DAY AHEAD LOAD FORECASTING FOR THE MODERN DISTRIBUTION NETWORK

Michael Jurasovic, 202924

October 2018

School of Engineering

This thesis is submitted in partial fulfilment of the requirements for the degree of Bachelor of Engineering with Honours, University of Tasmania.

Declaration

Abstract

Penetration of distributed energy resources in distribution networks is predicted to increase dramatically in the next seven years, bringing with it the opportunity for utilities to have a greater presence at low levels of the network. To achieve this effectively, utilities will require accurate short term load forecasts. This thesis presents several neural network-based load forecasting models that are applicable to forecasting both low and high levels of aggregate load. The models are simple to implement and require no tuning when used on different feeders.

One model is implemented as part of the Bruny Island CONSORT residential battery trial to produce a 24-hour horizon half-hourly online forecast every five minutes. When forecasting during anomalous peak holiday periods on a feeder that has a typical load of less than 1000kVA the forecasting system is able to successfully forecast major peaks with sufficient lead time and accuracy to enable the fleet of batteries to charge ahead of time and provide network support.

Acknowledgements

I would like to thank firstly my supervisors, Evan Franklin and Michael Negnevitsky, for their continued support, insight, and (often-ignored) suggestions throughout the year.

Secondly, Paul Scott for taking the forecasting system and implementing it as part of the battery trial, and for managing the weather forecast data.

Thirdly, Andrew Fraser, Laura Jones and the rest of TasNetworks for their guidance on requirements and assistance with data.

Contents

List of Figures

List of Tables

Nomenclature

The notation below is largely reproduced from Goodfellow et al. [1].

a	A scalar (integer or real)
$\boldsymbol{a}$	A vector
$\boldsymbol{A}$	A matrix
$\mathbb{R}$	The set of real numbers
a_i	Element i of vector $\boldsymbol{a}$, with indexing starting at 1
$A_{i,j}$	Element i, j of matrix $\boldsymbol{A}$
$\boldsymbol{A}_{i,:}$	Row i of matrix $\boldsymbol{A}$
$\boldsymbol{A}_{:,i}$	Column i of matrix $\boldsymbol{A}$
$\boldsymbol{A}^\top$	Transpose of matrix $\boldsymbol{A}$
$\boldsymbol{a}^\top$	Transpose of vector $\boldsymbol{a}$
$[\boldsymbol{x}_1 \dots \boldsymbol{x}_P] \in \mathbb{R}^{d\times P}$	Concatenation where $\boldsymbol{x} \in \mathbb{R}^d$
$\frac{dy}{dx}$	Derivative of y with respect to x
$\dfrac{\partial y}{\partial x}$	Partial derivative of y with respect to x
$\nabla_{\boldsymbol{x}} y$	Gradient of y with respect to $\boldsymbol{x}$
$f(\boldsymbol{x}; \boldsymbol{\theta})$	A function of $\boldsymbol{x}$ parametrized by $\boldsymbol{\theta}$.
$\sigma(x)$	Logistic sigmoid, $\dfrac{1}{1+\exp(-x)}$

Sometimes we use a function f whose argument is a scalar but apply it to a vector or matrix: $f(\boldsymbol{x})$, $f(\boldsymbol{X})$, or $f(\mathsf{\mathbf{X}})$. This denotes the application of f to the array element-wise. For example, if $\mathsf{\mathbf{C}} = \sigma(\mathsf{\mathbf{X}})$, then $\mathsf{C}_{i,j,k} = \sigma(\mathsf{X}_{i,j,k})$ for all valid values of i, j and k.

Chapter 1

Introduction

1.1 Background

Electricity distribution networks, and the way in which they are managed, are currently going through a significant transition, with perhaps more change over the last ten years than in the previous hundred. Until recently, generation and load were largely viewed and managed separately: power was produced almost exclusively by large, centralised generating units, and was consumed by customers after routing via the transmission and distribution networks. Networks were designed and built for demand profiles which were relatively stable over time, with demand forecasting at distribution feeder level required primarily for long term network planning purposes only. Increasingly, power is both consumed by customers connected to the distribution network and is also generated and manipulated by distributed energy resources (DER) within the distribution network, often behind the meter of individual consumers. The impact of increasing levels of DER in the system creates, among other things, both a need and an opportunity to more actively manage distribution networks, while also resulting in a generally less predictable net demand profile.

DERs are controllable devices in the power network that generate, store, and/or consume power. This includes solar photovoltaic generation (PV), battery storage, and electrical vehicles. In Australia, the dominant DER technology deployed to date is solar PV, with over 1.8 million systems now reportedly installed on residential properties [2]. It is widely anticipated that battery storage and electrical vehicle uptake may be just as rapid [3].

The Tasmanian distribution network, meanwhile, is forecast to experience significant increases in these technologies by 2025:

- 680% increase in battery storage capacity (from 11MWh to 75MWh) [4]
- 170% increase in PV installation capacity (from 130MW to 220MW) [4]
- 39% of new car sales will be electrical vehicles - the highest in the country [5]

The changing nature of the distribution network presents an opportunity to maximize the use of existing assets by delaying the need for network augmentations, while also providing customers with a more reliable supply of power. For example, batteries could be used to peak-shift, reducing maximum feeder load. However, to achieve this reliably and with optimal use of available DER generally requires sophisticated methods to optimize the power flow to and from the distributed resources.

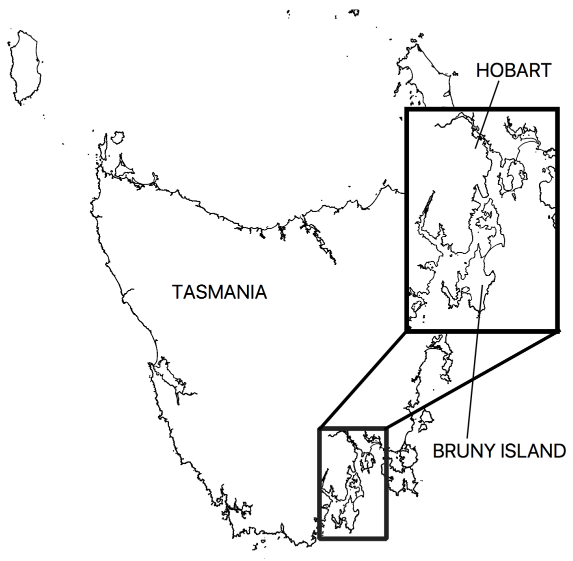


Figure 1.1: Bruny Island in southeast Tasmania experiences significant increases in load during holiday periods and was used as a case study for the load forecaster.

One method to achieve this coordination of DER in distribution networks is presented in [6] and has been implemented on Bruny Island, Tasmania as part of the CONSORT project (CONSumer energy systems providing cost-effective grid suppORT). Bruny Island, shown in Figure 1.1, is a popular holiday destination and during peak periods — such as Easter morning and afternoon peaks — the submarine feeder supplying the island becomes overloaded and is supplemented by a diesel generator located on the island. The aim of the CONSORT project was to effectively peak shift the load away from the morning and afternoon peaks to prevent the use of the generator.

To fulfil such an objective while using the available distributed resources optimally, the CONSORT project relies upon having an accurate, online, 24-hour horizon forecast at the feeder level. However, load forecasting methods commonly employed in industry are neither intended to forecast with high accuracy over a time period this short nor at the feeder level. [7]. An improved method for producing accurate feeder-level forecasts is not only highly desirable for this project, but will also in the future become a critical element of active distribution network management more generally.

In this thesis, a novel neural network-based day-ahead feeder level load forecasting system is developed, with its performance evaluated using ten years of historical demand data for training and testing. The forecaster is implemented live on Tasmania's Bruny Island distribution network, enabling the CONSORT project's residential battery systems to effectively support the network during periods of peak demand.

1.2 State of the art in load forecasting

1.2.1 Load forecasting - definition and general remarks

Before looking at state of the art load forecasting methods, it is important to first establish precisely what load forecasting is. Load forecasting is the process of predicting the electricity demand in a particular area at each period over a planning horizon [8]. For example, a load forecast with a period (or resolution) of one hour and a horizon of 24 hours would forecast the load at each hour over the next 24 hours. It is a specific case of time series forecasting. Load forecasting can be broadly categorised into short-term (STLF), medium-term (MTLF), and long-term load forecasting (LTLF). The exact horizon for each category differs between publications, but it is widely accepted that anything with a forecast horizon of up to 24 hours is short-term load forecasting. This thesis will deal exclusively with STLF. Generally load forecasting systems uses exogenous information as input, such as weather and date information, along with historical load data in order to produce a forecast.

What is clear throughout the literature is that there are a handful, perhaps a dozen or so, of clearly distinct techniques that are commonly applied to load forecasting. However, these methods are nearly always mixed, matched, and combined to form a complete forecasting system. As a result, there are a combinatorial number of approaches to forming a load forecasting system. This section will attempt to cover only the most prolific techniques, and the examples cited will generally also employ techniques which have not been discussed.

The literature presents no single load forecasting system that is superior under all conditions. A system that is well suited to forecasting load with a majority industrial customers may be poorly suited to forecasting load with a majority residential customers. This is because there is no single load forecasting problem - every household, feeder, and power system exhibits different patterns as a result of the different customers attached to it and thus presents a different problem. As a result, it is invalid to compare the performance of forecasting systems unless they are evaluated on the same datasets, making it difficult to establish a state of the art approach even at a similar aggregate level of load.

1.2.2 Load forecasting in industry

In section 1.1 it was stated that load forecasting methods commonly employed in industry neither intended to forecast with high accuracy over a 24 hour period nor at feeder level in the distribution network. This is supported by a survey conducted by CIGRE in 2016 [7] showing that, of 29 utilities, 79% use load forecasting for long term planning and 32% for short term operational planning. Interestingly, this survey also found that the majority of load forecasting software is developed in-house, and the methodology used is usually revised at least every four years.

The lack of attention to load forecasting at the feeder level is certainly echoed in the literature. The vast majority of published work considers forecasting at the city level and above, with a smaller portion looking at sub-GW load, and an even smaller portion looking at sub-MW load. Additionally, the publications looking at lower levels of aggregate load tend to be more recent, and are predominately published as conference papers rather than in peer-reviewed journals.

1.2.3 Patterns in load profiles

In a residential setting, such as a residential feeder, power is consumed as a result of customer actions - taking a shower causes the hot water system to consume power to reheat water, turning on a kettle or toaster causes power to be consumed, use of air conditioning results in power consumption, etc. It is intuitive to expect that there would be patterns underlying these actions or behaviours; customers probably take showers more often in the morning, and when it is cold customers are probably more likely to turn on their heaters. If the feeder in question happens to be in a popular holiday destination, then it might also be the case that overall more power is consumed during holiday periods as a result of more customers being in the area. Conversely, a feeder in a commercial area may experience a reduction in load over holiday periods because the workers are away. Any load forecasting system must be able to intrinsically model these underlying patterns if it is to effectively predict load.

Before looking at load forecasting techniques and systems, the typical patterns of load profiles will be discussed. It must be kept in mind though that all conclusions presented here are general; they will likely vary between countries, cultures, and feeders. Analysis of a specific feeder is undertaken in section 3.1.1.

Calendar day

The most significant factor contributing to a load profile is the calendar day [8]. Generally weekdays and weekends have differing load profiles, and even days during the week can be observed to have significantly different profiles. Meteorological seasonality usually plays a large role in load variation over a full year. This may be a direct result of day of the year, such as in the case of a commercial load that only operates over summer, or it may actually be a result of weather. Finally, holiday periods cause large deviations from standard profiles and can be difficult to predict due to the relatively few examples available. Holidays can also affect days adjacent to them as people may also take these days off work.

Weather

Second to calendar day, the most significant factor behind the load profile is weather [9]. Typically temperature and humidity are considered the most influential, as they drive customers to use heating and cooling which both consume large amounts of power. More recently, cloud cover and solar irradiance have become increasingly important as they affect rooftop photovoltaic (PV) generation [10]. In regards to PV, it may be further necessary to model its uptake - solar capacity is increasing in Australia [4] and so a forecasting system that relies on past examples to create future predictions may need to account for this.

1.2.4 Similar day and clustering

As load profiles are dependent primarily on calendar day and weather, a rudimentary load forecasting system can be implemented by simply finding a load profile from the past that has the most similar calendar day and weather properties to the period being forecast. Similarly, a clustering approach can be taken if it is assumed that load profiles will repeat themselves as a result of repeating weather and calendar day patterns.

Rahman and Hazim [11] used an expert-system based method to select a set of similar days from the past. These days were then modified by use of a lookup table based on differences in temperature and other exogenous factors between the similar and forecast period. Finally, all similar days were regressed to form a forecast. This method achieved a mean absolute percentage error (MAPE) of around 2% when forecasting at the level of a a single state of the United States with a 24-hour horizon.

Senjyu et al. [12] used a weighted sum-squared-error between the maximum temperature, minimum temperature, and day type of the period being forecast and previous days to select the most similar five days from the past. A fuzzy logic system was then used to generate weights to augment the similar days based on differences in weather and differences in load profile between the similar day and the day prior to the forecast day. This was found to out-perform a more rudimentary systems which only used average load of the similar days. The system achieved a MAPE of around 2.5% when forecasting hourly load of approximately 600MW with a 24-hour horizon. Interestingly, this system was further extended [13] to use a recurrent neural network instead of a fuzzy logic system and this resulted in a decrease in MAPE to 2.8%.

Dou et al. [14] used K-means clustering combined with variational mode decomposition and a self-adaptive evolutionary extreme learning machine to predict the load demand for renewable generation. The system assigns historical days to clusters and then finds the most similar cluster to the day being forecast and uses properties from that cluster to produce a forecast. The system was applied to forecasting photovoltaic power output, wind power output, and electrical load. The system achieves a mean absolute error of 10.8 MW when forecasting hourly load of around 60 MW with a 24-hour horizon.

1.2.5 Autoregressive integrated moving average

Autoregressive integrated moving average (ARIMA) models, introduced by Box et al. [15], are used widely for time series forecasting [8]. Generally, ARIMA models are used at the core of the forecasting system, and other methods are used in a supporting role. ARIMA-based models are generally considered to be complex and require a great deal of experience to effectively select appropriate parameters [16].

Taylor and McSharry [17] applied a seasonal autoregressive moving average (ARMA) model to forecast a variety of European loads, with means of between 3 and 50 GW. The method achieves an accuracy of approximately 1.75% when performing hourly forecasts with a 24-hour horizon. This was extended by Arora and Taylor [18] who used a rule-based system combined with the seasonal ARMA model to more accurately predict anomalous holiday periods. This rule based system dictates the intrayear seasonal cycle of the model, which is set such that it corresponds to both a matching anomalous day (Easter, Christmas, etc.) from a previous year and a matching day of the week from a previous year. Thus, the model will take historical data from the same anomalous day occurring on the same day of the week at some time in the past. This rule decreased the MAPE on anomalous days from approximately 5% to approximately 3% when evaluated on data from Great Britain with a typical load of around 30 GW.

Bennett et al. [19] used an ARIMAX model to forecast the properties of the next day's load (next day peak demand, next day morning peak, and next day total energy usage). These properties were then provided to a neural network which selected the appropriate cluster for the day, whose load profile mean was then modified to better match the predicted properties. The system achieves a MAPE of 12% when forecasting hourly loads of around 60 kVA over a 7-day horizon.

Karthika et al. [20] used an ARIMA model to perform a preliminary load forecast, which was then augmented by a support vector machine to account for non-linear exogenous variables such as temperature and day of week. The method achieved a MAPE of 4.15% on an hourly 24-hour horizon forecast.

Amjady [21] used an ARIMA model which was augmented by estimated forecasts from human power system operators. This additional information allowed the system to achieve a MAPE of approximately 1.75% when forecasting load on the order of 10 GW with hourly resolution and a 24-hour horizon. This compares to a MAPE of about 3% for a standard ARIMA model. This system has the downside of relying entirely upon a specialist expert to provide regular input information.

1.2.6 Support vector regression

Support vector regression (SVR) was introduced by Drucker et al. [22] and has been applied widely to short-term load forecasting. SVR maps an input feature vector into a higher dimensional space and applies linear regression.

Ceperic et al. [23] applied an ensemble of 24 SVR models to predict each hour of a 24-hour forecast. This was found to out-perform wavelet transform neural network methods [24][25], and echo state neural networks [26]. The SVR ensemble achieves a MAPE of 1.31% when forecasting hourly load with a 24-hour horizon on the ISO New England dataset. This load is on the order of 15 GW.

