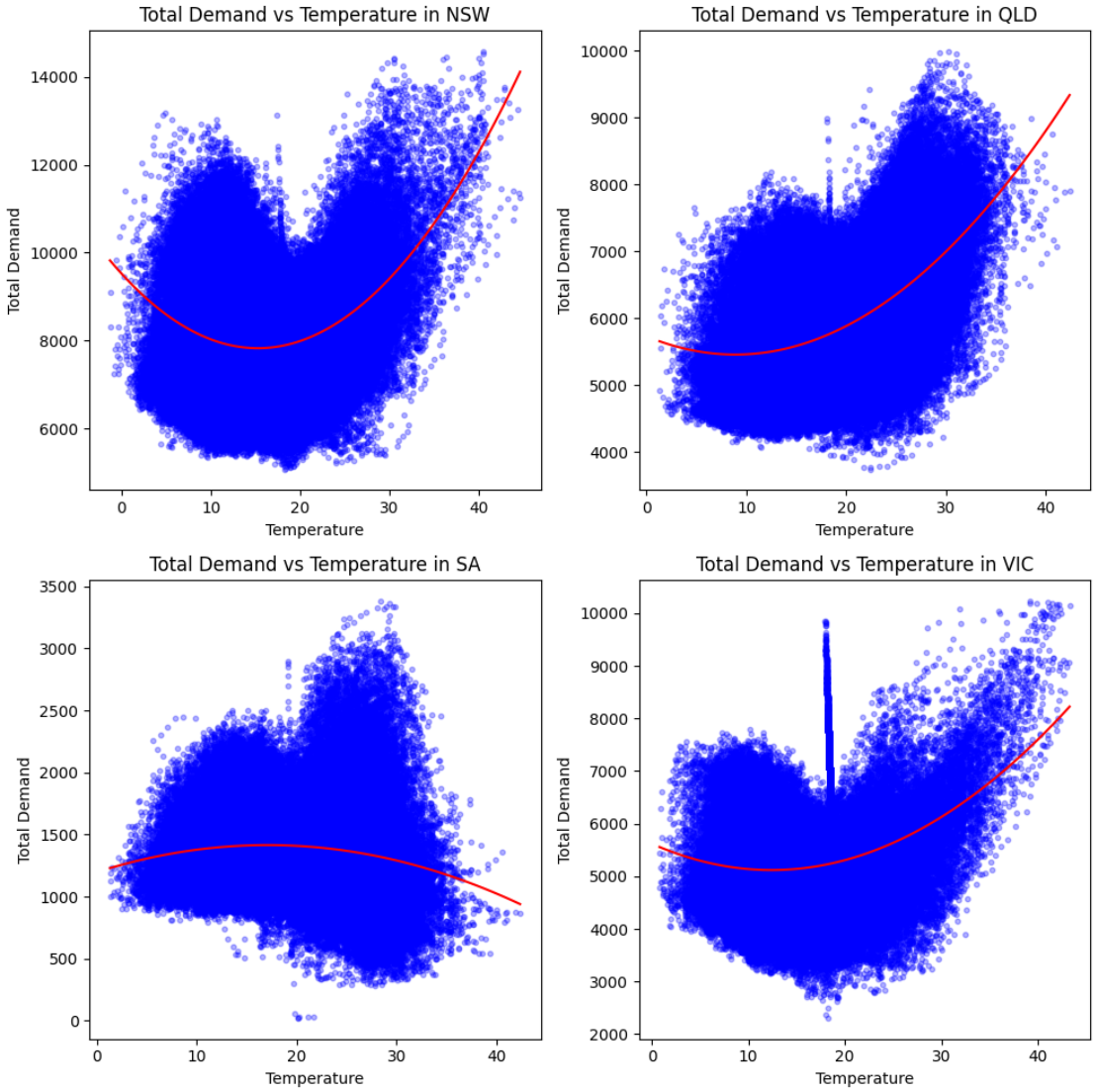

Figure 4.3: Demand vs. Temperature Correlation

The scatter plots in Figure 4.3 show a strong quadratic correlation between temperature and demand. This correlation can be attributed to increased heating or cooling for extreme temperatures. Interestingly, the quadratic relationship for SA is negative. However, by inspecting the plot, we can see that a higher-order function may better represent the relationship.

4.4 Periodic Feature Box Plots

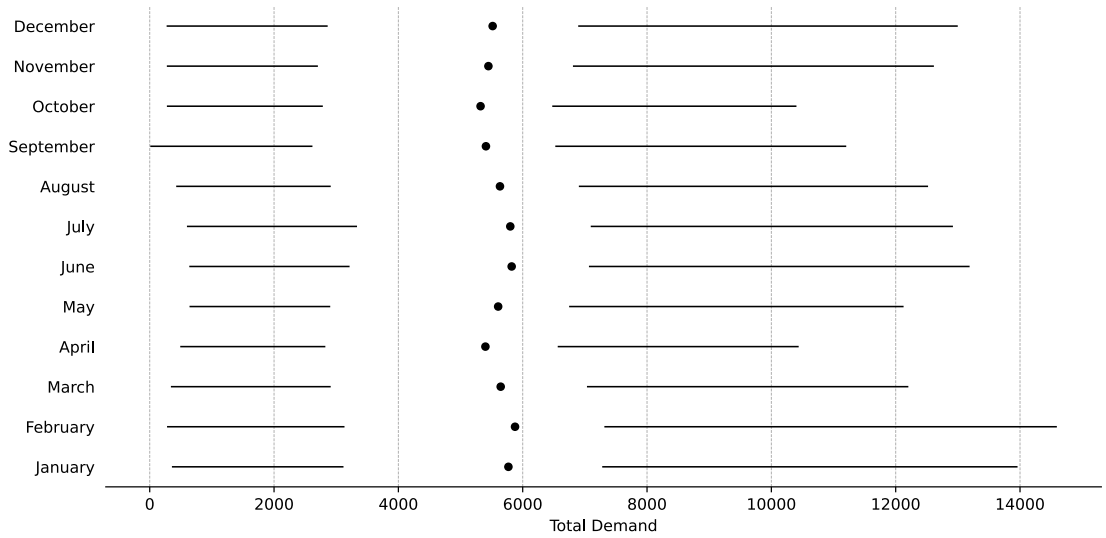

Figure 4.4: Demand by Month Quartile Plot

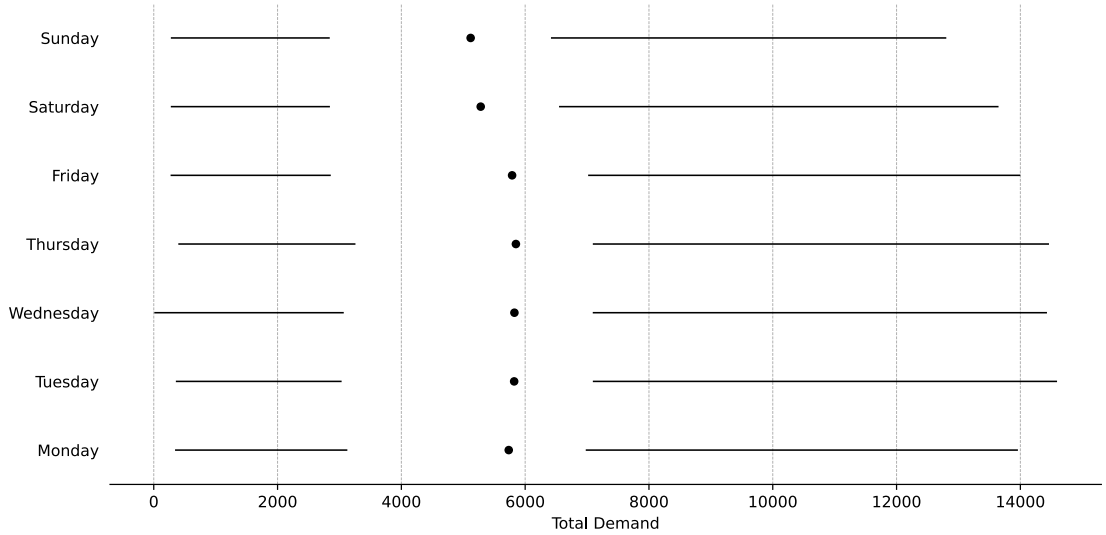

Figure 4.5: Demand by Day of Week Quartile Plot

Figure 4.4 shows a periodic correlation between the total demand and month of the year, which can again be attributed to increased heating and cooling in winter and summer. This correlation also demonstrates the value in performing the polar transformation described in Section 3.3.7. In Figure 4.8, there is an apparent increase in energy demand during the week and potentially a slight increase in mid-week demand.

