A Residual-Based Ensemble LSTM Model for Intermittent Electricity Consumption Forecasting

QPTS1[1]

Master of Science in Machine Learning

Jose Alcala
Dmitry Adamskiy

Submission date: 12 September 2022

[1]**Disclaimer:** This report is submitted as part requirement for the Master of Science in Machine Learning at UCL. It is substantially the result of my own work except where explicitly indicated in the text. The report will be distributed to the internal and external examiners, but thereafter may not be copied or distrbuted except with permission from the author.

Abstract

The changing demand for electricity is nowadays a major focus of extensive research. It can help us to reduce resource consumption, control household expenditure and much more. With the rise and development of the smart home, many researchers have begun to look at more detailed estimation of the electricity consumption of household appliances and apply it to applications such as appliance management and life pattern estimation. Our project collects the electricity consumption of each of the eight appliances commonly used in households over a two-year period. Based on the data collected, we build a deep learning model for multi-step forecasting. This model is based on the N-BEATS model, an ensemble model based on the calculation of residuals. And our model change the base weak learner as LSTM-based model, which we name it as N-BEATS-LSTM. Compared to our proposed benchmark, our model has a 33% performance improvement, ranging from 3% to 30% improvement compared to other deep learning models. In our experiments, we compared the contribution of various base model components to the overall performance of the model. At the end, we analysed the output of the N-BEATS-LSTM and found how residual computation works in the model.

For full code repository, please visit:
Github: https://github.com/informetis/uk-ucl-2022

Acknowledgements

I am proud to have had a fun and fruitful summer with Informetis. Thanks to my colleagues, Jose Alcala, Boruo Xu and Mica Menard. They have provided countless of advice and guidance during my project. Especially, as main advisor, Mica helped me with many issues, no matter the choice of models or difficulties with the code. I would also like to thank Prof. Dmitry Adamskiy for being the internal supervisor to monitor the progress of my project and remind me of the time to prepare my report. I am very happy to end my Masters studies with such a project and people.

Contents

Chapter 1

Introduction

In this chapter, we describe the background and motivation for the topic and briefly describe the research that has been done in this context. We will then present the problem that the paper needs to address and describe the objectives of the project. Finally, the overall framework of this thesis will be presented.

1.1 Motivation

Energy has been a crucial strategic resource that humans rely on for their livelihood ever since the industrial revolution. In the early days, energy relied on natural resources such as coal and oil, while in the last two centuries, electricity has become the primary source of energy. However, there is a limit to the amount of natural resources that can be used to produce electricity, and even though new clean energy sources such as wind and solar are being investigated, there is still under tremendous pressure on humanity to use electricity the environmental issues brought on by burning fossil fuels[40, 59, 17, 5, 67]. Therefore, being able to manage and use limited electrical energy well is a good solution to the energy problem.

Both horizontally and vertically have been investigated by researchers for the management and control of power use. The researchers concentrate on the creation of numerous applications on a horizontal scale. Advanced intelligent power grid systems adjust the supply of electricity in different areas through changes in real-time electricity consumption in order to save available power[28, 61]. Many buildings use air conditioning and lighting systems that are controlled by sensing the number of people in the space[66, 19, 25]. The state or the electricity company adjusts generation plans or tariffs by predicting future electricity demand[51, 27, 68, 64, 11]. The vertical dimension is studied from the macro to the micro level, from the country, the company or even the world, down to the individual building and household[50, 43, 41, 53, 29].

With the emergence of smart homes in recent years, focusing in a single household, predicting and regulating the electricity consumption of particular appliance in a home has also grown significantly in attention[13, 8]. The difficulty in managing the electricity consumption of individual appliances compared to macro electricity consumption is that there is less active data and a greater variation in the range of data. Therefore, to find a suitable model for estimating the expected electricity consumption of appliances has become the major concern of the research.