Elattar et al. [27] modified SVR such that during the training phase the more recent historical data had a larger influence on the model parameters. This approach achieved a MAPE of 3.62% when forecasting hourly load with a horizon between 16 and 88 hours on a dataset with a mean load of around 2.5 GW.

1.2.7 Artificial neural network

An artificial neural network (ANN) is a method for computation based loosely on biological brains [28]. ANNs can take on widely differing structures, and have have achieved state of the art performance in a wide range of fields including handwriting recognition [29], machine translation [30], speech recognition [31], and speech synthesis [32]. ANNs have been demonstrated to out-perform ARIMA models for electricity time series price forecasting [33]. For short-term time series vehicle traffic forecasting, ANNs have been shown to outperform both ARIMA and SVM (SVR) models [34].

Chen et al. [35] employed a wavelet neural network and similar day selection to perform a load forecast with a 24-hour horizon and 1-hour resolution. The load being forecast was on the order of 15 GW (ISO New England dataset) and the forecast was performed only once per day, at 9am. The authors calculated the wind chill temperature, based on ambient temperature and wind speed, and transformed it to have a linear relationship with load before using it as an input to the system.

Kong et al. [36][37] applied a long short term memory recurrent neural network for short term load forecasting of aggregate household load. The loads of 69 households were aggregated to form a load time series varying between 0 and 3000 kW. Forecasting this time series resulted in a MAPE of 8.59%. Additionally, the method was used to forecast individual household loads, achieving a MAPE of 44.39%. The paper does not state the forecast horizon or resolution. This method was found to out perform simpler multilayer perceptron models and clustering-only methods.

These results are achieved without the need to perform any data manipulation other than scaling between zero and one and using one-hot encoded data for date and holiday information. Additionally, the system requires only a single neural network model for forecasting at any time.

Ding et al. [38] applied a multilayer perceptron neural network to forecasting residential feeder level load with hourly resolution and a 24-hour horizon. The load was split into daily average power and intra-day load fluctuations, and each was forecast separately. Interestingly, their justification for use of a multilayer perceptron is that it is a universal function approximator. While true in theory [39], this statement pays no regard to the complexity of the neural network nor the methods by which the neural network parameters must be found. When evaluating the forecaster on a residential feeder with a mean load of around 50 kW a MAPE of 10.3% and mean absolute error (MAE) of 3.63 kW was achieved. When evaluating on a commercial feeder the system achieves a MAPE of 15.5% and a MAE of 75 kW - worse performance than the residential case despite the higher load. This discrepancy is put down to the residential feeder being the aggregate of many individual small independent customers, while the commercial feeder is an aggregate of only several customers. By the central limit theorem the residential feeder load at a given instant in time will more closely represent a normal distribution.

Hernández et al. [40] applied a multilayer perceptron to forecasting load in the 25 MW range. Sine and cosine functions were used to encode date information before supplying it to the neural network. The forecasting system achieves a MAPE of 2.4% when performing hourly forecasts with a 24-hour horizon. The paper presents a graph of MAPE versus forecast horizon, and notably the MAPE is not lower earlier in the forecast, as it is in most other papers.

Sun et al. [41] applied wavelet neural networks to forecast load in a distribution network in a hierarchical fashion. The substation level load was forecast first, then feeders, transformers, and finally individual customers. The forecasting system was evaluated on six feeders from the same substation, with mean loads ranging from 2.5 to 8 MW, and achieved MAPE scores between 3.5% and 5%.

1.3 Problem formation and scope

The aim of this thesis is to build upon existing research and develop a load forecasting system that is able to predict future load based on weather, holiday periods, and other exogenous factors. Bruny Island and the CONSORT project will be used as the primary case study. The forecasting system is expected to be equally applicable to any feeder or power system. Specifically, the system will have the following properties:

- The forecasting system will be designed to be practical to implement in industry; specifically it should not require specialised skills to implement and operate, and nor should it require significant modifications when forecasting different feeders.

- The forecasting system will produce a forecast with a horizon of up to 24 hours in the future in 30- and 60-minute intervals.
- The forecast will be able to begin from any point time.
- The forecasting system will predict load in kVA at each interval.
- The forecasting system will be aimed at predicting aggregate load at the feeder level. That is, between approximately 0.5 and 10MVA.
- The forecasting system will be especially aimed at predicting load during anomalous holiday periods.

Chapter 2

Analysis and methodology

This chapter presents all the implemented load forecasting models. The choice of each model is justified, and the architecture of each model is described with sufficient detail for it to be implemented.

2.1 Investigated models and methods

A time series forecasting model can generally be structured as a model that takes as input one or more matrices of historical and/or forecast data, and produces an output matrix of predictions. This structure is rather generic, and there are many models that can fulfil this requirement. However, for this thesis, only a handful of models have been selected for evaluation. Each model will be discussed briefly here, to provide the reader with a high-level overview, and then each model will be discussed in detail under its own heading in the following pages.

The seasonal autoregressive integrated moving average with exogenous factors (SARIMAX) model is an extension of the well known ARIMA [15] model which takes into account seasonal patterns and exogenous data. It has been around unchanged for decades and is used as a baseline for forecasting performance.

Sequence to sequence recurrent neural networks with long short-term memory cells gained notoriety in the field of neural machine translation in 2014 for achieving state of the art results with minimal tuning of the model [42]. The structure of this model is a natural fit for time series forecasting - neural machine translation maps one sequences of words (vectors) to another, just as a time series forecasting system maps a sequence of historical observations (vectors) to a sequence of predicted observations (also vectors). The simplicity and out-of-the-box performance of this model is extremely appealing.

The transformer neural network model was published in 2017, and produced state of the art results in neural machine translation using a fully attention-based architecture. This model is attractive because it may overcome the tendency for recurrent neural networks to "forget" data from early in the input sequence, while also having the ability to consider multiple information sub-spaces in the input.

The universal transformer model, published in July 2018, is a variant of the transformer, and claims to out-perform the transformer on algorithmic tasks. This model is included to assess its performance compared to the transformer.

Additionally, a similar profile selection method was developed to provide the models with relevant input data.

2.2 SARIMAX

The seasonal autoregressive integrated moving average with exogenous factors (SARIMAX) model is comprised of autoregressive (AR) and moving average (MA) terms. These will be described first and then used to build the SARIMAX model.

Consider a univariate time series $\boldsymbol{x} \in \mathbb{R}^T$, being drawn from some underlying process, with scalar elements $x_{t-1}, ..., x_0$. An autoregressive model predicts the next element of the series by

$$x_t = \alpha_1 x_{t-1} + \alpha_2 x_{t-2} + \ldots + \alpha_p x_{t-p} + \epsilon_t = \epsilon_t + \sum_{i=1}^{p} \alpha_i x_{t-i} \tag{2.1}$$

where p is the order of the AR model, α are the parameters of the model, and ϵ_t is a random normally distributed error with zero mean and variance σ^2.

A moving average model predicts the next element of the series by

$$x_t = \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + \ldots + \theta_q \epsilon_{t-q} + \epsilon_t = \epsilon_t + \sum_{i=1}^{q} \theta_i \epsilon_{t-i} \tag{2.2}$$

where q is the order of the MA model, θ are the parameters of the model, and ϵ_t is, again, a random normally distributed error with zero mean and variance σ^2.

The AR and MA models can be combined to form an ARMA model, given by

$$x_t = \epsilon_t + \sum_{i=1}^{q} \theta_i \epsilon_{t-i} + \sum_{i=1}^{p} \alpha_i x_{t-i} \tag{2.3}$$

An ARMA model requires the time series, $\boldsymbol{x}$, being forecast to come from a weakly stationary process. That is, the mean and autocovariance of the the process do not change with time. The ARIMA model is an extension to the ARMA model that can be applied when non-stationarity is exhibited by the underlying process. ARIMA applies differencing to the time series to produce $\hat{\boldsymbol{x}}$, and then applies the ARMA model. This differencing is applied such that $\hat{x}_t = x_t - x_{t-1}$ and is repeated d times. Differencing can sometimes be sufficient to make a series stationary, such that it is appropriate for forecasting with an ARMA model.

In order to concisely articulate the ARIMA model, the following notation will be used. Note that this notation overrides that defined in the Nomenclature section.

- The backward shift operator, defined by $\mathbf{B}^m x_t = x_{t-m}$.
- The difference operator, defined by $\nabla_s X_t = x_t - x_{t-s} = (1 - \mathbf{B}^s)x_t$.
- The parameter functions $\alpha_p(\mathbf{B}) = 1 - \alpha_1\mathbf{B} - \ldots - \alpha_p\mathbf{B}^p$ and $\theta_q(\mathbf{B}) = 1 + \theta_1\mathbf{B} + \ldots + \theta_q\mathbf{B}^q$.

Note that the difference operator can be raised to a power to allow repeated differencing, such that $\nabla_s^u X_t = [(1 - \mathbf{B}^s)^u]x_t$. Using this notation, the ARIMA model can be represented by the following expression

$$\alpha_p(\mathbf{B})(\nabla_1^d x_t) = \theta_q(\mathbf{B})\epsilon_t \tag{2.4}$$

This is identical to an ARMA model, except that x_t has been replaced with a version of itself, $\nabla_1^d x_t$, that has been differenced d times. The above model is denoted ARIMA(p, d, q).

Differencing with a lag of one is not always sufficient to make the series stationary even when performed multiple (d) times. A further extension of the ARIMA model is the seasonal ARIMA (SARIMA) model, which takes into account seasonal patterns which require differencing with a lag greater than one, such as daily, weekly, and yearly seasonalities commonly found in electrical load time series.
Denoted ARIMA$(p, d, q) \times (P, D, Q)s$, a SARIMA model can be represented as the following

$$\alpha_p(\mathbf{B})A_P(\mathbf{B}^s)(\nabla_1^d \nabla_s^D x_t) = \theta_q(\mathbf{B})\Theta_Q(\mathbf{B}^s)\epsilon_t \tag{2.5}$$

where (p, d, q) are the parameters of the non-seasonal ARIMA component, $(P, Q, D)s$ are the parameters of the seasonal component, $A_P(\mathbf{B}^s)$ and $\Theta_Q(\mathbf{B}^s)$ are defined similarly to $\alpha_p(\mathbf{B})$ and $\theta_q(\mathbf{B})$, and s is the number of periods over which the seasonality occurs (e.g. $s = 7$ for weekly seasonality if the data is daily).

The SARIMA model can be further extended to a SARIMA with exogenous variables (SARIMAX) model. Given a vector of exogenous inputs $\boldsymbol{z} \in \mathbb{R}^n$ corresponding to the exogenous inputs at time t, the SARIMAX model gives the next element of the series by

$$\alpha_p(\mathbf{B})A_P(\mathbf{B}^s)(\nabla_1^d \nabla_s^D u_t) = \theta_q(\mathbf{B})\Theta_Q(\mathbf{B}^s)\epsilon_t \tag{2.6}$$

$$x_t = u_t + \sum_{i=1}^{n} \beta_i z_i \tag{2.7}$$

where $\beta_1 \ldots \beta_n$ are a set of model parameters.

2.2.1 Implementation

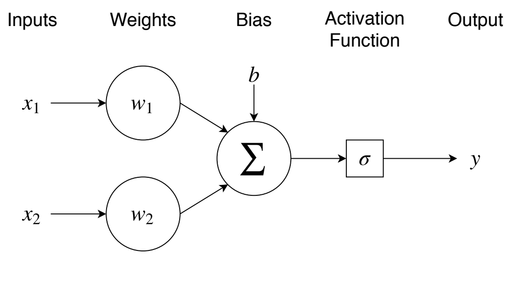
A SARIMAX model was implemented using the python statsmodels package [43]. The model is provided with historical load as the $\boldsymbol{x}$ input. For exogenous inputs, it is provided with future forecast temperature, holiday boolean, and holiday type. Holiday boolean and holiday type are discussed in section 2.8. Note that these inputs are different to the inputs provided to the neural network models.

2.3 Neural networks - a primer

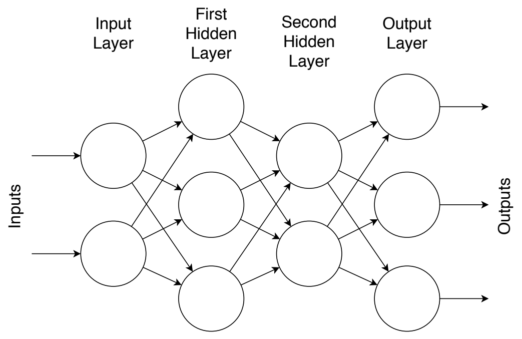
Neural networks will be described in a general sense here prior to discussion of the specific neural network architectures in the following sections. This section intends to establish the fundamental concepts that are applied when developing neural network models.

An artificial neural network (ANN) is a method for computation based loosely on biological brains [28]. An ANN is simply a function approximator; given a function $f^*(\boldsymbol{x})$, an ANN defines an approximation function $f(\boldsymbol{x};\boldsymbol{\theta})$ where $\boldsymbol{\theta}$ is a (usually learned) set of parameters that lead to the best approximation of f^* [1]. Generally, ANNs are formed by combining many fundamental sub-functions in a graph-like manner. A simple example is the multilayer perceptron. Consider a single perceptron, shown in figure 2.1(a). This single perceptron simply takes the weighted sum of all inputs plus a bias and applies an activation function, usually differentiable and highly non-linear, to produce an output y. A multilayer perceptron is formed by structuring individual perceptrons in layers and cascading them so that the outputs of the perceptrons in one layer are the inputs to the perceptrons in the next layer, as shown in figure 2.1(b) where the lines indicate connections from the output of one perception to the input of the next. The weights w are mapped to elements in the parameter vector $\boldsymbol{\theta}$.

The hidden layer can be repeated any number of times, allowing the neural network to be written in the form $f(\boldsymbol{x}) = f^n(f^{\cdots}(f^1(\boldsymbol{x})))$. This is a feedforward neural network, and is the only type of neural network this thesis will consider. In the case of recurrent neural networks discussed later in this thesis, where the outputs of sub-functions in the network form cycles, the network is unrolled to form a feedforward network.



(a) A single perceptron.



(b) A multi-layer perceptron, with each circle representing a single perceptron.

Figure 2.1: A multilayer perceptron is comprised of multiple individual perceptrons.

The set of parameters $\boldsymbol{\theta}$ is usually learned, or trained, by stochastic gradient descent [44] or a variant thereof. Consider an ANN producing an approximated output

$\hat{\boldsymbol{y}} = f(\boldsymbol{x};\boldsymbol{\theta})$ and a cost function $J(\boldsymbol{y},\hat{\boldsymbol{y}}) = J(\boldsymbol{y},\boldsymbol{x},\boldsymbol{\theta})$, with J being a differentiable function of its inputs, $\boldsymbol{y}$ being the correct output of f^*, and $\boldsymbol{\theta}$ being randomly initialized. The cost function produces a scalar describing the error between $\boldsymbol{y}$ and $\hat{\boldsymbol{y}}$. A common example of this is sum squared error, where $J(\boldsymbol{y},\hat{\boldsymbol{y}}) = \sum_{n=o}^{i}(|\boldsymbol{y}_n - \hat{\boldsymbol{y}}_n|^2)$. The gradient of J with respect to the parameters is given by $\nabla_{\boldsymbol{\theta}}J(\boldsymbol{y},\boldsymbol{x},\boldsymbol{\theta})$ and indicates the direction in which to move $\boldsymbol{\theta}$ in order to increase J. By iteratively applying equation 2.8, where ϵ is a scalar constant referred to as the learning rate, the value of $\boldsymbol{\theta}$ will be modified such that $J(\boldsymbol{y},\boldsymbol{x},\boldsymbol{\theta})$ is minimized - thus maximizing the approximation accuracy of f as measured by J. The action of applying equation 2.8 a single time is commonly referred to as taking a single training step, operation or iteration.

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \epsilon\nabla_{\boldsymbol{\theta}}J(\boldsymbol{y},\boldsymbol{x},\boldsymbol{\theta}) \tag{2.8}$$

In stochastic gradient descent, $\boldsymbol{x}$ and $\boldsymbol{y}$ are a randomly selected pair from the training set and so over many training iterations the entire training set will be approximately uniformly sampled. Typically, though, mini-batch gradient descent is used rather than stochastic gradient descent, whereby the training step is batched – multiple pairs of $\boldsymbol{x}$ and $\boldsymbol{y}$ are used to average the loss in parallel before performing an update on the weight vector. The number of pairs of $\boldsymbol{x}$ and $\boldsymbol{y}$ per batch is commonly referred to as the batch size. Mini-batch gradient descent is generally able to leverage parallel computation features of modern microprocessors to make it vastly faster than calculating J multiple times and taking the average.