4.5 Boolean Feature Box Plots

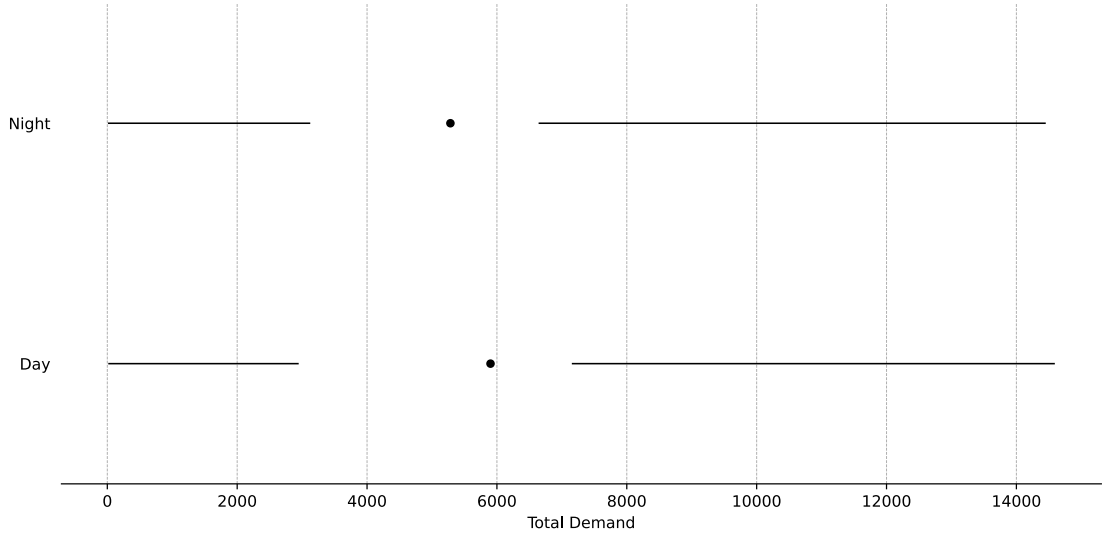

Figure 4.6: Daylight Quartile Plot

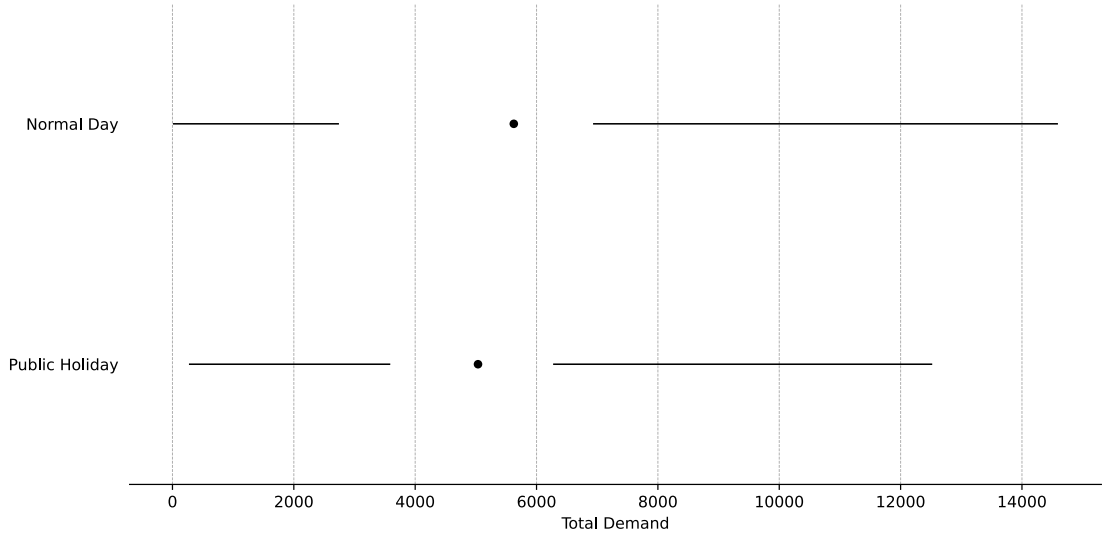

Figure 4.7: Public Holiday Quartile Plot

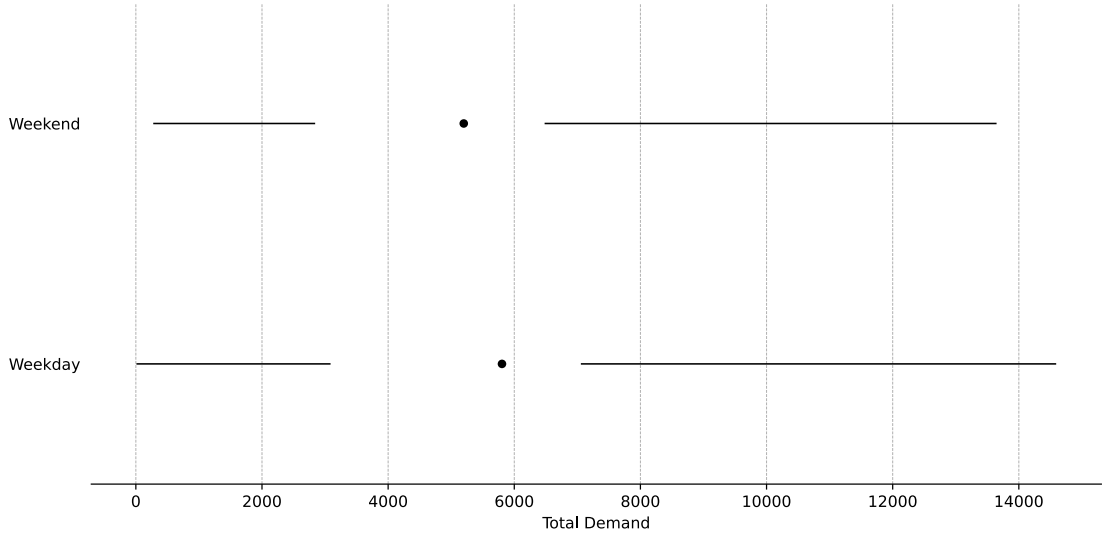

Figure 4.8: Weekday Quartile Plot

Figure 4.6 shows that whether there is daylight has a substantial impact on demand. This impact can be attributed to most of the population sleeping at night and, therefore, using less energy.

Figure 4.7 shows that whether it is a public holiday strongly predicts total demand. Furthermore, given that it is binary, it can improve model accuracy with less computation required for training. It should also be noted that the spread (range and IQ range) on values is much smaller for public holidays since there are significantly less public holidays leading to variability uncertainty in the data.