1.2 Objectives

For the data we have collected, we find that the problem we are dealing with belongs to a particular type of data in the time series, the intermittent demand problem[74]. This type of data contains a large amount of inactive data, such as zero demand in the demand calculation. This type of data is discrete and there are long time intervals between active states. Correspondingly, most of the appliances in our data are in standby mode, while the electricity consumption we really want to estimate is spread over a very small number of time stamps. We therefore also need more specific models to solve this problem.

Our project focused on commonly used household appliances. We collected data on the electricity consumption of eight appliances commonly used in the home over a 2-year period. Using this data, we will build machine learning models to identify the usage patterns of these appliances in the home and attempt to predict future electricity consumption. We need to build a machine learning model that enables multi-step prediction of future electricity consumption for various appliances. That is, our model needs to simultaneously predict the electricity consumption of multiple appliances at multiple timestamps in the future.

Artificial Neural Network(ANN) has achieved great success in a lot of fields in the past decades. And time series prediction using ANNs has also become a new way of research. Recurrent Neural Network(RNN)[31] is a type of ANN designed specifically for time series. Some variations of RNN, such as Long Short-Term Memory(LSTM)[31, 37], models, Gated Recurrent Units(GRU)[31, 21], are still the main backbone models in the latest research. And in recent years, with the advent of attention mechanisms, ANNs have taken the time series problem a step further.

Therefore, our project will use RNN to construct time series forecasting models. Generally speaking, our project will be divided into the following parts:

1. As there are many models available for time series forecasting, it is not straightforward to determine which model will achieve our objectives. So the first task of this project is to identify a model that is suitable for the data we have collected. We needed to compare a number of established models and select the best one in terms of performance, consumption and other dimensions. Since many classical machine learning algorithms can only focus on single-step prediction, our experiments will focus on ANNs and compare them with the benchmarks we have set up to select the optimal model.

2. After the optimal model has been obtained, we will tune the adjustable parameters. The model performance is tuned to be relatively optimal by hyper-parameter tuning. We will then analyse the tuned model and combine the actual data with the predicted results to analyse the feasibility of the model in practical use.

1.3 Our Contribution

1. For intermittent demand data, we propose the method of demand normalization to distribute demands of a single timestamp to adjacent timestamps. To make the data as continuous as possible.
2. A loss function SlicedMSE is proposed to reduce the amount of computation by calculating the mean square error over a time period, allowing parameter updates to be more focused on demand points. Two metrics, SlicdeMAE and ActiveMAE, are designed to calculate the error between forecast and actual demand in a time period and on demanded timestamps separately.
3. By comparing different deep learning models, we found the contribution of various different deep learning components, such as convolutional layers, attention mechanisms, etc., to the models.
4. for intermittent demand, an ensemble model based on residual calculation is designed and shown to outperform other deep learning models for the data we have collected. The validity of the residuals is also explored in depth.

1.4 Thesis Structure

In this chapter, we briefly describe the motivation for this project and what we are trying to achieve. In chapter 2, we present a summary of existing research on electricity consumption estimation and introduce several existing time series forecasting models. Chapter 3 contains details of the construction of the various models we wish to compare. Chapter 4 presents the data we collected and experiments, including the training process, parameter settings and experimental results. The final chapter summarises our project, identifies the shortcomings of the project, and makes conjectures for future research.

Chapter 2

Related Work

In this chapter we will first introduce the machine learning problem, regression analysis, and introduce the particular branch of the regression problem we are facing, the intermittent demand problem. This is followed by an introduction to some of the models that are currently used to solve this problem. Finally, we'll go back to applications and discuss recent research on estimating electricity usage, including macro studies on national or regional consumption as well as micro applications concentrating on households or specific appliances.

2.1 Time-series Data and Regression Analysis

Time series Forecasting aims to estimate future values or probability distributions based on a given time series of observations. In general terms, a time series is a set of random variables ordered in time, usually the result of observations of a potential process at a given sampling rate over an equally spaced period of time. Regression analysis, as one of the basic solutions for time series forecasting, is a very important method in supervised learning of machine learning.