Whether the loss minimization process lands on a global or local minimum is dependant on the function J and the specifics of the gradient descent implementation. Using a smaller batch size will introduce noise into the loss function, making it more likely for the optimization process to escape local minima, while using a larger learning rate might allow the training operation to "jump" out of local minima. Of course, both of these have downsides too – arguments for larger batch size and smaller learning rate can also be made. Overall, the training process is complex and nuanced. The training processes used in this thesis are discussed in section 2.7.

2.4 Recurrent neural network

The recurrent neural network (RNN) architecture was introduced by Rumelhart et al. [45] in 1986. Over the past five years RNN-based neural network models have achieved state of the art performance – though in many cases have since been outperformed by more obscure models – in speech recognition [31], neural machine translation [46], image captioning [47], and time series classification [48]. Over this same time span RNN-based neural networks have been used to forecast electrical load at the household-level [36][49], building level [50], feeder-level [51], and city/state level [52][53]. RNN-based models appear to offer the ability to forecast load with minimal preprocessing of the input data and so may be a very attractive option for practical load forecasting systems.

2.4.1 Generic recurrent neural network

RNNs operate on sequences of data by applying an identical function f to every element of the sequence to produce a sequence of state vectors $\boldsymbol{H} = [\boldsymbol{h}_1 \dots \boldsymbol{h}_P]$. The function f takes as input both the output of the function at the previous point in the sequence, and the current element of the input sequence $\boldsymbol{X} = [\boldsymbol{x}_1 \dots \boldsymbol{x}_P]$. An RNN can be unfolded to represent it as a traditional feedforward neural network with no recurrence. A generic RNN is shown in figure 2.2(a).

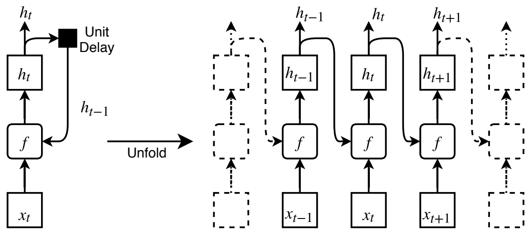
At each point t in the sequence $\boldsymbol{h}_t$ is taken as the output. $\boldsymbol{h}_t$ can be projected to the required output dimension – for example by equation 2.15, but any method can be used. An output dimension of 1 is typical for univariate load forecasting.

RNNs can be extended to multiple layers, with a two-layer RNN illustrated in figure 2.2(b). Each layer implements its own function, with layer n implementing function f^n and producing output $\boldsymbol{h}^n$. For layers after the first layer, instead of taking an $\boldsymbol{x}$ vector as input they take the state vector from the previous layer. There can be an arbitrary number of layers. A multi-layer RNN can in fact be thought of as a generalization of a single-layer, where the function f is simply comprised of multiple sub-functions.

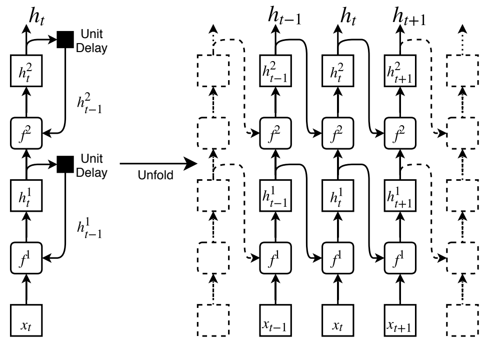
As an RNN simply applies the same function to each element of the input sequence its computation and memory requirements are $\mathrm{O}(n)$ with the input sequence length.

2.4.2 Long short-term memory

RNNs are prone to issues with vanishing and exploding gradients [1], which can be addressed through the use of a long short-term memory (LSTM) [54] cell as the function in the RNN. An LSTM cell is able to selectively add and remove elements from the state vector as it passes through the cell. LSTM cells have been demonstrated to outperform simpler functions in a variety of tasks when employed as the RNN function [55][56][57]. The LSTM cell extends the standard RNN model by separating the cell output $\boldsymbol{c}_t$ from the cell state $\boldsymbol{h}_t$, shown in figure 2.3(a), and its multi-layer configuration shown in figure 2.3(b). The LSTM cell is described by



(a) A generic recurrent neural network.



(b) A two-layer recurrent neural network.

Figure 2.2: Recurrent neural networks in single- and multi-layer configurations, with both recurrent and unrolled feedforward representations shown.

the following equations

$$\boldsymbol{f}_t = \sigma(\boldsymbol{W}_f \boldsymbol{x}_t + \boldsymbol{U}_f \boldsymbol{h}_{t-1} + \boldsymbol{b}_f) \tag{2.9}$$
$$\boldsymbol{i}_t = \sigma(\boldsymbol{W}_i \boldsymbol{x}_t + \boldsymbol{U}_i \boldsymbol{h}_{t-1} + \boldsymbol{b}_i) \tag{2.10}$$
$$\boldsymbol{o}_t = \sigma(\boldsymbol{W}_o \boldsymbol{x}_t + \boldsymbol{U}_o \boldsymbol{h}_{t-1} + \boldsymbol{b}_o) \tag{2.11}$$
$$\hat{\boldsymbol{c}}_t = \tanh(\boldsymbol{W}_c \boldsymbol{x}_t + \boldsymbol{U}_c \boldsymbol{h}_{t-1} + \boldsymbol{b}_c) \tag{2.12}$$
$$\boldsymbol{c}_t = \boldsymbol{f}_t \circ \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \circ \hat{\boldsymbol{c}}_t \tag{2.13}$$
$$\boldsymbol{h}_t = \boldsymbol{o}_t \circ \tanh(\boldsymbol{c}_t) \tag{2.14}$$

where

- $\circ$ denotes element-wise multiplication;
- the input vector $\boldsymbol{x}_t \in \mathbb{R}^N$;
- the cell output vector $\boldsymbol{h}_t \in \mathbb{R}^d$ (d is the hidden dimension of the model);
- the cell state vector $\boldsymbol{c}_t \in \mathbb{R}^d$;
- the forget gate vector $\boldsymbol{f}_t \in \mathbb{R}^d$;
- the input gate vector $\boldsymbol{i}_t \in \mathbb{R}^d$;

- the output gate vector $\boldsymbol{o}_t \in \mathbb{R}^d$;
- the cell candidate vector $\hat{\boldsymbol{c}}_t \in \mathbb{R}^d$;
- the learned cell state weight matrices $\boldsymbol{U} \in \mathbb{R}^{d \times N}$;
- the learned input weight matrices $\boldsymbol{W} \in \mathbb{R}^{d \times d}$;
- the learned bias vectors $\boldsymbol{b} \in \mathbb{R}^d$; and
- $\sigma(\boldsymbol{y}) = \frac{1}{1+e^{-\boldsymbol{y}}}$ represents the sigmoid function.

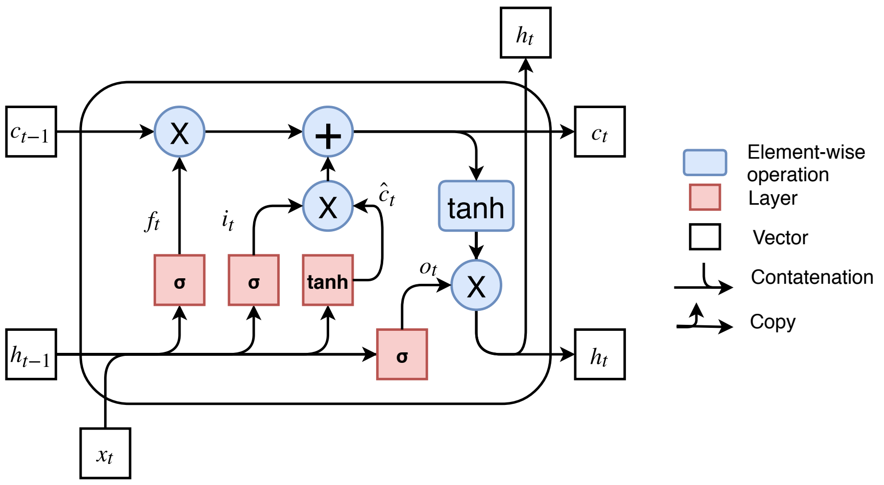
At first glance equations 2.9 through 2.14 may come across as esoteric, but they are in fact highly intuitive when studied alongside figure 2.3. There are three primary sub-sections within the LSTM cell, discussed in the three following paragraphs. Note that the use of two weight matrices ($\boldsymbol{W}$ and $\boldsymbol{U}$) in the LSTM equations is equivalent to concatenating $\boldsymbol{x}$ and $\boldsymbol{h}$ and using a single weight matrix - keeping consistency with figure 2.3.

First, data is removed from the cell state ($\boldsymbol{c}$) as it flows through the LSTM cell. The sigmoid function that produces $\boldsymbol{f}_t$ outputs values between 0 and 1. When these are multiplied with $\boldsymbol{c}_{t-1}$ (the previous cell state) some elements will be removed or reduced. The sigmoid layer that produces $\boldsymbol{f}_t$ is referred to as the forget gate, as it causes some data to be “forgotten” from the cell state depending on the previous cell output ($\boldsymbol{h}_{t-1}$) and the current input ($\boldsymbol{x}_t$).

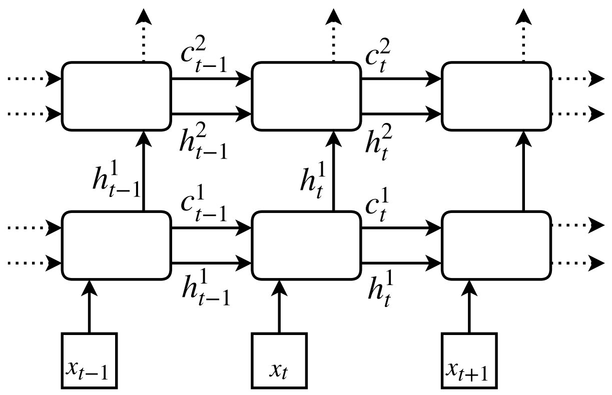
Next, information is added to the cell state. The tanh layer produces a new candidate cell state ($\hat{\boldsymbol{c}}_t$), and the input gate layer produces $\boldsymbol{i}_t$ in a similar fashion to $\boldsymbol{f}_t$. $\boldsymbol{i}_t$ decides which elements of the new candidate cell state vector are added to the actual cell state vector by reducing the magnitude of elements in $\hat{\boldsymbol{c}}_t$ based on the previous cell output ($\boldsymbol{h}_{t-1}$) and the current input ($\boldsymbol{x}_t$). After $\hat{\boldsymbol{c}}_t$ has been multiplied by $\boldsymbol{i}_t$ to selectively reduce elements it is added to the cell state. The sigmoid layer that produces $\boldsymbol{i}_t$ is referred to as the input gate, as it decides what data is added to the cell state as it passes through the cell.

Finally, a candidate output vector is produced by applying tanh element-wise to the cell state. The output gate layer produces $\boldsymbol{o}_t$ and this is multiplied with the candidate output vector to selectively remove elements, producing the final cell output $\boldsymbol{h}_t$. The sigmoid layer that produces $\boldsymbol{o}_t$ is referred to as the output gate, as it decides which elements of the candidate output are passed as the final output.

Together, these gating mechanisms allow the LSTM cell to selectively add and remove elements from the state as it passes through the cell, and allow the cell to selectively produce an output based on the current cell state. This generally allows the LSTM cell to retain data from many timesteps in the past, making it a superior choice to simpler functions when working with long sequences of data, though in some cases the LSTM cell can discard or “forget” data from early in the sequence.



(a) A long short-term memory cell.



(b) Multi-layer long short-term memory cell configuration.

Figure 2.3: An LSTM cell and it's multi-layer configuration in an RNN.

2.4.3 Sequence to sequence recurrent neural network

An RNN can be extended to form a sequence to sequence (S2S) architecture by applying an encoder RNN to an input sequence, and then having a decoder RNN produce an output sequence. The decoder RNN's initial cell state is set to the final cell state of the encoder [58]. This shared cell state is a latent representation of the input, which is used by the decoder to produce an output. The encoder and decoder can be of different lengths, depicted in figure 2.4 as R and P, allowing this architecture to naturally model tasks such as load forecasting where the input may consist of 48 hours of data while the output is only 24. The output matrix $\boldsymbol{H} \in \mathbb{R}^{R\times d}$ is then projected to the desired dimension.

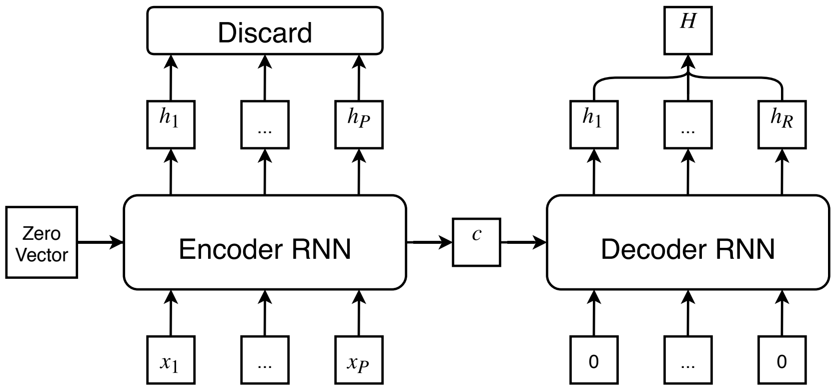


Figure 2.4: A sequence to sequence recurrent neural network architecture.

2.4.4 Implementation

A S2S RNN with LSTM cells was implemented in tensorflow [59], building on an implementation by Chevalier [60]. The encoder was provided with historical load and weather, and forecast weather. The projected decoder output was used as the load forecast. Teacher forcing was not used, as discussed later in section 2.5.7.

The input vectors, $\boldsymbol{x}_i$, are from the input matrix $\boldsymbol{X} \in \mathbb{R}^{T\times N}$, where $\boldsymbol{x}_i = (\boldsymbol{X}_{i,:})^\top$, T is the sequence length of the time series input, and N is the number of observations at each point in the time series.

The decoder output, $\boldsymbol{H} \in \mathbb{R}^{U\times d}$, was projected to a dimension of 1 using equation 2.15, where $\boldsymbol{W} \in \mathbb{R}^{d\times 1}$ is a learned weight matrix and $\boldsymbol{b} \in \mathbb{R}^1$ is a learned bias vector, U is the length of the time series forecast, and d is the hidden dimension of the model.

2.5 Transformer

The transformer neural network architecture, shown in figure 2.5, was introduced by Vaswani et al. [30] in 2017 and at the time was the state of the art in neural machine translation. It has since been outperformed by a variant of itself, the universal transformer [61], which is discussed in section 2.6. A transformer-based neural network has been applied to time series classification and regression in a medical context, achieving state of the art results [62]. The transformer has been extended to the image transformer model, which achieves state of the art results in image generation tasks [63]. To the best of the author's knowledge, there are no publications which apply the transformer – or any similar architecture – to electrical load forecasting.

The transformer suffers from no issues with "forgetting" elements from early in the input sequence, such as is possible in an LSTM RNN, as the transformer uses a multi-head attention mechanism to look over all elements of the input sequence simultaneously. However, you never get ought for naught, and in this case the transformer computation and memory requirements are $\mathrm{O}(n^2)$ with the input and output sequence lengths due to the multi-head self-attention. This contrasts with the $\mathrm{O}(n)$ requirements of an LSTM RNN.

The transformer architecture follows a similar sequence-to-sequence/encoder-decoder architecture as the sequence to sequence RNN architecture described in section 2.4.3: the encoder transforms an input sequence $\boldsymbol{X} = [\boldsymbol{x}_1 \ldots \boldsymbol{x}_P]$ into a latent representation $\boldsymbol{Z} = [\boldsymbol{z}_1 \ldots \boldsymbol{z}_M]$, and the decoder transforms $\boldsymbol{Z}$ into an output sequence $\boldsymbol{Y} = [\boldsymbol{y}_1 \ldots \boldsymbol{y}_R]$. This aligns with the requirements of a load forecasting system, where P is the length of the time series input, $\boldsymbol{x}_t$ is of dimension equal to the number of variables in the input time series, R is the length of the time series output, and $\boldsymbol{y}_t$ is of dimension equal to the number of variables in the output time series. The latent representation $\boldsymbol{Z}$ is used solely by the model as an internal representation of the input data.