Chapter 5

Analysis and Results

5.1 Result Summary

We report the mean absolute percentage error (MAPE), the loss function of all models, and the root mean squared error (RMSE). When reviewing the basic ranking of the models (see Table 5.1), we see that all MLP models outperformed all BD-LSTM (see Section 5.2 for caveats). However, no broadly superior model is apparent from these results, as the univariate MLPs performed best on 1-hour-ahead forecasts, while the multivariate MLPs performed best on 24-hour-ahead forecasts. The competition between the models is further illustrated by the underlying MAPE results (see Table 5.2 and Figure 5.1). These illustrate that, although their batch-wise mean MAPEs are nearly indistinguishable, they perform substantially differently with different prediction horizons. Amongst the BD-LSTMs, the shallow multivariate model performs best, and the shallow univariate model performs worst. Between the deep BD-LSTMs, the multivariate model performed better on the 24-hour prediction horizon, and the univariate model performed marginally better on the one-hour prediction horizon.

A similar outcome is evident in the equivalent tables and visualisation for the test RMSE (Tables 5.3 and 5.4; Figure 5.2). Again, the multivariate MLPs perform best on 24-hour-ahead forecasts, and the univariate MLPs perform best on one-hour-ahead forecasts. However, the batch-wise means now place the multivariate MLPs as the best overall model on balance. Additionally, the univariate shallow BD-LSTM performs sufficiently well on this metric – especially on the 24-hour prediction horizon, where it places third – to place fourth overall. The deep multivariate BD-LSTM outperforms both univariate LSTMs on all aspects of RMSE.

The success of the MLPs as a category is also seen in Table 5.5. Here, we can see that the MLPs' mean MAPE is superior to the BD-LSTMs' by both prediction horizons and their mean. Table 5.7 shows that the shallow models performed marginally better than the deep models by both prediction horizons and their mean. Table 5.9 shows that the multivariate models outperformed the univariate models overall, but the univariate models did perform marginally better on the 24-hour prediction horizon. Identical patterns are borne out by the aggregated mean RMSEs in Tables 5.6, 5.8, and 5.10.

Model	1 Hour	24 Hour	Mean	Mean Rank
Multivariate Shallow MLP	3	1	1	1.67
Univariate Deep MLP	2	3	2	2.33
Multivariate Deep MLP	4	2	3	3.00
Univariate Shallow MLP	1	5	4	3.33
Multivariate Shallow BD-LSTM	5	4	5	4.67
Multivariate Deep BD-LSTM	7	6	6	6.33
Univariate Deep BD-LSTM	6	8	7	7.00
Univariate Shallow BD-LSTM	8	7	8	7.67

Table 5.1: Ranking by MAPE results for the eight models. The tests covered the one- and 24-hour ahead prediction horizons and their batch-wise mean. The models are ordered by their mean ranking.

Model	1 Hour	24 Hour	Mean
Multivariate Shallow MLP	2.45	6.73	4.59
Univariate Deep MLP	1.97	7.31	4.64
Multivariate Deep MLP	2.55	6.85	4.70
Univariate Shallow MLP	1.78	7.66	4.72
Multivariate Shallow BD-LSTM	2.68	7.46	5.07
Multivariate Deep BD-LSTM	3.38	7.99	5.68
Univariate Deep BD-LSTM	3.21	8.47	5.84
Univariate Shallow BD-LSTM	3.75	8.29	6.02

Table 5.2: MAPE test results for the eight models. The tests covered the one- and 24-hour ahead prediction horizons and their batch-wise mean. The models are ordered by the mean.

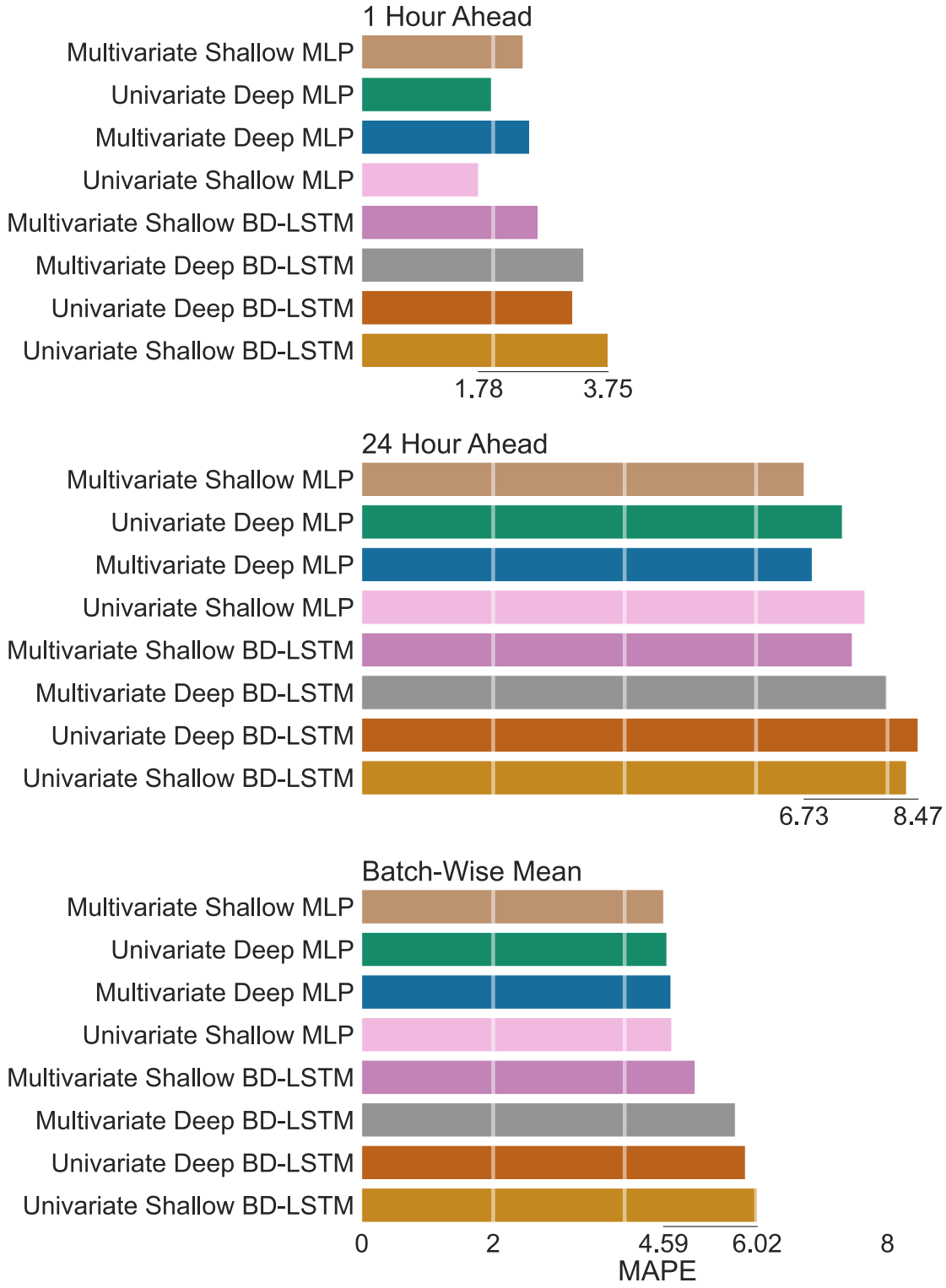

Figure 5.1: Horizontal bar charts with range frames showing the MAPE test results for the eight models. The tests covered the one- and 24-hour ahead prediction horizons and their batch-wise mean. The models are ordered by the mean.