However, the regression models we employ vary depending on the data. The data in a time series indicate that past data have implied a pattern of change in the present or future data, including trending, seasonality, or irregularities[39]. Trending reflects the direction of a time series over a long period of time, and can be characterised by a continuous upward or downward trend over a significant period of time, or by a steady trend. Seasonality reflects a fixed length and amplitude of fluctuations in a time series affected by various cyclical factors. Irregularity reflects the non-trending and non-periodic irregularity of a time series that is influenced by various unexpected events and contingencies.

The task of regression is that given an input vector, the model has to output the target vector. It usually reflects the relationship between the input vector and the target vector[39, 12]. In mathematical terms, this problem can be described as $Y = F(X, w)$, where X denotes the input vector in D_{in}-dimensional N_{in}-steps, Y denotes the target vector in D_{out}-dimensional N_{out}-steps, and w denotes the parameters of the model.

2.1.1 Linear and Non-Linear Regression

The most basic regression model is the linear regression model, where the target vector is considered to be a linear combination of the input vectors[12, 39], which can be written as:

$$Y(X, w) = w_{bias} + \sum_i w_i X_i \tag{2.1}$$

But the disadvantage of the linear model is that it cannot predict seasonal or irregular data because of its linear character. If one wants to predict a different type of data, then one needs to transform the linear operation into a non-linear operation, written as:

$$Y(X, w) = w_{bias} + \sum_i w_i \phi_i(X_i) \tag{2.2}$$

where ϕ denotes a non-linear function. We have many options for non-linear equations such as polynomials, exponential families, etc. Even for many seasonal data, we can use trigonometric functions as non-linear equations. The most common method we use for solving regression problems is the least Mean Squared Error method. For linear and non-linear regression we can write the equation of the squared error w.r.t parameters w[12, 71, 39]:

$$E(w) = (Y_{true} - Y(X, w))^2 \tag{2.3}$$

Our goal is therefore to modify the parameters to achieve the minimum error. Then, based on the principle of gradient descent, we can write the parameter update equation as:

$$w^{(i+1)} = w^{i} - \alpha \bigtriangledown E(w) \tag{2.4}$$

Where α refers to the learning rate and represents the degree of each time we want the parameters to change[12, 69]. Based on these two regression methods we have also obtained many variations, such as Lasso regression[76] or Ridge regression[38] with a regularization or penalty term. But at the same time the disadvantages of these methods are obvious. All of these methods are based on the fact that we have a basic understanding of the data and we assume that they have a certain distribution or that their shape on the time axis will resemble a function of some kind that we are familiar with. This is how we can choose the right mapping function to fit the data we have collected. Thus we will need more methods to solve for irregular data in time series.

2.1.2 Support Vector Regression

Support Vector Machines(SVM) was originally used for classification problems such as pattern recognition, and was later extended to non-linear regression estimation and curve fitting, which also performed well and called Support Vector Regression(SVR). SVMs have been proposed and developed, and then extended to different areas such as regression tasks by Vapnik and co-workers in the 1990s[26, 22].