The encoder is constructed of a stack of L identical layers, each containing two sub-layers. The first is multi-head self-attention and the second is a position-wise feed-foward network. Both sub-layers have a residual connection around them and are fed into a normalization layer.

The decoder is similar to the encoder except for a third layer which implements multi-head attention on the final output of the encoder. The input to the decoder is the output of the decoder, but shifted right by one. This requires an iterative approach to be used to predict all points in the time series. The self-attention in the decoder is masked so that when evaluating a query at time t it does not assign large weights to keys/values occurring after t in time, making the decoder autoregressive.

The individual components of the transformer are discussed in the following sections.

2.5.1 Input embedding

The input $\boldsymbol{X} \in \mathbb{R}^{T \times N}$, where the rows represent T points in time and the columns represent N time series, is embedded by applying a dense layer to produce an em-

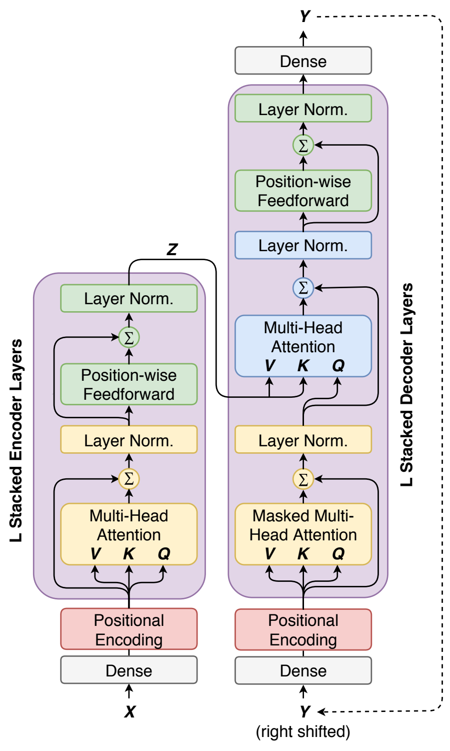


Figure 2.5: The Transformer architecture.

bedded $\boldsymbol{X'} \in \mathbb{R}^{T \times d}$, where d is the hidden dimension of the model and d is the same for both the encoder and the decoder. This is intended to allow the neural network to learn the relationships and dependencies between the different input time series. The embedded representation is given by Equation 2.15, with learned weights $\boldsymbol{W} \in \mathbb{R}^{N \times d}$ and a learned bias vector $\boldsymbol{b} \in \mathbb{R}^{d}$.

$$\text{dense}(\boldsymbol{X}) = \max(0, \boldsymbol{X}\boldsymbol{W} + \boldsymbol{b}) \tag{2.15}$$

2.5.2 Positional encoding

The model has no way of telling the position or order of each element in the input, so this information is injected in the positional encoding layer. This is done by using a learned lookup table to add the same value to the inputs at both test and train time depending on their position in time in the input. Specifically, the positional encoding layer adds a matrix lookup table of embeddings $\boldsymbol{E}$ to the input, where $\boldsymbol{E}$ is of the same dimension as the input.

2.5.3 Multi-head attention

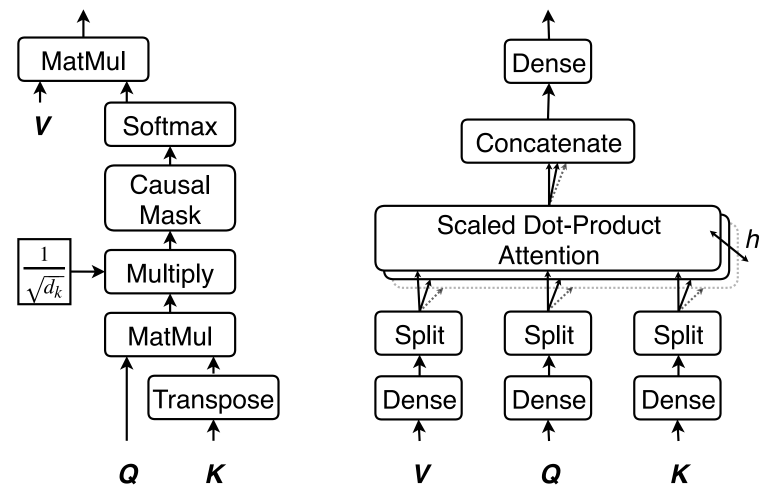
The primary innovation of the Transformer architecture is multi-head attention. Generic attention and dot-product attention will now be described as these are prerequisite to describing multi-head attention.

Given a single query vector and a set of key and value pairs (with each key and each value being a vector), an attention function matches the query to the keys to produce a weight for each key by applying an arbitrary fitness function. These key weights are then used to create an output vector comprised of the weighted sum of the values, where each value's weight is the weight assigned to its corresponding key.

Scaled dot-product attention, shown in Figure 2.6(a), is a specific implementation of an attention function. It uses the dot product of the query and each key to generate the weights, which are then passed through a softmax function such that the sum of all weights is equal to 1. In practice, the query row vectors are combined into a single matrix, $\boldsymbol{Q}$, allowing cheap matrix calculations to be used to evaluate the attention outputs in parallel. The keys and values are represented by row vectors in $\boldsymbol{K}$ and $\boldsymbol{V}$, respectively.

The dot product of the keys and queries is scaled (hence the name) by multiplying it by $1/\sqrt{d_k}$, with d_k being the key and query dimension, to prevent the dot product from becoming large when d_k is large as this may cause the softmax gradient to become very small and affect the gradient descent training.

The causal mask shown in Figure 2.6(a) is used exclusively in the decoder self-attention to prevent the attention function from matching any query to a key that occurs after itself in time. This is achieved by leaving the lower triangular portion of the matrix untouched and setting the other values to be large negative numbers, indicating a very poor match.



(a) Scaled dot product attention.

(b) Multi-head attention.

Figure 2.6: Attention mechanisms used in the transformer architecture.

Multi-head attention, shown in Figure 2.6(b) (b), applies a separate dense layer to each of the values, queries, and keys. The dense layer is applied per Equation 2.15 with learned weights $\boldsymbol{W} \in \mathbb{R}^{d \times d}$ and a learned bias vector $\boldsymbol{b} \in \mathbb{R}^{d}$. The outputs of the dense layers are then split along the last axis into h sets, or heads. As a result the key, query, and value dimension is reduced by a factor of h to $\frac{d}{h}$. Scaled dot-product attention is then run independently on each set. The results are concatenated and put through a final dense layer to produce the output of the attention function. The dense layer function on the output is defined by Equation 2.15 where $\boldsymbol{W} \in \mathbb{R}^{d \times d}$ is a learned weight matrix and $\boldsymbol{b} \in \mathbb{R}^{d}$ is a learned bias vector.

The dense layer combined with the split allows the multi-head attention to pick out information from different subspaces in the input and direct these to different attention heads. This is in contrast to a single head which must average all subspaces.

2.5.4 Feed-forward

The feed-forward layer is a two layer network with a rectified linear unit in the middle. Given an input $\boldsymbol{X} \in \mathbb{R}^{T \times d}$, the output $\boldsymbol{X'} \in \mathbb{R}^{T \times d}$ is populated by Equation 2.16 where $\boldsymbol{W}_1 \in \mathbb{R}^{d \times 4d}$, $\boldsymbol{b}_1 \in \mathbb{R}^{4d}$, $\boldsymbol{W}_2 \in \mathbb{R}^{4d \times d}$, and $\boldsymbol{b}_2 \in \mathbb{R}^{d}$ are learned weights and biases.

$$\text{feedforward}(\boldsymbol{X}) = \max(0, \boldsymbol{X}\boldsymbol{W}_1 + \boldsymbol{b}_1)\boldsymbol{W}_2 + \boldsymbol{b}_2 \tag{2.16}$$

2.5.5 Decoder dense output

The output of the decoder is passed through a dense layer to project the hidden dimension to the desired dimension of 1. The layer is implemented per Equation 2.15 where $\boldsymbol{W} \in \mathbb{R}^{d\times 1}$ is a learned weight matrix and $\boldsymbol{b} \in \mathbb{R}^1$ is a learned bias vector. By adjusting the dimension of $\boldsymbol{W}$ and $\boldsymbol{b}$ the network could be modified to perform multiple forecasts simultaneously.

2.5.6 Residuals & normalization

Residual connections [64] are applied around each sub-layer. That is, the output of each sub-layer is given by $\boldsymbol{X'} = \boldsymbol{X} + \text{subLayer}(\boldsymbol{X})$ where $\text{subLayer}(\boldsymbol{X})$ is the original output of the sub-layer. The outputs are then normalized by applying layer normalization [65].

2.5.7 Dropout and training

To help prevent overfitting to the training data, dropout [66] is applied during training at the output of both positional encoding layers, and immediately after the softmax operation in all multi-head attention layers.

When testing or being used for inference (that is, when not performing a training step), the decoder outputs are generated sequentially one at a time. After each output value is generated it is shifted right by one and populated in the decoder input and the model is executed again until all the outputs have been generated. Values that have not yet been generated are set to zero in the decoder input. These zero values do not affect the output of the decoder, as the decoder self-attention is masked so that it does not make use of them. When training, the decoder input is set to the known expected value and the model is executed — and the learnable parameters updated — a single time.

This process of supplying the known good values - or the previously generated values when in inference mode - to the decoder input is known as teacher forcing or professor forcing [67]. This is common practice in neural machine translation (NMT), but does not necessarily make sense in time series forecasting tasks. In NMT, words are usually represented as vectors with hundreds or thousands of elements. Adjacent words in the sentences are nearly always very different. However, in time series forecasting, the input vectors adjacent to each other are very similar. That is, the time series data generally has a high autocorrelation - for example the time series at 12pm and 1pm are likely to be very similar. This results in the model tending to simply reproduce the input at the output, as this results in a time series forecast that is perfect except shifted forward in time by one period.

This effect was overwhelming in the sequence to sequence models and caused the models to fail to converge, and so teacher forcing was never used. In the transformer, on the other hand, the models still converged. However, the performance of the models with and without teacher forcing is different.

2.5.8 Implementation

The transformer was implemented in tensorflow [59], building on an implementation by Park [68]. The encoder was provided with historical load and weather, and forecast weather. The projected decoder output was used as the load forecast.

Two models were considered – one with teacher forcing and one without. When teacher forcing was disabled, the decoder input matrix was simply set to zeros of the same dimension as when using teacher forcing.

The input matrix $\boldsymbol{X} \in \mathbb{R}^{T \times N}$, where T is the sequence length of the time series input, and N is the number of observations at each point in the time series, was provided directly to the model's encoder input, as described above.

The decoder output, $\boldsymbol{Y} \in \mathbb{R}^{U \times d}$, was projected to a dimension of 1 using equation 2.15, where $\boldsymbol{W} \in \mathbb{R}^{d \times 1}$ is a learned weight matrix and $\boldsymbol{b} \in \mathbb{R}^{1}$ is a learned bias vector, U is the length of the time series forecast, and d is the hidden dimension of the model.

2.6 Universal transformer

The universal transformer, shown in figure 2.7, is a variation on the transformer where the layers within the encoder and decoder (except for the timestep encoding) share parameters. This architecture was published in July 2018 [61] and there are only two published publications which reference it. The universal transformer outperforms the original transformer in the following tasks [61]:

- Neural machine translation – at which the universal transformer is the state of the art.
- Algorithmic tasks: copying the input to the output, copying the reversed input to the output, and adding values from the input.
- Natural language question-answering.
- Natural language modelling.

All layers within the encoder and decoder layers of the universal transformer are identical to those described in section 2.5. In the universal transformer, the encoder and decoder are stacked L times just as they are in the transformer. However in the universal transformer, all of the weights within the encoder layers are shared, and all the weights within the decoder layers are shared. This results in the depiction shown in figure 2.7, where the output of each encoder/decoder layer is fed back into itself L times.

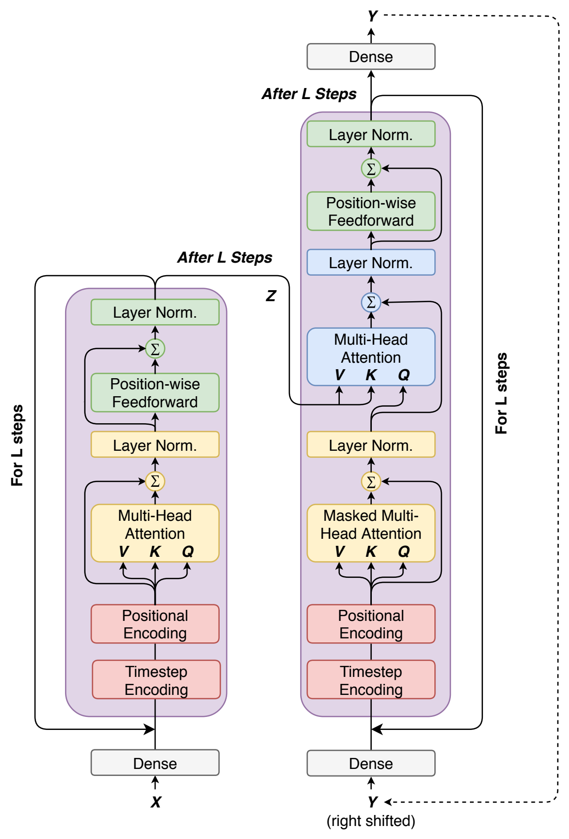


Figure 2.7: Universal transformer architecture.

The exception to this weight sharing is the temporal encoding layer. This layer is identical to the positional encoding layer described in section 2.5.2 and allows the model to keep track of which point in time (how many times it has recurred so far) it is currently at.

The original universal transformer publication specifies sinusoidal positional and timestep encoding [61]. The use of sinusoidal encoding versus fixed learned encoding (as used in this thesis) is discussed in the paper that originally presents the transformer [30], with the conclusion that it produces nearly identical results to sinusoidal encoding. The authors note in [30] that sinusoidal encoding is preferable in neural machine translation applications (where sequence lengths are variable and dictated by sentence lengths) because it allows the model to generalize to sequence lengths that it has not encountered during training. In the applications in this thesis, all sequence lengths are constant. For this reason, fixed learned encoding is used in the universal transformer in this thesis. One downside to using fixed learned encoding is that it introduces more parameters to the model, which may have negative effects on the performance and training of the model.

2.6.1 Implementation

The universal transformer was implemented in tensorflow [59], largely using the same codebase as the the transformer. Other details are identical to the transformer, except for teacher forcing which was always used.

2.7 Training and regularization of neural networks

Training and regularization is common to all described neural network models. The encoder inputs, decoder inputs, and expected outputs were summed with randomly distributed noise with a mean of 0 and standard deviation of 0.01 before being supplied to the model during training. This has been demonstrated to improve the generalization ability of a neural network [69] [70] and can be considered an extension of dropout [66].

All models were trained using the Adam optimizer [71] and a modified sum of errors squared loss function. Given a vector $\hat{\boldsymbol{y}}$ of predictions from the model and a vector $\boldsymbol{y}$ of expected predictions the loss function l is given by

$$l = \sum_{t=0}^{R} ((\boldsymbol{y}_t - \hat{\boldsymbol{y}}_t)^2 \times |\boldsymbol{y}_t|^c) \tag{2.17}$$

where c is a model hyperparameter. For $c > 0$ this function accentuates loss when the actual value is large — making the model more accurate at forecasting peaks.

2.8 Similar profile selection

Load profiles are influenced by exogenous data such as weather, day of the week, and holiday type [8]. A simple and intuitive method of load forecasting is to find periods in the past with similar exogenous data to the period being forecast and then use the load profiles from these past periods to form a forecast [12]. However, these similar period methods can be insufficient to capture complex patterns, especially over holiday periods which occur only once per year [35].

The forecasting system was provided with historical load and weather profiles from periods that had similar exogenous data to the period of the load profile being forecast. Similar periods were identified by first finding candidate similar periods an integer multiple of 1 year ±30 days away from the period being forecast. These candidates were then filtered down to periods with exactly matching hour and minute.

Then the weighted Euclidean distance between the period being forecast and each candidate similar period was calculated using the following features:

- maximum future (forecast) temperature,
- minimum future (forecast) temperature,
- maximum past load,
- current holiday type,
- current day of the week (when holiday always falls on same day of the week),
- current day of the month (when holiday always falls on same date), and
- current month of the year (when holiday always falls on same date).