Model	1 Hour	24 Hour	Mean	Mean Rank
Multivariate Shallow MLP	3	1	1	1.67
Multivariate Deep MLP	4	2	2	2.67
Univariate Deep MLP	2	4	3	3.00
Univariate Shallow MLP	1	6	5	4.00
Multivariate Shallow BD-LSTM	5	3	4	4.00
Multivariate Deep BD-LSTM	6	5	6	5.67
Univariate Deep BD-LSTM	7	7	7	7.00
Univariate Shallow BD-LSTM	8	8	8	8.00

Table 5.3: Ranking by RMSE results for the eight models. The tests covered the one- and 24-hour ahead prediction horizons and their batch-wise mean. The models are ordered by their mean ranking.

Model	1 Hour	24 Hour	Mean
Multivariate Shallow MLP	157.53	460.65	344.25
Univariate Deep MLP	127.79	544.83	395.71
Multivariate Deep MLP	166.32	500.69	373.06
Univariate Shallow MLP	119.28	577.52	416.99
Multivariate Shallow BD-LSTM	189.88	538.41	403.69
Multivariate Deep BD-LSTM	218.85	569.55	431.44
Univariate Deep BD-LSTM	223.82	594.86	449.41
Univariate Shallow BD-LSTM	241.10	595.01	453.97

Table 5.4: RMSE test results for the eight models. The tests covered the one- and 24-hour ahead prediction horizons and their batch-wise mean. The models are ordered by the mean.

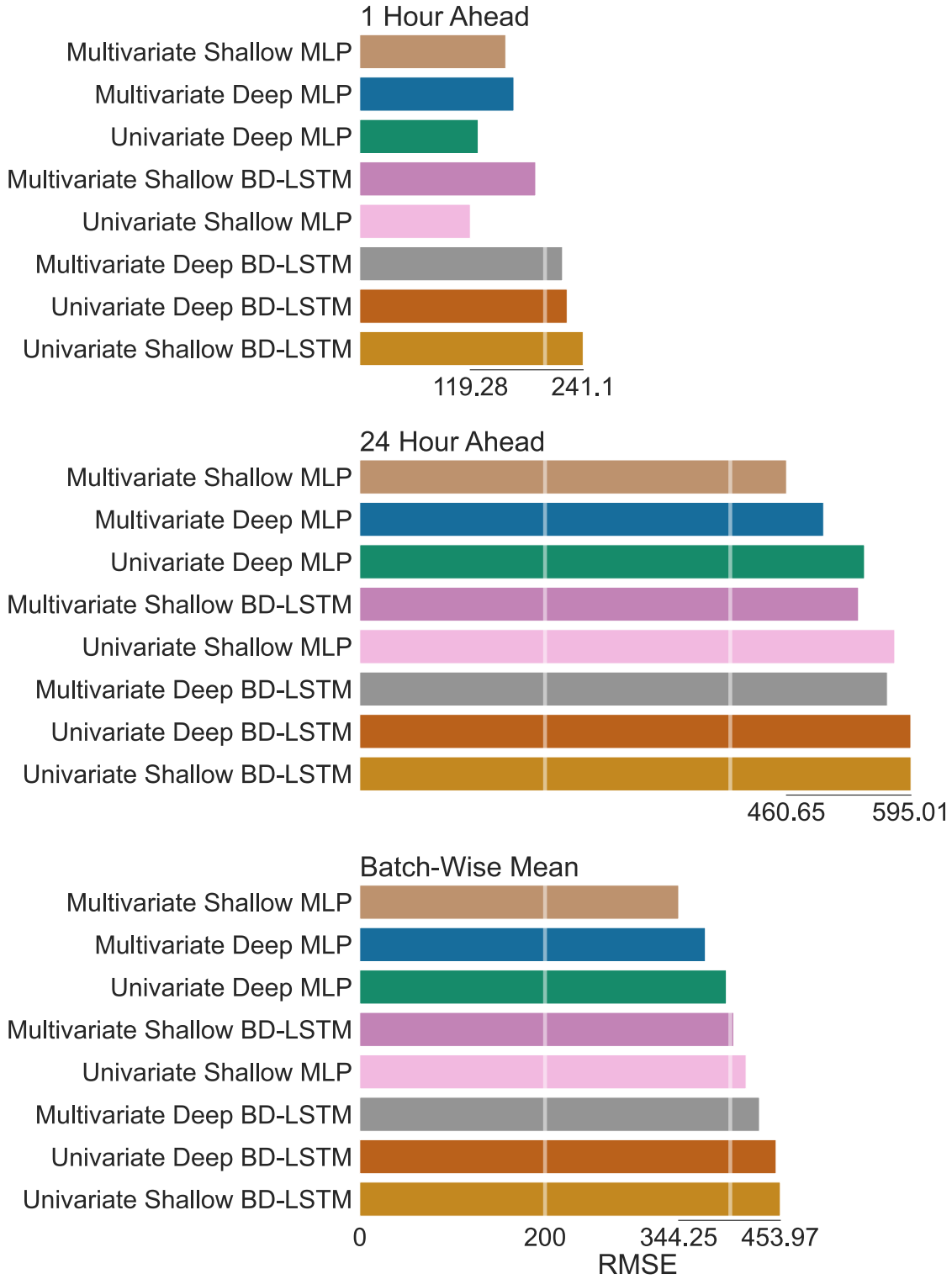

Figure 5.2: Horizontal bar charts with range frames showing the RMSE test results for the eight models. The tests covered the one- and 24-hour ahead prediction horizons and their batch-wise mean. The models are ordered by the mean.

Model Group	1 Hour	24 Hour	Mean
BD-LSTM	5.65	3.25	8.05
MLP	4.66	2.19	7.14

Table 5.5: Mean MAPE results grouped by model group.

Model Group	1 Hour	24 Hour	Mean
BD-LSTM	434.63	218.41	574.46
MLP	382.50	142.73	520.92

Table 5.6: Mean RMSE results grouped by model group.

Complexity	1 Hour	24 Hour	Mean
Deep	5.22	2.78	7.66
Shallow	5.10	2.67	7.53

Table 5.7: Mean MAPE results grouped by model complexity.

Complexity	1 Hour	24 Hour	Mean
Deep	412.41	184.19	552.48
Shallow	404.72	176.95	542.90

Table 5.8: Mean RMSE results grouped by model complexity.

Input Type	1 Hour	24 Hour	Mean
Multivariate	5.01	2.77	7.26
Univariate	5.30	2.68	7.93

Table 5.9: Mean MAPE results grouped by model input type.

Input Type	1 Hour	24 Hour	Mean
Multivariate	388.11	183.15	517.33
Univariate	429.02	178.00	578.06

Table 5.10: Mean RMSE results grouped by model input type.

5.2 Learning Curve Analysis

The learning curves of the MLPs (Figure 5.3) overall show that the models fit the data well. The exception is the deep multivariate model: its comparatively chaotic test curve is unusual. Instability in the test curve could result from an insufficiently large test set. However, the test set was enormous. This hypothesis would also fail to explain the instability in the train curve. We hypothesise that this has resulted from the learning rate scheduling. The larger learning rate in the schedule may have been too great, such that it deteriorated the coefficients much of the time. Additionally, both the train and test values were decreasing when the training stopped, indicating that this model may be underfitting the data. Additional training would improve this. Another point of potential concern for model misspecification is that the test MAPE of the multivariate shallow MLP increases for the final 8000-9000 epochs. However, as the models only saved the weights of the iteration performing best on the test set, the final model tested would not suffer from the overfitting evident in this plot.

The learning curves of the BD-LSTMs (Figure 5.4) show that the models are generally, if not entirely, underfitting the data. A well-fitted model will decrease its loss function rapidly before the loss largely stabilises, and it may gradually improve after that point for a large number of epochs. None of these models could have undergone the gradual improvements, and it is unclear that the univariate deep model has completed the initial stabilisation. Subsequently, the ranking and comparisons made in Section 5.1 are, at best, inconclusive. However, given that the train and test curves move in tandem, the models do not show signs of more pervasive underfitting or overfitting. As such, additional training should be sufficient to correct the underfitting present and make meaningful comparisons between the models [40].

Figures 5.3 and 5.4, at least tentatively, also show that all models are substantially biased compared to the AEMO Market Management System forecasts. Although we cannot be sure that further training of the LSTM and multivariate deep MLP models would not result in substantial improvements in the loss, the results of this study do not make that outcome appear likely. Further research and experimentation would likely be needed to improve upon AEMO's model.

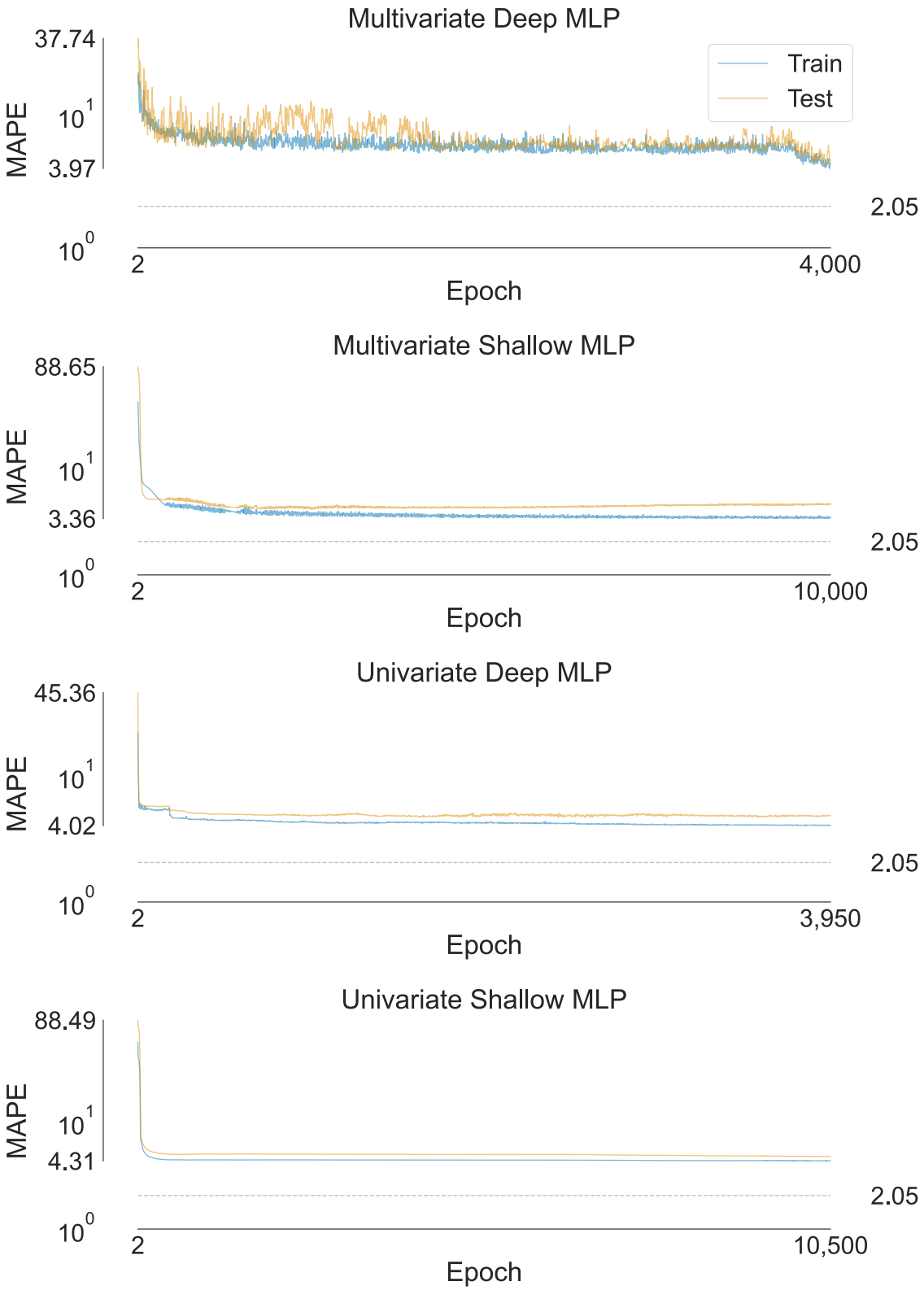

Figure 5.3: Training histories for each MLP showing the training and test loss (MAPE) by epoch of training. Only even epochs are shown, as the learning rate scheduling makes the full history appear noisier than it is. The dotted line represents the equivalent MAPE of the AEMO Market Management System model. The y-axis has a logarithmic scale.