The holiday type indicates the current holiday — Easter or Christmas for example — and is encoded as a time series of integers with a different integer for each holiday. When the holiday type always occurs on the same date each year then the month of the year and day of the month were used, whereas when the holiday type always occurs on the same day of the week each year then the day of the week was used. The candidate similar periods with the lowest Euclidean distance were selected as the final set of similar periods.

When training and testing the model the similar periods were selected from both the past and the future, as the train and test datasets were only five years each. It was assumed that, for testing, there were no changes in the patterns underlying the load profile over the duration of the testing set.

The following holidays were used:

- Australia Day;
- Regatta;

- Eight Hours Day;
- Easter;
- Anzac Day;
- Royal Hobart Show Day; and
- Christmas/New Year (Dec 21 through Jan 6).

The following heuristics were used to expand holidays when they occurred near weekends, to attempt to account for the tendency for people to take extra time off work around holidays.

- If a holiday occurs on a Monday, mark the previous two days as the same holiday.
- If a holiday occurs on a Tuesday, mark the previous three days as the same holiday.
- If a holiday occurs on a Friday, mark the following two days as the same holiday.
- If a holiday occurs on a Thursday, mark the following three days as the same holiday.

Chapter 3

Results and discussion

This chapter presents case studies of Bruny Island and ISO New England using the proposed forecasting models.

3.1 Bruny Island

Bruny Island, shown in Figure 1.1, is located approximately two kilometres off the coast of south-east Tasmania with a permanent resident population of approximately 800 people. The island is a popular holiday destination, with Easter periods typically experiencing an influx of up to 500 cars in a single day. The island is supplied by two feeders, depicted in Figure 3.1, with one feeder supplying a small portion of the island to the North and the other supplying the main portion of the island to the South. This case study deals only with the feeder supplying the main portion of the island to the South.

During holiday period morning and afternoon peaks the submarine feeder reaches its capacity and a diesel generator located on the island is used to reduce the feeder load. The substantial increase in load over the Easter holiday period for multiple years can be seen in Figure 3.2.

To avoid the use of the generator, the CONSORT project installed a set of residential batteries on the island for the purposes of peak-shifting. These batteries are coordinated by the network aware coordination algorithm (NAC). In order to peak-shift while making efficient use of the batteries, the NAC requires an accurate forecast of load with a 24-hour horizon and 30-minute resolution.

The proposed forecasting models were evaluated on historical data, with the details of the implementations and the results presented in the following sections. A transformer-based model was implemented for online forecasting as part of the NAC, with details given in section 3.1.7.

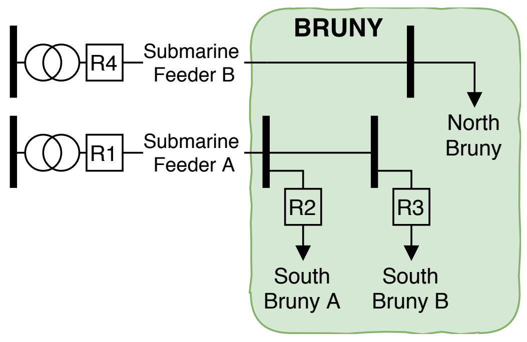


Figure 3.1: Single line schematic of the distribution network on Bruny Island.

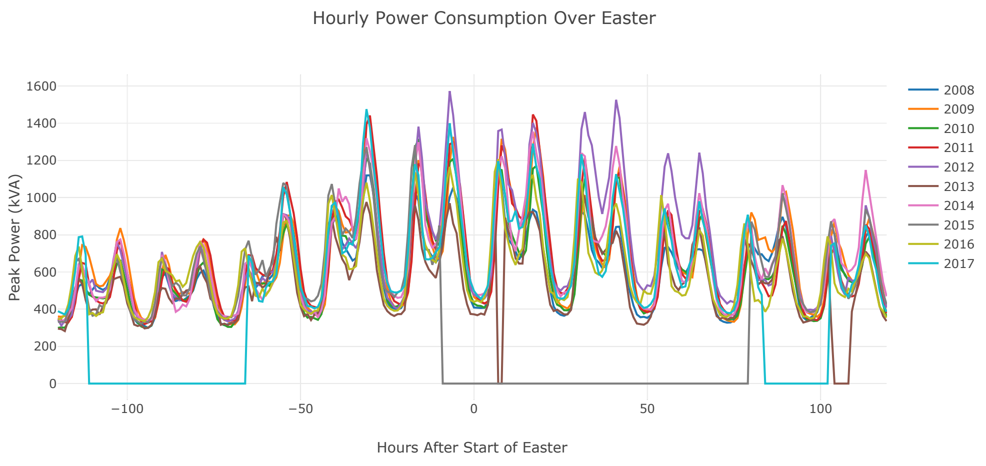


Figure 3.2: Easter load on Bruny Island 2008 through 2017. Unusable missing/bad data can also be seen in this graph.

3.1.1 Data analysis

In section 1.2.3 the general properties of load profiles and how they are influenced by exogenous factors was discussed. Now, these general properties will be briefly investigated for the Bruny Island feeder.

Available data

The following data is available:

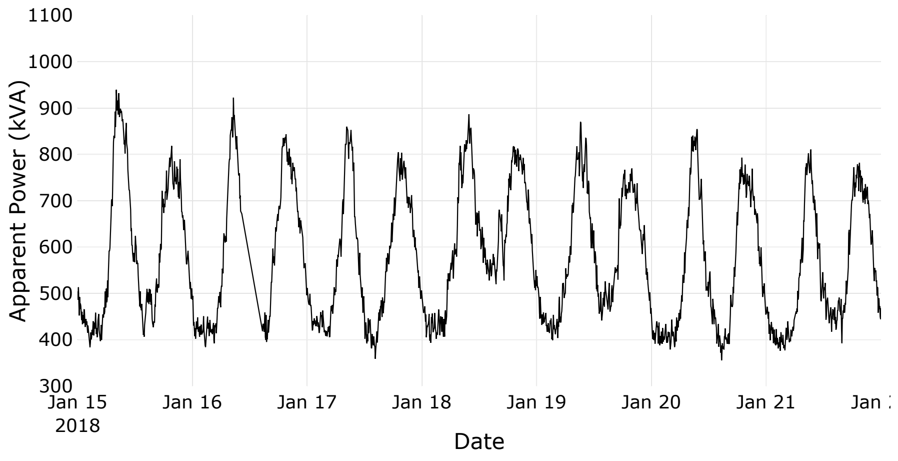
- Apparent power measured at recloser R1 (figure 3.1) between January 1, 2007 and June 25, 2018
- Ambient Temperature, solar irradiance, humidity, and wind speed at Lenah Valley between October 1, 2009 and June 25, 2018
- Vehicle movement data in 10 minute resolution between June 4, 2015 and March 24, 2018

Vehicle movement data is provided as a set of observations every 10 minutes recording the number of vehicles arriving on the island, and the number of vehicles departing the island. By integrating this, a relative number of vehicles on the island can be established. There is a significant amount of bad or missing data throughout these datasets which has been handled by limiting the use of data where there are too many missing or bad values.

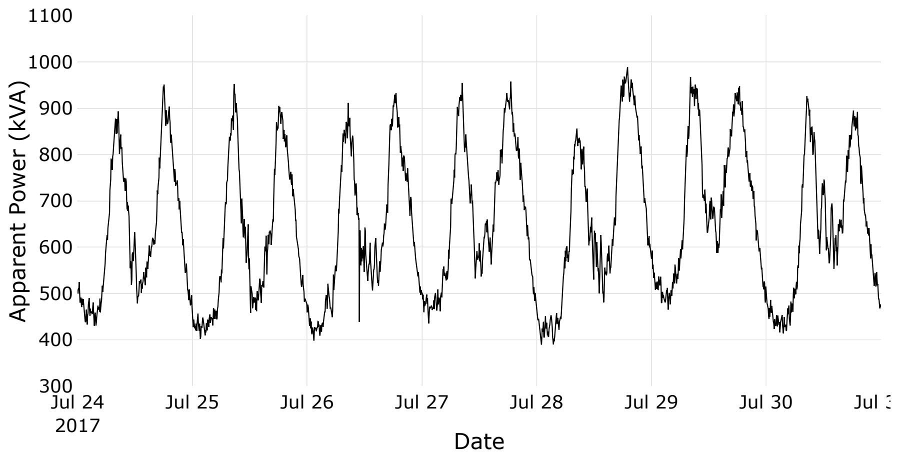
Analysis

Figure 3.3 shows apparent power draw on Bruny Island over a winter week, a summer week, and over an entire year. These two weeks were selected to avoid special days such as holidays and are representative of typical weeks. Several differences can be seen between the summer and winter weeks. In Summer, the midday load is about the same as the overnight load, whereas in Winter it is quite different. Summer afternoon peaks are smaller than the morning peaks, whereas in winter they are similar. At a high level, it is clear that the load is generally larger in winter – likely as a result of increased residential heating. An intuitive reaction to this might be to use different forecasting models for summer and winter - but of course then a delineation must be decided on for when to switch between the two models. As one of the aims of the forecasting system is to be practical to implement, it is desirable to have a single system that is be able to forecast both summer and winter. By supplying temperature as an exogenous input this should be possible.

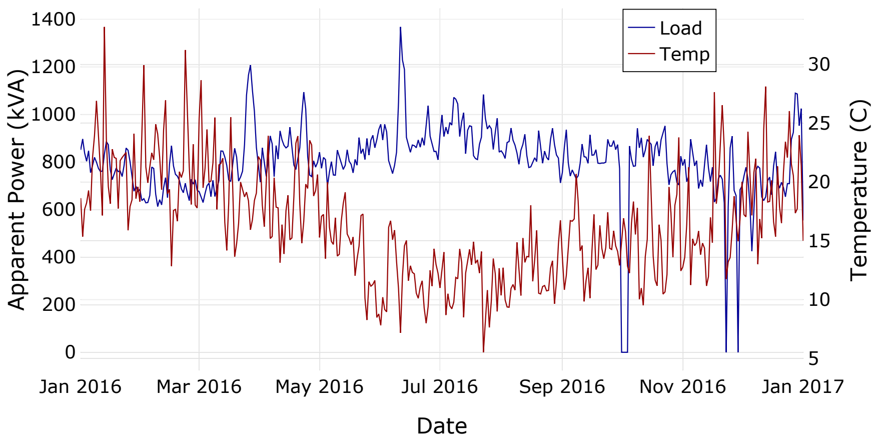
It was mentioned in section 1.2.3 that weekends and weekdays tend to have different load profiles, but this is not immediately obvious from figure 3.3. Figure 3.4(a) shows this difference between days of the week. What is immediately obviously is that the differences between days of the week are not restricted to 24 hour bounds



(a) Load over a single summer week in 2018.



(b) Load over a single winter week in 2017.



(c) Peak daily load and peak daily temperature over all of 2017.

Figure 3.3: Bruny Island load profiles.

– the different load profile shapes gently merge into each other over the course of an afternoon or morning. It can be seen that the Friday profile morphs into the Saturday profile during the afternoon, perhaps as people arrive on the island for the weekend, and the Sunday profile morphs into a weekday profile over the afternoon, perhaps as people depart the island.

Figures 3.4(b) and 3.4(c) further support that the changes in load profiles are a result of people arriving on or leaving the island. It can be seen that many vehicles tend to arrive on the island on Friday afternoon, leading to the change in load profile between Friday and Saturday. Likewise, many vehicles leave the island on Sunday afternoon, shifting the load profile from Sunday to a weekday.

This brief investigation serves to highlight some of the challenges that the load forecasting system must handle.

3.1.2 Forecasting tasks

The forecasters were evaluated on several tasks, shown in table 3.2, and discussed in section 3.1.4. The hourly task has the forecaster predict the future 24 hours of load in one-hour resolution given the previous 24 hours of load and the future 24 hours of date/time, holiday, and temperature data. The half-hourly task is the same as hourly, except using half-hour resolution data. The hourly long task has the forecaster predict the future 24 hours of load in one-hour resolution given the previous 168 hours (one week) of load and the future 24 hours of date/time, holiday, and temperature data. The hourly (c=3) task is the same as the hourly task but with the loss function modified, as described in section 2.7, with $c = 3$.

The tasks listed as being "with similar profiles" include five similar profiles ($k = 5$, as described in the following section Data and model configuration). The tasks that do not list similar profiles have no similar profiles, $k = 0$.

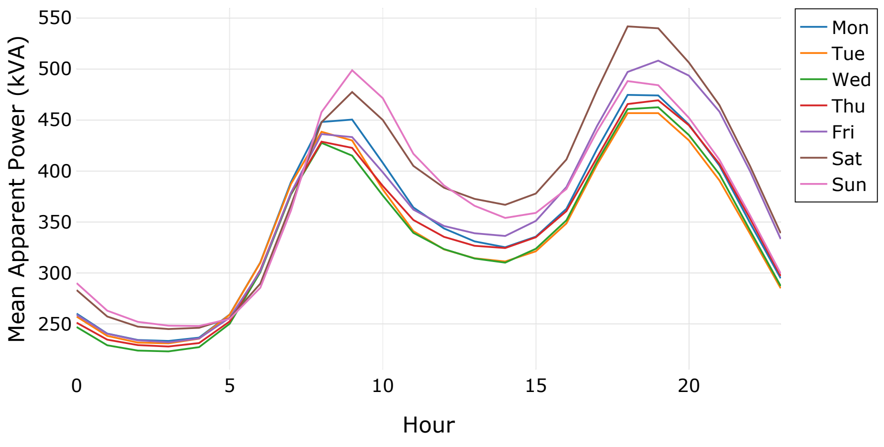
3.1.3 Data and model configuration

The available data was split into a training set (October 1 2009 through February 15 2014) and a testing set (February 16 2014 through June 25 2018). For the hourly task, the network was supplied with data in hourly resolution from the previous and future 24 hours:

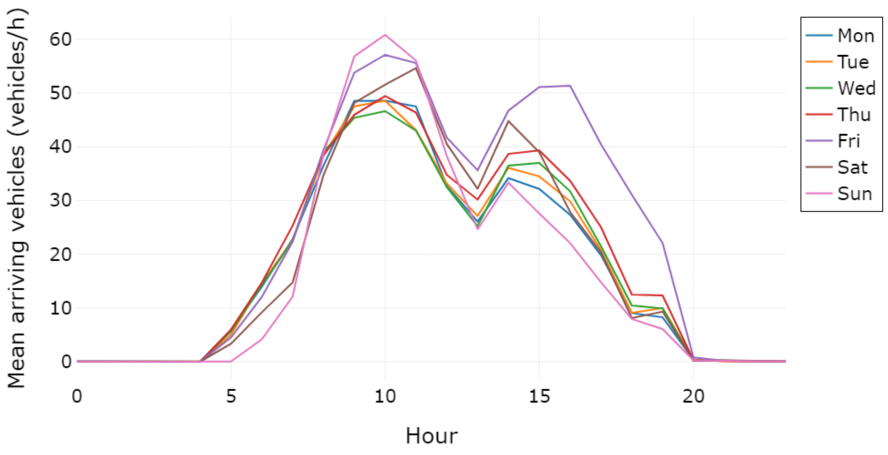
- The previous 24 hours of load, $\boldsymbol{l} = [l_1\, l_2 \ldots l_{24}]^\top$, with l_{24} being the most recent load.

- Temperature for the future 24 hours, $\boldsymbol{t} = [t_1 t_2 \ldots t_{24}]^\top$, with t_1 being the soonest point in the future.

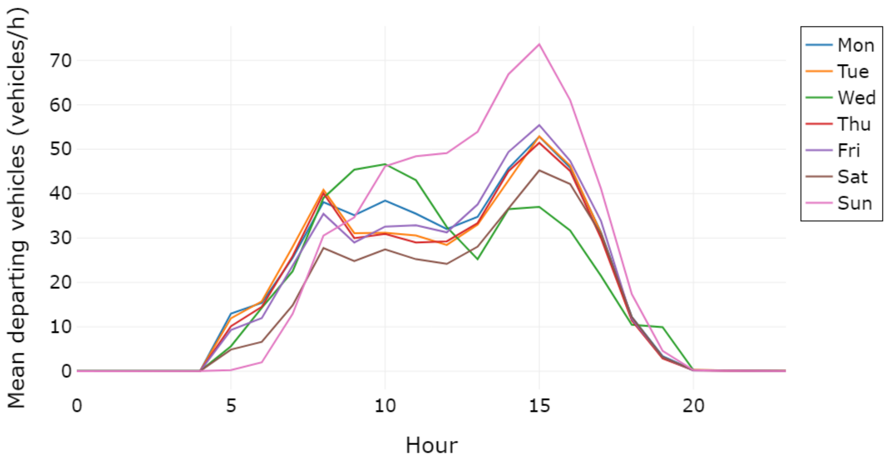
- Day of the week as an integer from 0 to 6 (local time) over the future 24 hours, $\boldsymbol{d} = [d_1\, d_2 \ldots d_{24}]^\top$, with d_1 being the soonest point in the future.