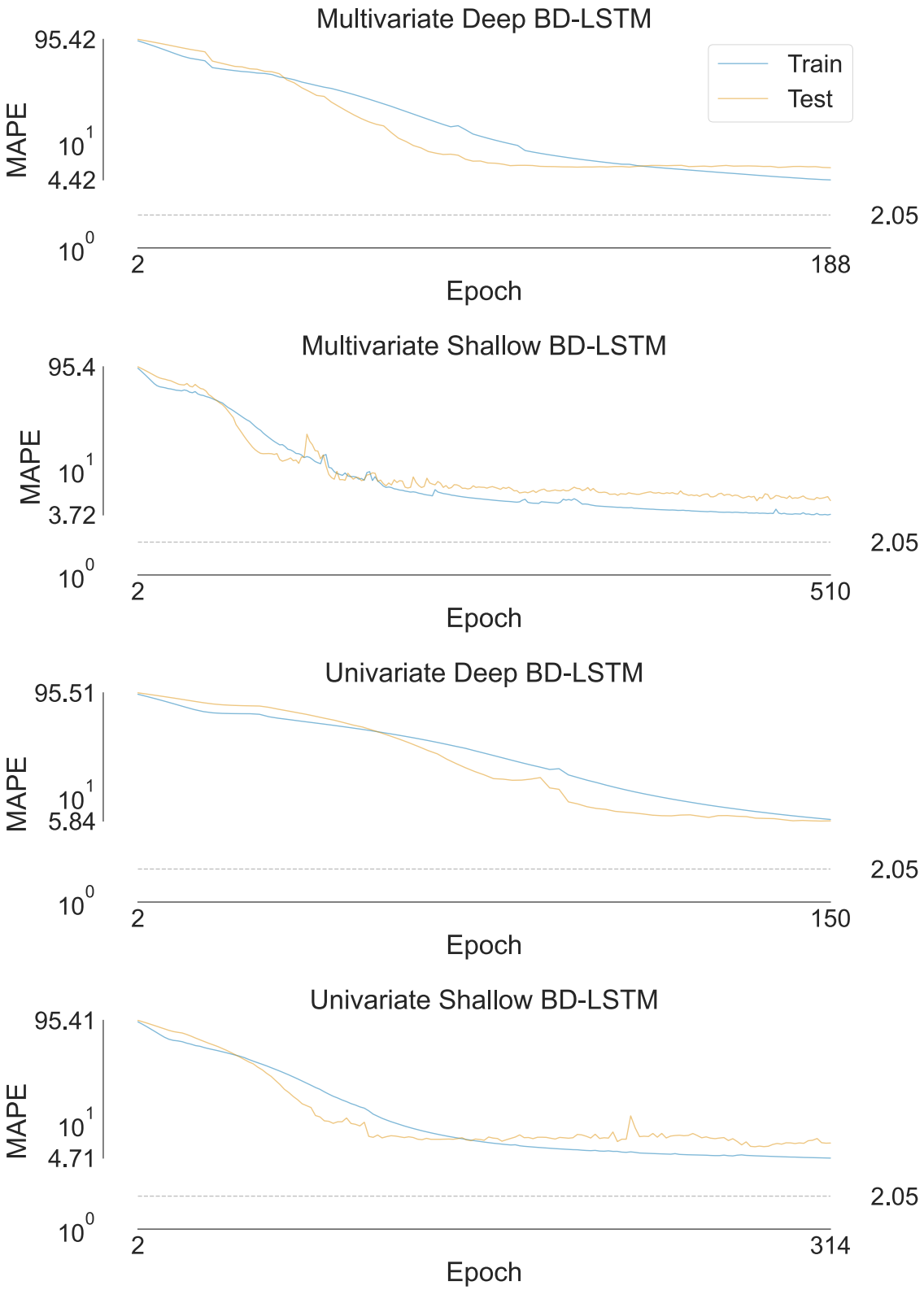

Figure 5.4: Training histories for each BD-LSTM showing the training and test loss (MAPE) by epoch of training. Only even epochs are shown, as the learning rate scheduling makes the full history appear noisier than it is. The dotted line represents the equivalent MAPE of the AEMO Market Management System model. The y-axis has a logarithmic scale.

Chapter 6

Discussion

The performance results for the eight models tested are given in Tables 5.2 and 5.4. The MLPs performed markedly better than the BD-LSTMs, with all MLPs outperforming all BD-LSTMs by the mean metric, with the one exception of the multivariate shallow BD-LSTM having a lower mean RMSE than the univariate shallow MLP. At first glance, this is surprising, given the added complexity of the BD-LSTM networks, with their recurrent layer designed to capture long-term dependencies in time-series data, compared to a feed-forward MLP. If we examine the loss curves in Figure 5.4, we see that, for the BD-LSTM networks, the curves were only just approaching stabilisation, particularly for the deep networks, and still had the potential to improve given more training time. The MLP networks, alternatively, had stabilised early in the training and easily reached their peak performance. Had more time been available to train these models, based on the research surveyed in Chapter 2, we would expect the BD-LSTM to outperform the MLP networks. The time constraint of this project is therefore a defining factor in the outcome of the comparison between model architectures. However, many other conclusions can be drawn.

In particular, the increased computational load imposed by using deep and complex NN models dramatically affects training time. This effect is demonstrated in Table 3.7, which shows that in a similar time period of around two hours, the most complex model (the multivariate deep BD-LSTM) was only able to complete 21 epochs, whereas the simplest model (the univariate shallow MLP) was able to complete 10,500. The former model was run for 189 epochs in this study over nearly ten hours, and while it continued to show signs of improvement when training ceased, ultimately, it was outperformed by the simpler model on nearly all metrics[1]. In fact, the simple model performed best out of all models for the 1-hour-ahead prediction, with a MAPE of 1.78 and an RMSE of 119.28, compared to the multivariate deep BD-LSTM which ranked 6th for 1-hour-ahead predictions, with MAPE of 3.38 and RMSE of 218.85.

Apart from more training time, another improvement that could contribute to the inherent advantage of more complex models is additional time for hyperparameter tuning and testing of model configuration. With time being a significant constraint, a compromise had to be made when selecting hyperparameters, with a grid search performed on the shallow version of each network and the results applied to both shallow and deep networks. Consequently, each model was not optimised. Ideally, each model architecture should have its own set of hyperparameters that

[1]The single exception to this is that the multivariate shallow BD-LSTM had a marginally lower RMSE on 24-hour-ahead forecasts than the univariate shallow MLP

were determined by more sophisticated means, as outlined in Chapter 2. An optimised architecture and hyperparameter set may have helped to get the most out of the BD-LSTM models.

Tables 5.2 and 5.4 show that in three out of the four model pairs, the multivariate model outperformed the univariate model, on average, in terms of both MAPE and RMSE. The exception was the deep MLP, where the average MAPE for the univariate case was slightly better (4.64 compared to 4.70). However, for the RMSE, the reverse was true: 395.71 compared to 373.06. So, in general, we have shown that multivariate models have outperformed their univariate equivalents (with the same architecture and hyperparameters). We expected this outcome, as more information is being fed into the models, allowing them to learn more complex relationships between attributes. The downside of adding more features is the increase in training time, as evident in Table 3.7. However, this effect was far more pronounced in the BD-LSTM variants than for the MLPs, where the differences were marginal.

Comparing the NN pairs by complexity (i.e. shallow vs. deep network), the results were inconclusive as to whether a deeper architecture is likely to provide higher accuracy. In both multivariate pairs (MLP and BD-LSTM), the shallow network outperformed the deep network, on average. While most differences between pairings were small, the multivariate BD-LSTM pair had the most significant difference in performance, with the shallow network's MAPE average being 5.07, compared to the deep network's 5.68. In both univariate pairs, the deep networks performed better. With the MLP networks able to train long enough to reach their optimal performance, we can conclude that, for the chosen architectures and hyperparameters, there is no clear advantage to increasing model depth. The computationally expensive BD-LSTM networks required more training time to reach their optimal performance, especially the deep variant. As such, no definitive conclusion can be drawn from these results.

Another factor that has impacted results is the differing load profiles over each state, seen in Figure 4.1. Combining the datasets from each state has resulted in more generalised models, but models may be improved by training only on data specific to each state. We recommend that state-specific models be tested and compared to the general case. Further, our imputation method was insufficiently robust to handle large stretches of missing data from the Victorian dataset. Issues with imputation resulted in a large cluster of values skewing the results. Better imputation should be implemented in future tests.

Further improvements could come from more sophisticated temperature or weather data use. While the electrical load data was supplied for the entirety of each state, the temperature observations are from a single station in each state and may not be representative of the temperature across large and varying Australian states and their climates. It is recommended that temperature data across all weather stations be obtained and an average temperature value be used, possibly weighted by population density around each weather station, as electrical load heavily correlates with population.

Additionally, the relatively biased results presented here were surprising, given the results seen as we designed the models. This disparity between expectation and outcome may partly result from overfitting the training and validation data. However, we hypothesise that there would be additional benefits to training one

model per state. Given that Australia is large and includes multiple climatic regions, it is plausible that the data used in this report and its patterns vary more across states than our models could manage. Future research could review whether one model per state or climatic region provides greater accuracy than one model for all regions.

Chapter 7

Conclusion and Further Issues

Overall, we recommend further testing to optimise the BD-LSTM networks and ensuring sufficient training time to obtain results in line with the research outlined in Chapter 2. It remains plausible that these changes would create a model superior to the AEMO Market Management System model. It would also be worth further investigating and comparing the performance of a 1D-CNN model, a relatively new approach to STLF that has shown promise, and a GRU, commonly used for this kind of task. With that said, good performance was able to be achieved using very simple NN architecture, with the top performing model, on average, being the multivariate shallow MLP.

This study has shown that multivariate modelling has an advantage, that the addition of more exogenous variables, such as solar radiance, wind, and solar panel uptake, could enhance model accuracy, and that the feature engineering implemented was worthwhile.