(a) Average load profiles for each day of the week.



(b) Average number of cars arriving on the island for each day of the week.



(c) Average number of cars leaving the island for each day of the week.

Figure 3.4: Bruny Island average load profiles.

- Minutes since midnight (local time) for the future 24 hours, $\boldsymbol{m} = [m_1 m_2 ... m_{24}]^\top$, with m_1 being the soonest point in the future.

- Boolean 1 or 0 indicating whether it is a holiday at each hour over the next 24 hours, $\boldsymbol{h} = [h_1\ h_2\ ...\ h_{24}]^\top$, with h_1 being the soonest point in the future.

- Holiday type as an integer from 0 to n at each hour over the future 24 hours, where there are n different holidays in the year, $\boldsymbol{g} = [g_1\ g_2\ ...\ g_{24}]^\top$, with g_1 being the soonest point in the future.

- Month of the year from 0 to 11 (local time) over the future 24 hours, $\boldsymbol{n} = [n_1\ n_2\ ...\ n_{24}]^\top$, with n_1 being the soonest point in the future.

Additionally, similar profile data was constructed for k similar days, as described in section 2.8. The weights used are given in table 3.1. The extremely large weights are used to ensure that similar profiles are always selected from the same month and day of month/day of week unless there are none available. The similar day data is aligned with the same points in time as the $\boldsymbol{t}$ vector. The order in which the similar days were added to the $\boldsymbol{X}$ input matrix was randomized during both training and testing/inference. The similar profile data is:

- 24 hours of similar profile load $\boldsymbol{l}_{sk} = [l_{sk1}\ l_{sk2}\ ...\ l_{sk24}]^\top$.

- 24 hours of similar profile temperature (corresponding to the same points in time as $\boldsymbol{l}_{sk}$), $\boldsymbol{t}_{sk} = [t_{sk1}\ t_{sk2}\ ...\ t_{sk24}]^\top$.

This data was concatenated to form the input matrix given in equation 3.1.

$$\boldsymbol{X} = [\boldsymbol{l}\ \boldsymbol{t}\ \boldsymbol{d}\ \boldsymbol{m}\ \boldsymbol{h}\ \boldsymbol{g}\ \boldsymbol{n}\ \boldsymbol{l}_{s1}\ \boldsymbol{t}_{s1}\ \dots\ \boldsymbol{l}_{sk}\ \boldsymbol{t}_{sk}] \in \mathbb{R}^{24\times(7+2k)} \tag{3.1}$$

The half-hourly configuration is the same as the hourly configuration, except all vectors contain 48 elements (representing 24 hours of data) and so $\boldsymbol{X} \in \mathbb{R}^{48\times(7+2k)}$.

The hourly long configuration modifies only the $\boldsymbol{l}$ vector, such that $\boldsymbol{l} = [l_1 l_2 ... l_{168}]^\top$ representing one week of hour-resolution data, with l_{168} being the most recent observation. The other vectors were zero padded to match this length. With $\boldsymbol{z} \in \mathbb{R}^{144}$ being a zero vector, and $\hat{\boldsymbol{v}} = [\boldsymbol{z}^\top \boldsymbol{v}^\top]^\top$ for any vector $\boldsymbol{v}$, the input $\boldsymbol{X}$ is given by equation 3.2.

$$\boldsymbol{X} = [\boldsymbol{l}\ \hat{\boldsymbol{t}}\ \hat{\boldsymbol{d}}\ \hat{\boldsymbol{m}}\ \hat{\boldsymbol{h}}\ \hat{\boldsymbol{g}}\ \hat{\boldsymbol{n}}\ \hat{\boldsymbol{l}_{s1}}\ \hat{\boldsymbol{t}_{s1}}\ \dots\ \hat{\boldsymbol{l}_{sk}}\ \hat{\boldsymbol{t}_{sk}}] \in \mathbb{R}^{168\times(7+2k)} \tag{3.2}$$

The forecasting models were configured with the parameters given in table 3.1 and trained for 10^5 iterations. All the models presented in section 2 are evaluated. The "Transformer" model is the transformer with teacher forcing enabled, and the "Transformer no TF" model is the transformer with teacher forcing disabled.

Table 3.1: Offline Evaluation model parameters.

Model/Method	**Parameter**	**Description**	**Value**
SARIMAX	p	AR model order	2
	d	Number of differences	1
	q	MA model order	1
	P	Seasonal AR model order	1
	D	Number of seasonal differences	1
	Q	Seasonal MA model order	1
	s	Period of seasonality	24
Sequence to Sequence	L	Number of encoder and decoder layers	2
	d	Hidden dimension	16
	D	Dropout fraction	0
	c	Loss function modifier	0
	l	Learning rate	0.001
	-	Training batch size	16
Transformer	L	Number of encoder and decoder layers	2
	d	Hidden dimension	16
	h	Number of attention heads	2
	D	Dropout fraction	0.2
	c	Loss function modifier	0
	l	Learning rate	0.001
	-	Training batch size	16
Universal Transformer	L	Number of encoder and decoder layers	2
	d	Hidden dimension	16
	h	Number of attention heads	2
	D	Dropout fraction	0.2
	c	Loss function modifier	0
	l	Learning rate	0.001
	-	Training batch size	16
Similar Profile Selection	-	Maximum future temperature weight	10
	-	Minimum future temperature weight	20
	-	Maximum past load weight	30
	-	Holiday type weight	1e9
	-	Day of week weight	1e6
	-	Day of month weight	1e6
	-	Month of year weight	1e6

3.1.4 Offline evaluation results and discussion

The forecaster was trained on the training dataset (October 1 2009 through February 15 2014) and tested on the testing dataset (February 16 2014 through June 25 2018). The train and test datasets were then switched in order to cross-validate the results. All presented results are the average of the cross validated results. Results are shown in table 3.2.

Both the mean absolute percentage error (MAPE) and mean absolute error (MAE) are shown. These metrics are calculated for all load, for load only over 1MVA, and for load that is at a first large peak. A first large peak is a morning or afternoon peak that is over 1MVA and is greater than 36 hours after any previous peak that was over 1MVA. The first large peak metric is used because it is generally considered difficult to forecast. These two metrics are intended to assess the performance of the system on anomalous holiday periods.

The SARIMAX model proved to produce results that are relatively poor compared to the other models, and so was not considered beyond the hourly without similar days task. The universal transformer is consistently a poor performer, never producing the best results for any task or metric. The transformer model is generally outperformed by, or at least very similar to, the transformer without teacher forcing model. Excluding these models, it is clear that there is no model which is overall superior between the S2S and transformer without teacher forcing models.

The S2S model appears to excel at the hourly task, achieving the best MAPE scores above 1 MVA and at first large peak, while also being close to the transformer without teacher forcing on the overall MAPE score. The S2S model generally produces good results across the board.

The transformer without teacher forcing is generally good at minimizing the overall MAPE score. It also performs exceedingly well in the hourly long task, achieving the lowest overall MAPE while the other two metrics are close to the hourly with similar and $c = 3$ task. This model is also arguably the best at the half-hourly tasks, achieving the best results on the majority of metrics across both tasks. The dominance of the transformer-based models on the half-hourly and hourly long tasks is likely a result of the transformer's ability to handle long sequence lengths, as discussed in section 2.5.

The hourly with $c = 3$ tasks show no consensus on which model is generally superior. Interestingly, the hourly long task achieves similar results to the hourly with $c = 3$ with similar profiles task. It may be the case that increasing the c hyper-parameter simply adds bias to the model – further analysis should be undertaken to assess this.

In some cases the MAPE decreases after introducing similar profiles. It could be that in these cases the similar profile data is providing poor quality data as input, or perhaps the model is not able to effectively handle the high dimensionality of the input. It is difficult to say for sure why this happens.

The MAPE versus horizon plot is shown in figure 3.5 for the hourly task. Only the sequence to sequence (S2S) and transformer without teacher forcing models are shown. The cross-validation results, denoted with “CV” in the legend, are the results when the model is trained on data from 2014-2018 and evaluated on data from 2009-2014. The cross-validated results agree within 0.8%. The corresponding distribution of forecast error (though not for the cross validated results) can be seen in figure 3.6. The two distributions show an encouraging resemblance to a normal distribution.

This case study suggests that the sequence to sequence model offers good overall performance, while the transformer without teacher forcing is capable of outperforming the sequence to sequence model, but at the risk of more variability across the anomalous holiday metrics. The transformer with teacher forcing and universal transformer models do not achieve competitive results, although notably the universal transformer has not been evaluated without teacher forcing which may improve performance as it does for the transformer model. The similar profile method generally improves the performance of the forecasts. Increasing the c parameter causes the model to increase accuracy when forecasting peak values at the expense of overall accuracy, as intended.

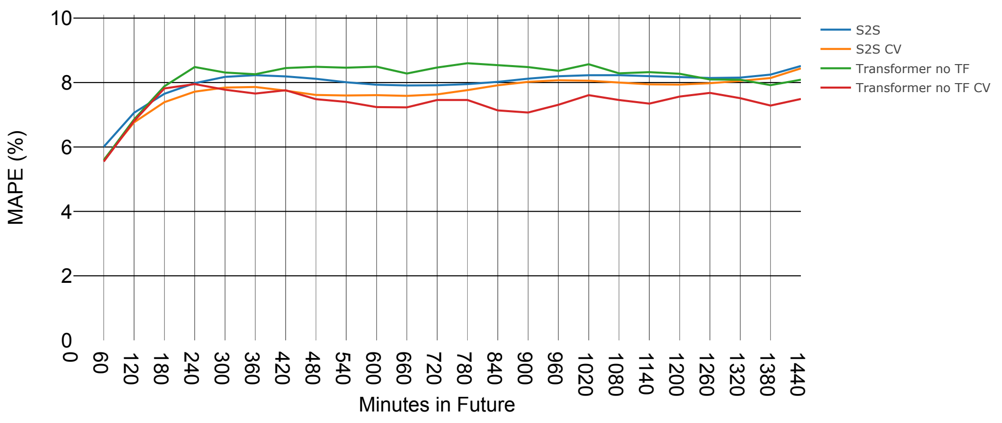


Figure 3.5: Mean absolute percentage error of each point in the forecasts for Bruny Island, task “Hourly”.

.

3.1.5 Training data requirements

So far the models have been trained with 4.5 years of data. Table 2.7 shows how the MAPE changes on the hourly task when the amount of training data is decreased. The training data was always the data closest to the train set. These results indicate that either there are diminishing improvements to MAPE as the amount of training

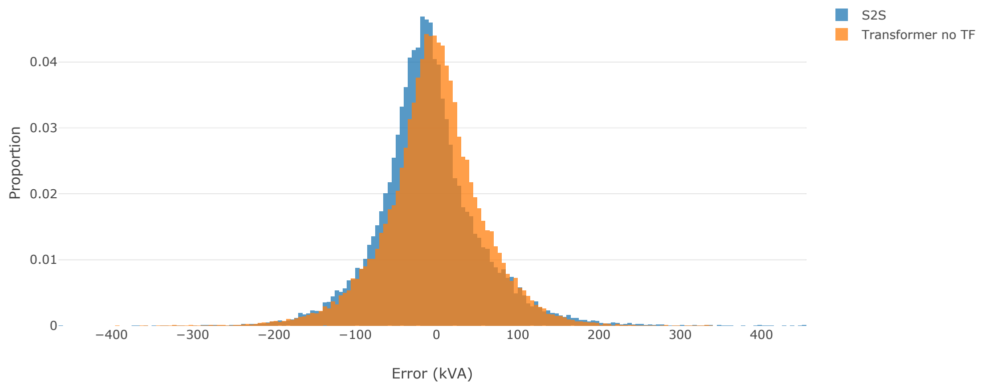


Figure 3.6: Distribution of forecast error corresponding to Figure 3.5.

Table 3.2: Offline evaluation results.

Task	**Model**	**MAPE (%)**	**MAE (kVA)**	**MAPE over 1 MVA (%)**	**MAE over 1 MVA (kVA)**	**MAPE at first large peak (%)**	**MAE at first large peak (kVA)**
Hourly	SARIMAX	14.34	82.1	12.31	137.7	18.86	200.9
	S2S	7.85	42.8	**9.87**	**110.6**	**11.42**	**112.6**
	Transformer	7.82	43.1	10.33	115.1	13.00	140.1
	Transformer no TF	**7.77**	**42.6**	11.45	128.9	13.24	142.0
	U. Transformer	8.20	46.0	11.72	131.6	14.52	155.2
Hourly with similar profiles	S2S	7.68	41.6	**9.77**	**109.6**	**11.05**	**118.7**
	Transformer	7.40	40.7	10.25	115.2	12.45	133.6
	Transformer no TF	**7.24**	**39.6**	10.32	115.8	12.47	133.7
	U. Transformer	8.15	44.8	11.10	123.5	12.67	135.5
Half-hourly	S2S	**7.80**	**42.7**	9.88	111.3	12.30	131.5
	Transformer	9.14	50.9	13.03	148.0	15.80	170.0
	Transformer no TF	8.24	44.0	**9.03**	**101.5**	**10.22**	**110.4**
	U. Transformer	8.60	48.0	13.60	154.6	14.33	154.5
Half-hourly with similar profiles	S2S	8.44	45.0	10.23	116.2	11.10	119.1
	Transformer	7.82	42.5	10.78	121.6	**10.98**	**118.5**
	Transformer no TF	**7.41**	**41.0**	**10.16**	**114.1**	12.54	134.8
	U. Transformer	9.05	48.1	10.22	114.6	11.90	136.0
Hourly long with similar profiles	S2S	7.49	40.9	**9.43**	**106.2**	10.10	109.5
	Transformer	6.97	38.0	10.82	122.5	11.30	121.8
	Transformer no TF	**6.68**	**36.8**	9.85	111.8	**9.54**	**103.1**
	U. Transformer	6.74	38.5	10.35	116.1	12.82	138.2
Hourly; $c = 3$	S2S	8.96	46.8	**8.61**	**95.9**	**9.61**	**103.0**
	Transformer	**8.14**	**44.5**	10.46	117.5	12.25	130.9
	Transformer no TF	8.65	45.4	9.36	105.0	10.76	115.1
	U. Transformer	9.29	48.5	11.05	124.5	12.97	138.9
Hourly with similar profiles; $c = 3$	S2S	**8.32**	**43.8**	9.20	103.1	10.68	114.8
	Transformer	8.58	45.5	**8.90**	**99.5**	**9.29**	**99.9**
	Transformer no TF	8.86	45.4	8.99	101.1	10.21	109.7
	U. Transformer	9.00	48.3	10.07	112.5	9.34	100.5

data increases, or the data from further in the past does not help with forecasting data in the future.

Table 3.3: Training data requirements.

Years of Training Data	Model	MAPE (%)
4.5	S2S	7.85
	Transformer no TF	7.77
3.5	S2S	7.95
	Transformer no TF	7.66
2.5	S2S	7.96
	Transformer no TF	7.96
1.5	S2S	8.31
	Transformer no TF	8.25

3.1.6 Autoregression requirements

All models presented have been supplied with at least the past 24 hour of load. If this data is not supplied, the models are still able to forecast load. Evaluated on the hourly task with the $\boldsymbol{l}$ past load vector removed, the sequence to sequence model achives a MAPE of 10.68% and the transformer without teacher forcing model achieves a MAPE of 10.53%. With a careful training regime it may be possible to train these models to robustly handle missing input data.

3.1.7 Online CONSORT implementation

In July 2018 the load forecasting system was implemented as part of the CONSORT project and was used to assist with dispatch of residential batteries. The model implemented was a transformer (with teacher forcing enabled), and is somewhat different to the models discussed in the offline evaluation. Although the implemented model achieved sufficient accuracy while in use, it has since been determined to be a sub-optimal configuration. As the data and model configuration is somewhat different than described for the offline evaluation in section 3.1.3, the data and model configuration section is largely repeated here.

Data and Model Configuration

The following data was available from 2009-2018:

- Apparent power at reclosers R1 through R4 (Figure 3.1).
- Temperature at Lenah Valley, Tasmania (50km from Bruny Island).

- Apparent power consumption at St Helens, Tasmania.

This data was averaged to 30 minute resolution and split into a training set containing data from October 2009 through September 2014, and a testing set containing data from October 2014 through April 2018.

The network was supplied with data from the previous and future 24 hours, for a total input sequence length of 96 (representing 48 hours at 30 minute resolution). The output sequence length was 48 (24 hours). The data was configured differently than described in section 3.1.3. The $\boldsymbol{X}$ input matrix was constructed with all vectors containing 96 elements, representing the previous 24 and the future 24 hours. The past load vector $\boldsymbol{l} = [l_1\, l_2\, ...\, l_{48}\, 0\, ...\, 0]^\top \in \mathbb{R}^{96}$ was populated with the past 48 load observations (representing 24 hours) and zero padded to match the shape of the rest of the data.

The following time series were supplied to the model input:

- Apparent power from recloser R1 (Figure 3.1), with future values set to zero.
- Temperature.
- Day of the week as an integer from 0 to 6 (local time).
- Minutes since midnight (local time).
- Boolean 1 or 0 indicating whether it is a holiday.
- Holiday type.

When used for inference, temperature forecasts were obtained from the Bureau of Meteorology. Additionally, five similar periods were identified using data from R1 by the process described in section 2.8. The data over the similar periods for each of the following time series was provided as input:

- Reclosers R1, R2, R3, and R4 (Figure 3.1) (as separate time series).
- St Helens recloser.
- Lenah Valley temperature.

In total, 36 time series were provided as input to the model. St Helens was included because it was observed to display similar patterns to Bruny Island around holiday periods.

The forecasting system was configured with the parameters in table 3.4, with the upper section giving transformer model parameters and the lower section giving weights used for similar period selection. The model was trained for 10^5 iterations.

Table 3.4: Case study model parameters.

Parameter	Description	Value
L	Number of encoder and decoder layers	4
d	Hidden dimension	32
h	Number of attention heads	4
D	Dropout fraction	0.2
c	Loss function modifier	3
-	Training batch size	16
-	Maximum future temperature weight	10
-	Minimum future temperature weight	20
-	Maximum past load weight	30
-	Holiday type weight	1e9
-	Day of week weight	1e6
-	Day of month weight	1e6
-	Month of year weight	1e6

Online evaluation results

When implemented live on the Bruny Island distribution network during the July 2018 school holiday period, as part of the CONSORT project, the forecaster was observed to reliably forecast large demand peaks. This enabled the fleet of distributed batteries to be used effectively in providing network support via net demand peak reduction. An accurate forecast, issued early enough in advance of the occurence of the demand peak, was observed to give the batteries adequate time to store energy in the lead up to, and discharge during the demand peak period. In at least one instance over the test period this was sufficient to avoid the island's diesel generator from being used at all, when it otherwise almost certainly would have been required. Data collected during this peak demand period can be seen in Figure 3.7. The upper section shows 24 hours of historical load in black, plus the most recent 24-hour horizon forecast in dashed black (recalculated every five minutes) and all old forecasts in grey. The lower section shows the battery charge rate, where a negative value of battery charge rate indicates the batteries are supporting the grid.

Typically the generator is switched on when load exceeds 1050 kVA. During the first peak the graph shows the batteries supplying between 50 and 100 kW to the island. Without this support from the batteries, the generator would have likely been required to operate.

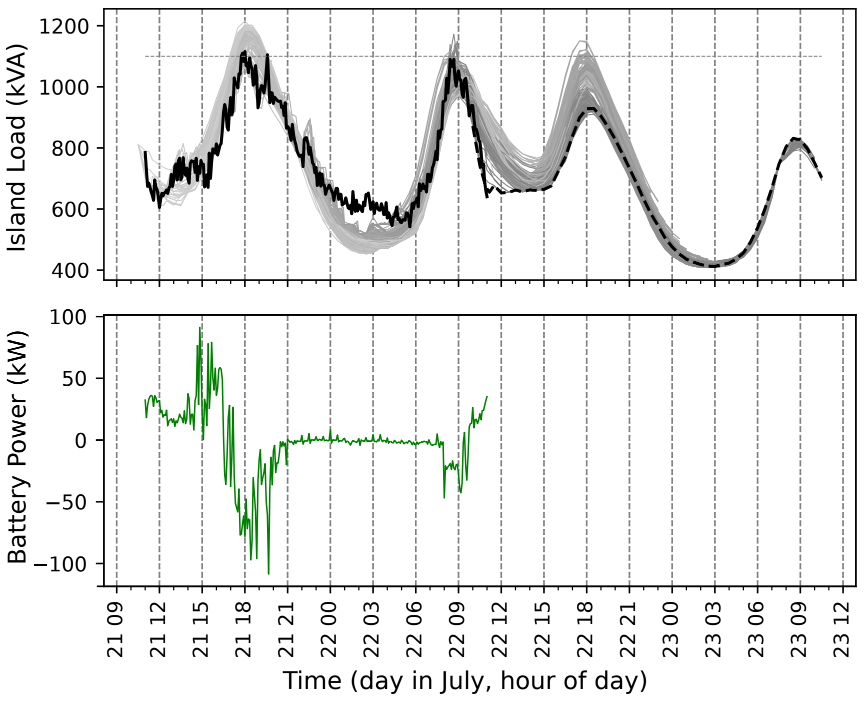


Figure 3.7: Results from the forecasting system's implementation in the CONSORT project.

3.2 ISO New England

ISO New England is a regional transmission organization operating in the New England area of the United States. Data from ISO New England has been used by several papers for evaluating the performance of their load forecasting systems [23][35]. The aim of this section is to compare the proposed forecasting models to the models and methods used by Ceperic et al. [23], whose method is discussed in section 1.2.6, and Chen et al. [35], whose method is discussed in section 1.2.7 and to validate that the models used for Bruny Island can also be applied to other feeders with no modification. A week of load from the ISO New England dataset is shown in figure 3.8.

Ceperic et al. [23] and Chen et al. [35] trained their model on data from March 2003 until January 2006 and tested on January 1 2006 to December 31 2006. Publicly available Historical electrical load data was obtained from the ISO New England website [72]. Publicly available historical temperature was obtained from the National Centers for Environment Information [73] at Concord, New Hampshire, United States (latitude 43.204900 degrees, longitude -71.502740 degrees). This is different temperature data to that used by Ceperic et al. [23] as that data could not be accessed.

The following holidays were used, and the same heuristics described in section

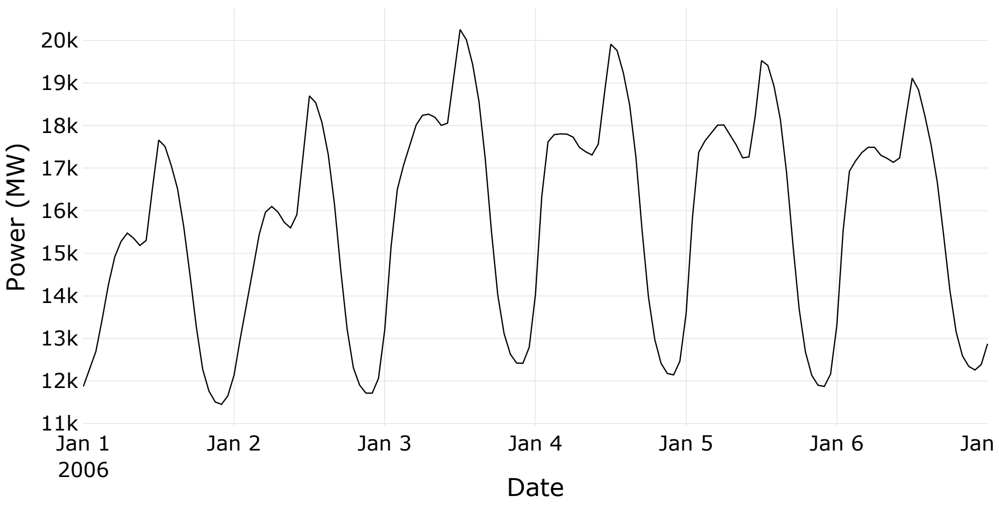


Figure 3.8: A week of load for ISO New England.

2.8 were applied.

- Birthday of Martin Luther King, Jr.;
- Washington's Birthday;
- Memorial Day;
- Independence Day;
- Labor Day;
- Colombus Day;
- Veteran's Day;
- Thanksgiving;
- Easter; and
- Christmas/New Year (Dec 21 through Jan 6).

The forecasting was performed identically to the "hourly long" task described in section 3.1.2, except with no similar days ($k = 0$). The input matrix $\boldsymbol{X}$ was constructed by equation 3.2. Ceperic et al. [23] and Chen et al. [35] turned heuristics about heating and cooling load based on temperature into features for the neural network. This is generally considered good practice [74]. However, as one of the primary aims of this thesis is to develop a practical load forecasting system that does not require significant modifications when forecasting for different feeders, this data was not provided as an input.

The model configurations are the same as that described in table 3.1, and the models were trained for 10^5 iterations. The results are shown in table 3.5. The results show a MAPE worse than other papers, but considering the differences in inputs (heuristic weather inputs), that [35] only performs forecasts once per day, and that the models here are identical to the models used for Bruny Island, these results are encouraging. The MAPE versus forecast horizon and distribution in forecast error are shown in figures 3.9 and 3.10.

These results demonstrate that the same models that forecast load at a low level in the distribution network can also be applied to forecasting at a high level in the transmission network with absolutely no modification.

Table 3.5: ISO New England results.

Model	**MAPE (%)**
Transformer no TF	2.14
S2S	2.15
SSA-SVR [23]	1.31
SIWNN [35]	1.71
ANN [35]	2.03
Transformer	3.09
U. Transformer	3.65

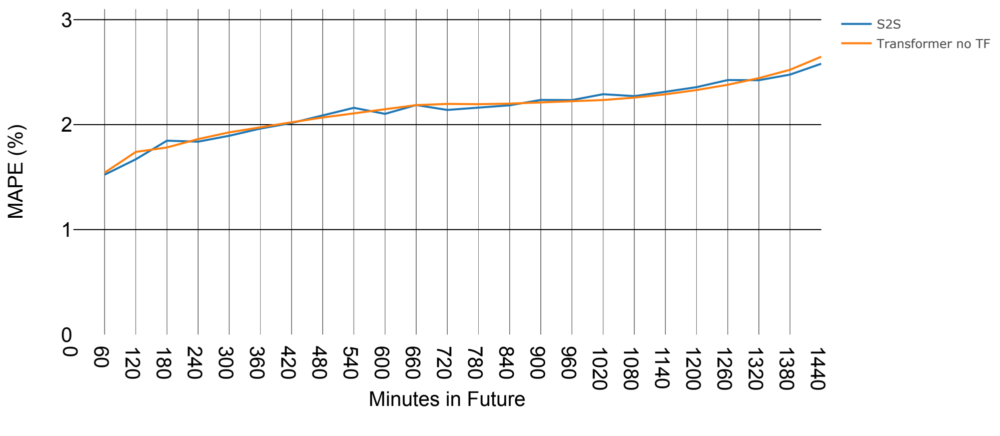


Figure 3.9: Mean absolute percentage error of each point in the forecasts for ISO New England, task “Hourly long”.
.

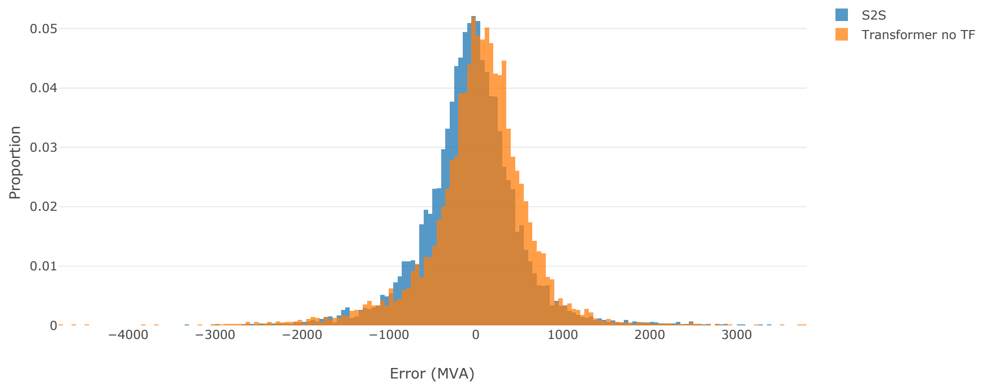


Figure 3.10: Distribution of forecast error corresponding to Figure 3.9.

3.3 Implementation requirements

The training time of the models without similar days is presented in table 3.6. The models were restricted to two cores/four threads on an AMD Ryzen 1700 at 3.5 GHz. The performance is generally not improved by using a GPU or more CPU cores because of the small batch size, except for the transformer on the hourly long task where the speed can be doubled with a GPU or more cores. It is likely that by tuning the batch size and learning rate it could be more feasible to train on a GPU in a far smaller number of iterations, reducing the overall training time. Each model requires less than 2 GB of memory.

Table 3.6: Training and evaluation times.

Task	Model	Training time	Evaluation time (ms)
Hourly	S2S	28m	17
	Transformer no TF	52m	31
Half-hourly	S2S	43m	26
	Transformer no TF	1h 32m	55
Hourly long	S2S	52m	31
	Transformer no TF	3h 5m	110

Chapter 4

Conclusion

Four neural-network based models were evaluated for short-term load forecasting: sequence to sequence recurrent neural network, transformer with and without teacher forcing, and universal transformer. The models were applied to the Bruny Island submarine feeder and to the standard ISO New England dataset. Metrics assessing the MAPE above 1 MVA and MAPE at first large peak were used to assess the performance of the forecasters on anomalous holiday periods. The transformer with teacher forcing and universal transformer models did not produce results competitive with the sequence to sequence and transformer without teacher forcing models.

The sequence to sequence model was found to achieve overall good results on the Bruny Island feeder when evaluated on overall MAPE and MAPE above 1 MVA. The transformer without teacher forcing model was found to achieve the best results on overall MAPE on the Bruny Island feeder, but was somewhat more volatile on the metrics above 1 MVA when compared to the sequence to sequence model. The transformer without teacher forcing model also appeared to handle forecasting tasks involving long sequence lengths better than the sequence to sequence model. Overall, though, which model is superior on this feeder is not clear and will depend on whether the forecasting is hourly or half-hourly, and whether accuracy overall or accuracy over holiday periods is most important.

A transformer-based model was implemented as part of the Bruny Island CONSORT battery trial and was able to reliably and accurately predict large peaks over anomalous holiday periods. This was in one instance able to assist the CONSORT project in coordinating the batteries on Bruny Island to avoid using the diesel generator.

The models were additionally evaluated on a standard ISO New England dataset. When using identical models to those applied on Bruny Island the sequence to sequence and transformer without teacher forcing models were able to achieve a MAPE of 2.15% compared to other papers of 1.71% and 1.31%.

The presented models – sequence to sequence and transformer without teacher forcing – are versatile and reliable when used to forecast load with a 24-hour horizon and 30- or 60-minute resolution. The models are practical to implement and can be applied to feeders ranging from the 1 MVA through 20 GW level with no modifications. The models can reliably predict anomalous holiday periods given simple inputs describing the holidays and optionally given similar load profiles from the past.

4.1 Future research

Transfer learning

OpenAI has had success using transfer learning for natural language processing tasks with the transformer model [75]. Transfer learning could be applied to load forecasting, whereby a single neural network model is first trained to forecast many different feeders, and then this pre-trained model is finally trained to forecast a single feeder. This could potentially expand the generalization ability of the model, while also likely reducing the amount of training time required when starting from a pre-trained model.

Time-weighted loss function

Section 3.1.5 indicated that the most recent training data may be the most important. It is possible to modify the loss function such that it weights samples from more recently higher. Whether this would be superior to simply restricting the training set to recent data is unknown.

Monthly and hourly models

Similar to Ceperic et al. [23], different models could be train for every month and every hour. This introduces 288 models, making the training time potentially quite large, so pre-training or transfer learning could be used to make this more approachable. This may not work well when considering holiday periods that change yearly, such as Easter which may be in either March or April.