We could not draw a clear conclusion regarding the potential advantage of increasing model depth, as our results on this were mixed. However, deeper models require more training time, and the models presented here may have fallen substantially below their full potential in this test due to time constraints. The BD-LSTM cases, in particular, could improve with lesser time constraints. More work should be done to investigate this matter.

While training time is an essential factor to consider and a major differentiator between the models compared in this study, it is time that only needs to be spent in the development phase. Once the model is trained, it can be implemented without significant additional cost. Therefore the model producing the best prediction accuracy should be chosen, providing it can be tuned and trained within the client's time frame.

References

[1] T. Yalcinoz, U. Eminoglu, Short term and medium term power distribution load forecasting by neural networks, Energy Conservation and Management 46 (9=10) (2005) 1393–1405.

[2] M. Djukanovic, S. Ruzic, B. Babic, D. Sobajic, Y.-H. Pao, A neural-net based short term load forecasting using moving window procedure, Electrical Power Energy Systems 17 (6) (1995) 391–397.

[3] K. Berk, A. Hoffmann, A. Muller, Probabilistic forecasting of industrial electricity load with regime switching behavior, International Journal of Forecasting 34 (2) (2018) 147–162.

[4] A. Kavousi-Fard, H. Samet, F. Marzbani, A new hybrid modified firefly algorithm and support vector regression model for accurate short term load forecasting, Expert Systems with Applications 41 (13) (2014) 6047–6056.

[5] Y. Dong, X. Ma, T. Fu, Electrical load forecasting: A deep learning approach based on k-nearest neighbors, Applied Soft Computing 99.

[6] A. Singh, S. Khatoon, M. Muazzam, An overview of electricity demand forecasting techniques, Network and Complex Systems 3 (2) (2013) 38–49.

[7] H. Hippert, C. Pedreira, S. R.C., Neural networks for short-term load forecasting: a review and evaluation, IEEE TRANSACTIONS ON POWER SYSTEMS 16 (1).

[8] M. Raza, A. Khosravi, A review on artificial intelligence based load demand forecasting techniques for smart grid and buildings, Renewable and Sustainable Energy Reviews 50 (2015) 1352–1372.

[9] R. Chandra, S. Goyal, G. Rishabh, Evaluation of deep learning models for multi-step ahead time series prediction, IEEE Access 9 (2021) 83105 – 83123.

[10] M. El-Hawary, Advances in electric power and energy systems : load and price forecasting, 1st Edition, Wiley, 2017.

[11] A. Hoori, A. Al Kazzaz, R. Khimani, Y. Motai, A. Aved, Electric load forecasting model using a multicolumn deep neural networks, IEEE Transactions on Industrial Electronics 67 (8) (2019) 6473–6482.

[12] T. Czernichow, A. Piras, K. Imhof, P. Caire, Y. Jaccard, B. Dorizzi, A. Germond, Short term electrical load forecasting with artificial neural networks, Engineering Intelligent Systems 2 (1996) 85–99.

[13] G. Dudek, Multilayer perceptron for short-term load forecasting: from global to local approach, Neural Computing and Applications 32 (2019) 3695–3707.

[14] I. Yazici, O. Beyca, D. Delen, Deep-learning-based short-term electricity load forecasting: A real case application, Engineering Applications of Artificial Intelligence 109.

[15] S. Wang, X. Wang, S. Wang, D. Wang, Bi-directional long short-term memory method based on attention mechanism and rolling update for short-term load forecasting, Electrical Power and Energy Systems 109 (2019) 470–479.
[16] L. Smith, A disciplined approach to neural network hyper-parameters: Part 1 - learning rate, batch size, momentum, and weight decay, Tech. rep., US Naval Research Laboratory (2018).
[17] L. Smith, No more pesky learning rate guessing games, Tech. rep., US Naval Research Laboratory (2015).
[18] L. Smith, Cyclical learning rates for training neural networks, 2017, p. 464–472.
[19] L. Smith, N. Topin, Super-convergence: Very fast training of neural networks using large learning rates, in: Artificial intelligence and machine learning for multi-domain operations applications, Vol. 11006, SPIE, 2019, pp. 369–386.
[20] B. Csaji, Approximation with artificial neural networks (2001).
[21] A. Gasparin, S. Lukovic, C. Alippi, Deeplearning for time seriesforecasting:the electricload case, CAAI Transactions on Intelligence Technology 7 (1) (2021) 1–25.
[22] D. Park, M. El-Sharkawi, R. Marks, L. Atlas, M. Damborg, Electric load forecasting using an artificial neural network, IEEE Transactions on Power Engineering 6 (1991) 442–449.
[23] D. Srinivasan, A. Liew, C. Chang, A neural network short-term load forecaster, Electric Power Systems Research, 28 (1994) 227–234.
[24] F. Mohammad, K. Lee, Y.-C. Kim, Short term load forecasting using deep neural networks, arXiv preprint arXiv:1811.03242.
[25] G. Dudek, Forecasting time series with multiple seasonal cycle using neural networks with local learning, in: Artificial Intelligence and Soft Computing, Springer Berlin Heidelberg, 2013, pp. 52–63.
[26] K. Chen, K. Chen, Q. Wang, Z. He, J. Hu, Short-term load forecasting with deep residual networks, IEEE Transmission Smart Grid 10 (2018) 3943–3952.
[27] G. Hinton, S. Osindero, Y. Teh, A fast learning algorithm for deep belief nets, Neural Computation 18 (2006) 1527–1554.
[28] H. Hewamalage, C. Bergmeir, K. Bandara, Recurrent neural networks for time series forecasting: Current status and future directions, International Journal of Forecasting 37 (1) (2020) 388–427.
[29] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural computation 9 (8) (1997) 1735–1780.
[30] G. Din, A. Marnerides, Short term power load forecasting using deep neural networks, International Conference on Computing, Networking and Communications (ICNC) (2017) 594–598.
[31] M. Schuster, K. Paliwal, Bidirectional recurrent neural networks, IEEE Transactions on Signal Processing 45 (11) (1997) 2673–2681.
[32] Gradient-based learning applied to document recognition, Vol. 86.
[33] Handwritten digit recognition with a back-propagation network, Vol. 86.
[34] X. Shi, Z. Chen, H. Wang, D.-Y. Yeung, W.-K. Wong, W.-C. Woo, Convolutional lstm network: A machine learning approach for precipitation nowcasting, Advances in neural information processing systems 28.

[35] H.-Z. Huai-zhi Wang, G.-Q. Li, G.-B. Wang, G.-P. Peng, H. Jiang, Y.-T. Liu, Deep learning based ensemble approach for probabilistic wind power forecasting, Applied Energy 188 (2017) 56–70.
[36] Proceedings of the 34 th International Conference on Machine Learning, Sydney, Australia.
[37] E. De Cian, E. Lanzi, R. Roson, The impact of temperature change on energy demand: A dynamic panel analysis, FEEM Working Paper No. 46.2007, University Ca' Foscari of Venice, Dept. of Economics Research Paper Series No. 04-2007.
[38] G. Hoque, A better way to handle missing values in your dataset: Using iterativeimputer (part i), Towards Data Science.
[39] R. J. Hyndman, G. Athanasopoulos, Forecasting: Principles and Practice, OTexts, 2018.
[40] A. Ng, Machine Learning Yearning, Andrew Ng, 2018.

Appendices

A.1 Code

This project was managed through *GitHub*. Our code and more information about our work can be found at Team K ZZSC9020 Capstone Project.

A.2 Data

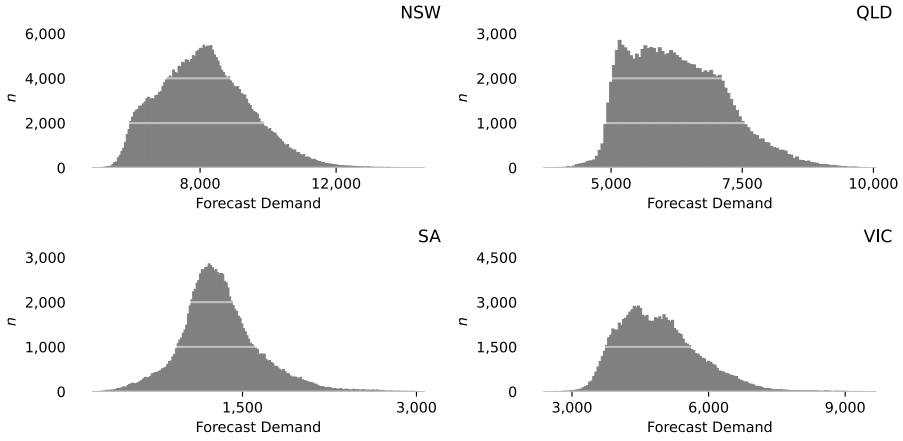

Figure A.1: A histogram of forecasted demand for NSW.

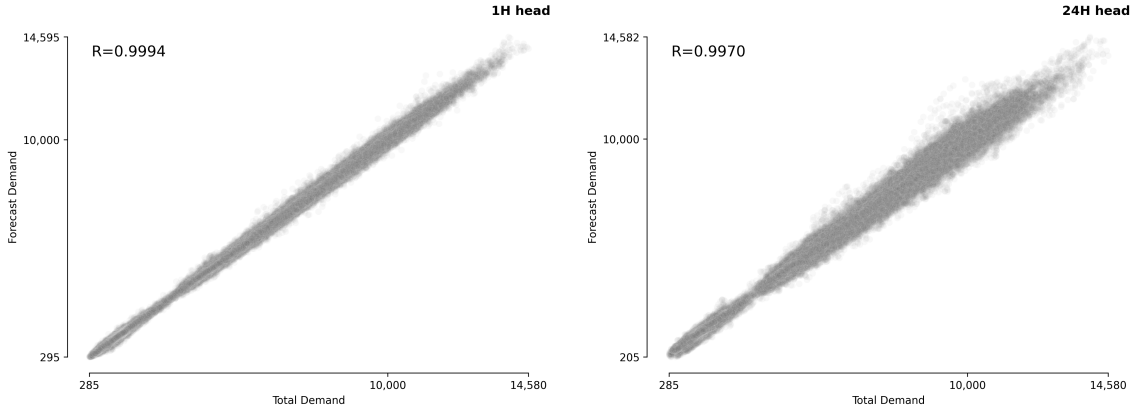

Figure A.2: Scatter plots of forecast vs. total demand for NSW. The one-hour-ahead (i.e. two-periods-ahead) correlation appears on the left, and the one-day-ahead correlation appears on the right.

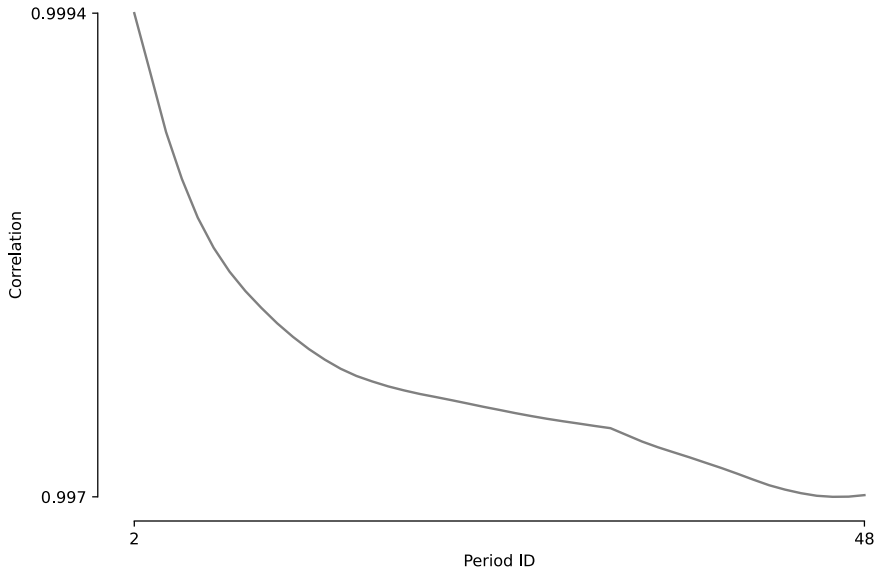

Figure A.3: A line plot of the Pearson correlation coefficient of forecast and total demand vs. the period ID. Each period represents a thirty-minute interval. As such, the X-axis also represents the correlations from one-hour-ahead to one-day-ahead forecasts.

A.3 Pre-processing Pipeline

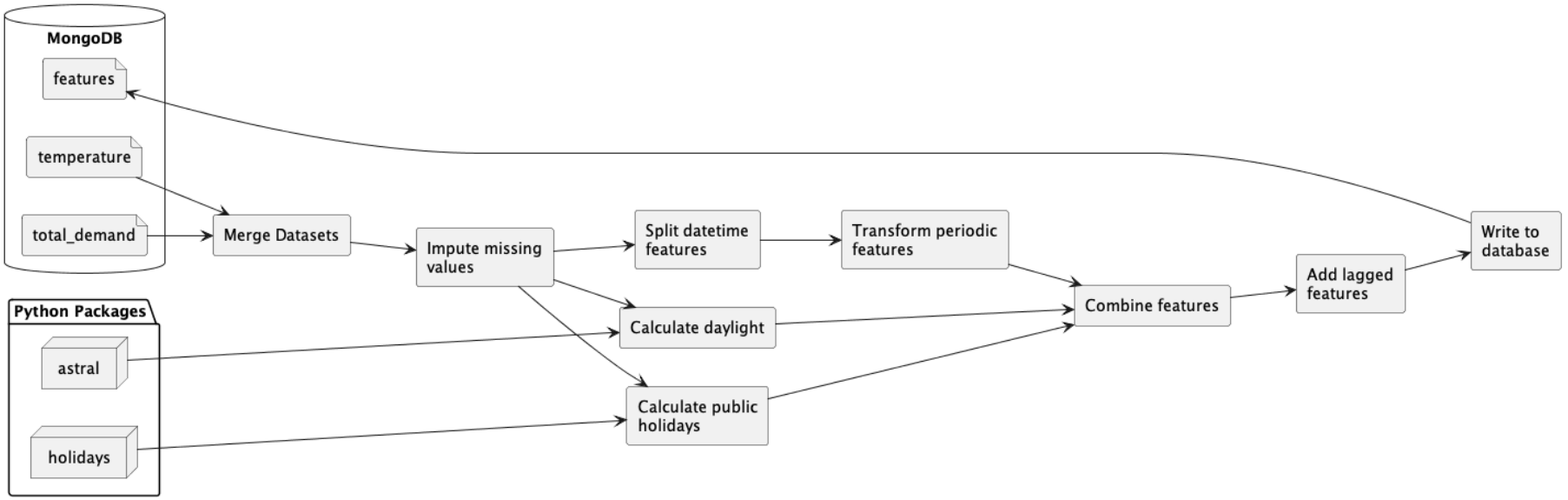

Figure A.4: Feature Engineering Pipeline.