Multi-task learning

Multi task learning is the process of using a neural network model to perform multiple tasks simultaneously, and has been shown to improve performance in some tasks [76]. This could applied to load forecasting by forecasting multiple feeders simultaneously. It’s intuitive that this might be beneficial, as there are many feeders which exhibit similar patterns – Bruny Island and St Helens in Tasmania, for example, both experience similar peaks around holiday periods. This could also allow the forecasting system to be more resilient against poor quality data, as data from one feeder would be used to help forecast others.

Generic forecaster

The transformer, and its multi-head self attention, is able to process sequences of long length with no regard for position or order. This could be used to create a generic forecaster, where the model is supplied with all past data up to the current time when performing forecasts. A model like this might be able to look over the past data and produce a single forecast that matches the past patterns. Using multi-head self attention for this would be prohibitively computationally expensive, so localised multi-head attention [30] could be applied, as used by Song et al. [62]. If a single model could be made to forecast all feeders this may ultimately be computationally cheaper than using a single model for every feeder.

Sequence to sequence with attention

Sequence to sequence models can be augmented with an attention mechanism [46] to allow the model to consider all elements of the input simultaneously while producing the output sequence. This may overcome the tendency of LSTM RNN networks to “forget” data from early in the input sequence. Very recently LSTM RNN models have been combined with transformer models [77], producing promising results. Applying combinations of LSTM and attention to load forecasting may produce superior results to either on their own.

Signal decomposition

Chen et al. [35] used wavelet decomposition to decompose input data into low and high frequency components before forecasting the next day’s low and high frequency load separately. Bedi and Toshniwal [51] used empirical mode decomposition to decompose input data into different intrinsic mode functions before applying different LSTM RNNs to each and recombining the results to form a final forecast. By applying methods such as these it would likely be possible to improve the accuracy of the load forecasting. However, it may jeopardize the ability for the forecasting models to be applied to different feeders with no modification.

References

[1] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning.* MIT Press, 2016, http://www.deeplearningbook.org.

[2] W. Johnston, “National survey report of photovoltaic applications in australia 2017,” 2017. [Online]. Available: http://apvi.org.au/pv-in-australia-2017/

[3] Bloomberg New Energy Finance, “New energy outlook 2018,” 2018. [Online]. Available: https://about.bnef.com/new-energy-outlook/

[4] W. Gerardi and D. O’Connor, “Projections of uptake of small-scale systems,” 2017. [Online]. Available: https://www.aemo.com.au/Electricity/National-Electricity-Market-NEM/Planning-and-forecasting/Electricity-Forecasting-Insights/2017-Electricity-Forecasting-Insights/Key-component-consumption-forecasts/PV-and-storage

[5] “2016 national electricity forecasting report,” 2016. [Online]. Available: https://www.aemo.com.au/Electricity/National-Electricity-Market-NEM/Planning-and-forecasting/National-Electricity-Forecasting-Report

[6] P. Scott and S. Thiébaux, “Distributed Multi-Period optimal power flow for demand response in microgrids,” in *ACM e-Energy*, Bangalore India, jul 2015. [Online]. Available: http://users.cecs.anu.edu.au/~pscott/extras/papers/scott2015.pdf

[7] CIGRE. (2016) Cigre working group concludes demand forecasting study. [Online]. Available: https://www.cigreaustralia.org.au/news/features/cigre-working-group-concludes-demand-forecasting-study/

[8] R. Weron, *Modeling and Forecasting Electricity Loads and Prices.* John Wiley & Sons Ltd, dec 2006.

[9] H. Hippert, C. Pedreira, and R. Souza, “Neural networks for short-term load forecasting: a review and evaluation,” *IEEE Transactions on Power Systems*, vol. 16, no. 1, pp. 44–55, 2001.

[10] AEMO, “Forecast accuracy report 2017,” 2017.

[11] S. Rahman and O. Hazim, "A generalized knowledge-based short-term load-forecasting technique," *IEEE Transactions on Power Systems*, vol. 8, no. 2, pp. 508–514, may 1993.

[12] T. Senjyu, S. Higa, and K. Uezato, "Future load curve shaping based on similarity using fuzzy logic approach," *IEE Proceedings - Generation, Transmission and Distribution*, vol. 145, no. 4, p. 375, 1998.

[13] T. Senjyu, P. Mandal, K. Uezato, and T. Funabashi, "Next day load curve forecasting using recurrent neural network structure," *IEE Proceedings - Generation, Transmission and Distribution*, vol. 151, no. 3, p. 388, 2004.

[14] C. Dou, Y. Zheng, D. Yue, Z. Zhang, and K. Ma, "Hybrid model for renewable energy and loads prediction based on data mining and variational mode decomposition," *IET Generation, Transmission & Distribution*, vol. 12, no. 11, pp. 2642–2649, jun 2018.

[15] G. E. P. Box, G. M. Jenkins, and G. C. Reinsel, *Time Series Analysis.* John Wiley & Sons, Inc., jun 1970.

[16] A. E. Desouky and M. E. Kateb, "Hybrid adaptive techniques for electric-load forecast using ANN and ARIMA," *IEE Proceedings - Generation, Transmission and Distribution*, vol. 147, no. 4, p. 213, 2000.

[17] J. W. Taylor and P. E. McSharry, "Short-term load forecasting methods: An evaluation based on european data," *IEEE Transactions on Power Systems*, vol. 22, no. 4, pp. 2213–2219, nov 2007.

[18] S. Arora and J. W. Taylor, "Short-term forecasting of anomalous load using rule-based triple seasonal methods," *IEEE Transactions on Power Systems*, vol. 28, no. 3, pp. 3235–3242, aug 2013.

[19] C. J. Bennett, R. A. Stewart, and J. W. Lu, "Forecasting low voltage distribution network demand profiles using a pattern recognition based expert system," *Energy*, vol. 67, pp. 200–212, apr 2014.

[20] S. Karthika, V. Margaret, and K. Balaraman, "Hybrid short term load forecasting using ARIMA-SVM," in *2017 Innovations in Power and Advanced Computing Technologies (i-PACT).* IEEE, apr 2017.

[21] N. Amjady, "Short-term hourly load forecasting using time-series modeling with peak load estimation capability," *IEEE Transactions on Power Systems*, vol. 16, no. 3, pp. 498–505, 2001.

[22] H. Drucker, C. J. C. Burges, L. Kaufman, A. Smola, and V. Vapnik, "Support vector regression machines," in *Proceedings of the 9th International*

Conference on Neural Information Processing Systems, ser. NIPS'96. Cambridge, MA, USA: MIT Press, 1996, pp. 155–161. [Online]. Available: http://dl.acm.org/citation.cfm?id=2998981.2999003

[23] E. Ceperic, V. Ceperic, and A. Baric, "A strategy for short-term load forecasting by support vector regression machines," *IEEE Transactions on Power Systems*, vol. 28, no. 4, pp. 4356–4364, nov 2013.

[24] A. RochaReis and A. AlvesdaSilva, "Feature extraction via multiresolution analysis for short-term load forecasting," *IEEE Transactions on Power Systems*, vol. 20, no. 1, pp. 189–198, feb 2005.

[25] N. AMJADY and F. KEYNIA, "Short-term load forecasting of power systems by combination of wavelet transform and neuro-evolutionary algorithm," *Energy*, vol. 34, no. 1, pp. 46–57, jan 2009.

[26] A. Deihimi and H. Showkati, "Application of echo state networks in short-term electric load forecasting," *Energy*, vol. 39, no. 1, pp. 327–340, mar 2012.

[27] E. E. Elattar, J. Goulermas, and Q. H. Wu, "Electric load forecasting based on locally weighted support vector regression," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 40, no. 4, pp. 438–447, jul 2010.

[28] M. Negnevitsky, *Artificial intelligence: a guide to intelligent systems.* Pearson Education, 2005.

[29] S. Sabour, N. Frosst, and G. E Hinton, "Dynamic Routing Between Capsules," *ArXiv e-prints*, Oct. 2017.

[30] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Curran Associates, Inc., 2017, pp. 5998–6008. [Online]. Available: http://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf

[31] C. Chiu, T. N. Sainath, Y. Wu, R. Prabhavalkar, P. Nguyen, Z. Chen, A. Kannan, R. J. Weiss, K. Rao, E. Gonina, N. Jaitly, B. Li, J. Chorowski, and M. Bacchiani, "State-of-the-art speech recognition with sequence-to-sequence models," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, April 2018, pp. 4774–4778.

[32] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. W. Senior, and K. Kavukcuoglu, "Wavenet: A generative

model for raw audio," *CoRR*, vol. abs/1609.03499, 2016. [Online]. Available: http://arxiv.org/abs/1609.03499

[33] P. Mandal, A. K. Srivastava, T. Senjyu, and M. Negnevitsky, "A new recursive neural network algorithm to forecast electricity price for PJM day-ahead market," *International Journal of Energy Research*, vol. 34, no. 6, pp. 507–522, may 2010.

[34] Z. Zhao, W. Chen, X. Wu, P. C. Y. Chen, and J. Liu, "LSTM network: a deep learning approach for short-term traffic forecast," *IET Intelligent Transport Systems*, vol. 11, no. 2, pp. 68–75, mar 2017.

[35] Y. Chen, P. Luh, C. Guan, Y. Zhao, L. Michel, M. Coolbeth, P. Friedland, and S. Rourke, "Short-term load forecasting: Similar day-based wavelet neural networks," *IEEE Transactions on Power Systems*, vol. 25, no. 1, pp. 322–330, feb 2010.

[36] W. Kong, Z. Y. Dong, Y. Jia, D. J. Hill, Y. Xu, and Y. Zhang, "Short-term residential load forecasting based on LSTM recurrent neural network," *IEEE Transactions on Smart Grid*, pp. 1–1, 2017.

[37] W. Kong, Z. Y. Dong, D. J. Hill, F. Luo, and Y. Xu, "Short-term residential load forecasting based on resident behaviour learning," *IEEE Transactions on Power Systems*, vol. 33, no. 1, pp. 1087–1088, jan 2018.

[38] N. Ding, C. Benoit, G. Foggia, Y. Besanger, and F. Wurtz, "Neural network-based model design for short-term load forecast in distribution systems," *IEEE Transactions on Power Systems*, vol. 31, no. 1, pp. 72–81, jan 2016.

[39] K. Hornik, M. B. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, pp. 359–366, 1989.

[40] L. Hernández, C. Baladrón, J. M. Aguiar, B. Carro, A. Sánchez-Esguevillas, and J. Lloret, "Artificial neural networks for short-term load forecasting in microgrids environment," *Energy*, vol. 75, pp. 252–264, oct 2014.

[41] X. Sun, P. B. Luh, K. W. Cheung, W. Guan, L. D. Michel, S. S. Venkata, and M. T. Miller, "An efficient approach to short-term load forecasting at the distribution level," *IEEE Transactions on Power Systems*, vol. 31, no. 4, pp. 2526–2537, jul 2016.

[42] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in neural information processing systems*, 2014, pp. 3104–3112.

[43] J. Taylor. Statsmodels. [Online]. Available: statsmodels.org

[44] L. Bottou, “Large-scale machine learning with stochastic gradient descent,” in *Chapman & Hall/CRC Computer Science & Data Analysis*. Chapman and Hall/CRC, dec 2011, pp. 17–25.

[45] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors,” *Nature*, vol. 323, no. 6088, pp. 533–536, oct 1986.

[46] M.-T. Luong, H. Pham, and C. D. Manning, “Effective approaches to attention-based neural machine translation,” *arXiv preprint arXiv:1508.04025*, 2015.

[47] T. Yao, Y. Pan, Y. Li, Z. Qiu, and T. Mei, “Boosting image captioning with attributes,” in *IEEE International Conference on Computer Vision, ICCV*, 2017, pp. 22–29.

[48] F. Karim, S. Majumdar, H. Darabi, and S. Chen, “Lstm fully convolutional networks for time series classification,” *IEEE Access*, vol. 6, pp. 1662–1669, 2018.

[49] H. Shi, M. Xu, and R. Li, “Deep learning for household load forecasting—a novel pooling deep rnn,” *IEEE Transactions on Smart Grid*, vol. 9, no. 5, pp. 5271–5280, Sept 2018.

[50] J. Zheng, C. Xu, Z. Zhang, and X. Li, “Electric load forecasting in smart grids using long-short-term-memory based recurrent neural network,” in *2017 51st Annual Conference on Information Sciences and Systems (CISS)*. IEEE, mar 2017.

[51] J. Bedi and D. Toshniwal, “Empirical mode decomposition based deep learning for electricity demand forecasting,” *IEEE Access*, vol. 6, pp. 49 144–49 156, 2018.

[52] A. Narayan and K. W. Hipel, “Long short term memory networks for short-term electric load forecasting,” in *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE, oct 2017.

[53] G. M. U. Din and A. K. Marnerides, “Short term power load forecasting using deep neural networks,” in *2017 International Conference on Computing, Networking and Communications (ICNC)*. IEEE, jan 2017.

[54] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[55] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, “Empirical evaluation of gated recurrent neural networks on sequence modeling,” 12 2014.

[56] R. Jozefowicz, W. Zaremba, and I. Sutskever, “An empirical exploration of recurrent network architectures,” in *International Conference on Machine Learning*, 2015, pp. 2342–2350.

[57] Q. V. Le, N. Jaitly, and G. E. Hinton, “A simple way to initialize recurrent networks of rectified linear units,” *arXiv preprint arXiv:1504.00941*, 2015.

[58] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, “Learning phrase representations using rnn encoder-decoder for statistical machine translation,” 2014.

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

[60] G. Chevalier. (2017) Sequence to sequence (seq2seq) recurrent neural network (rnn) for time series prediction. [Online]. Available: github.com/guillaume-chevalier/seq2seq-signal-prediction

[61] M. Dehghani, S. Gouws, O. Vinyals, J. Uszkoreit, and Ł. Kaiser, “Universal Transformers,” *ArXiv e-prints*, Jul. 2018.

[62] H. Song, D. Rajan, J. Thiagarajan, and A. Spanias, “Attend and diagnose: Clinical time series analysis using attention models,” 2018. [Online]. Available: https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16325

[63] N. Parmar, A. Vaswani, J. Uszkoreit, L. Kaiser, N. Shazeer, and A. Ku, “Image transformer,” *CoRR*, vol. abs/1802.05751, 2018. [Online]. Available: http://arxiv.org/abs/1802.05751

[64] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016, pp. 770–778.

[65] J. L. Ba, J. R. Kiros, and G. E. Hinton, “Layer normalization,” *arXiv preprint arXiv:1607.06450*, 2016.

[66] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: A simple way to prevent neural networks from overfitting,” *J.*

Mach. Learn. Res., vol. 15, no. 1, pp. 1929–1958, Jan. 2014. [Online]. Available: http://dl.acm.org/citation.cfm?id=2627435.2670313

[67] A. M. Lamb, A. G. A. P. GOYAL, Y. Zhang, S. Zhang, A. C. Courville, and Y. Bengio, “Professor forcing: A new algorithm for training recurrent networks,” in *Advances In Neural Information Processing Systems*, 2016, pp. 4601–4609.

[68] K. Park. (2017) A tensorflow implementation of the transformer: Attention is all you need. [Online]. Available: github.com/Kyubyong/transformer

[69] C. Wang and J. Principe, “Training neural networks with additive noise in the desired signal,” *IEEE Transactions on Neural Networks*, vol. 10, no. 6, pp. 1511–1517, 1999.

[70] W. M. Brown, T. D. Gedeon, and D. I. Groves, *Natural Resources Research*, vol. 12, no. 2, pp. 141–152, 2003.

[71] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014.

[72] Iso new england. [Online]. Available: www.iso-ne.com

[73] N. C. for Environment Information. National centers for environment information. [Online]. Available: www.ncdc.noaa.gov/data-access

[74] M. Zinkevich. (2018) Rules of machine learning: Best practices for ml engineering. [Online]. Available: developers.google.com/machine-learning/guides/rules-of-ml/

[75] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, “Improving language understanding by generative pre-training,” *URL https://s3-us-west-2. amazonaws. com/openai-assets/research-covers/language-unsupervised/language_ understanding_paper. pdf*, 2018.

[76] T. Evgeniou and M. Pontil, “Regularized multi–task learning,” in *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD ’04. New York, NY, USA: ACM, 2004, pp. 109–117. [Online]. Available: http://doi.acm.org/10.1145/1014052.1014067

[77] M. X. Chen, O. Firat, A. Bapna, M. Johnson, W. Macherey, G. Foster, L. Jones, N. Parmar, M. Schuster, Z. Chen *et al.*, “The best of both worlds: Combining recent advances in neural machine translation,” *arXiv preprint arXiv:1804.09849*, 2018.