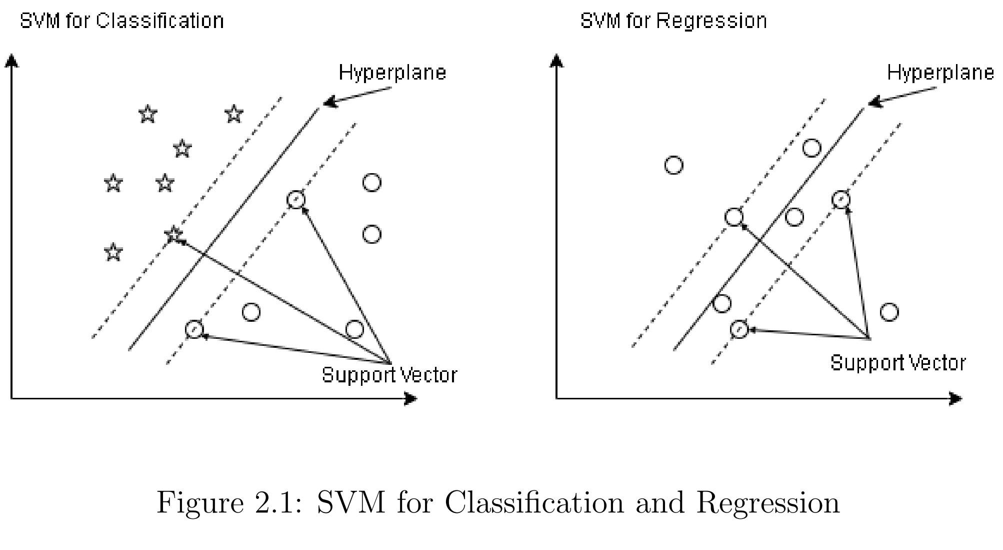


Figure 2.1: SVM for Classification and Regression

The core idea of SVM classification is to find hyperplanes between categories that can distinguish between categories within a certain error (e.g. Figure 2.1 Left). On the contrary, in the SVM regression task, we need to find hyperplanes that contain the data within a certain error as much as possible (e.g. Figure 2.1 Right). The data points distributed on the acceptable error plane are called support vectors. The advantage of using SVMs for regression tasks is that the curves we fit are only related to the support vectors. So our curve can be of any shape, even if we can't describe it with a familiar function.

Further, because it is not possible for all data to be linearly separable, many SVM studies have used kernel tricks to achieve regression or classification of non-linear data. The use of kernel tricks enables the original data points to be mapped into a new higher dimensional space, perhaps better able to find the hyperplane in a higher space.

As a result, SVR does demonstrate his strengths in irregular data regression problems, but his weaknesses are also apparent. When we are solving linearly indistinguishable data, we need to use kernel tricks to create higher dimensional spaces, which means we need more memory space and time. So SVR does have a good performance on small datasets, but when it' s applied to larger datasets it may no longer be applicable.

2.1.3 Ensemble Learning for Regression

Random Forest

Random forest regression is also a model that, like SVR, was first applied to classification tasks and later transplanted to regression tasks. It belongs to the Bagging method of ensemble learning algorithms, where the general idea is to train multiple weak learners to pack together to form a strong model. In the training phase, the random forest uses bootstrap sampling to collect several different sub-training datasets from the input training dataset to train several different decision trees in turn. In the prediction phase, the random forest averages the predictions of the internal decision trees to obtain the final result[35, 36]. RF is expressed as Equation(2.5), F stands for the whole model and f for the weak learner.

$$F(w, X) = \frac{1}{N}\sum_{i}^{N} f_i(w, X) \tag{2.5}$$

In general we use Classification And Regression Tree (CART) as a weak learner to compose RF. CART generates a binary tree based on different generative and pruning strategies, with each leaf outputting a prediction. However, for our regression problem, the predictions will generally be distributed over the entire real number domain. In this way, the number of leaf nodes may tend to be infinite, consuming very large amounts of memory. On the other hand, RF reduces the variance of the model by Bagging, but if the size of CART is not controlled, it can easily cause the model to be overfitted.

Gradient Boosting Decision Tree(GBDT)

Another important approach to ensemble learning, the Boosting regression model is even better than Bagging. The most notable of these is the Gradient Boosting Decision Tree (GBDT). Gradient Boosting is based on the idea of gradient descent, where the basic principle is to train new weak learner based on the negative gradient information of the loss function of the current model, and then to combine the trained weak learner into the current model in an accumulative form[34].

In contrast to the Bagging approach, which makes weak learners more diversity through Bootstrap, Boosting considers errors as an optimisation goal. Adaboost, for example, is to increase the weighting of error examples[30, 33]. The Gradient Boosting approach, on the other hand, goes a step further and aims to optimise the whole model in the function space[62]. In each iteration, the Gradient Boosting algorithm first calculates the negative gradient of the current model over all samples, and then uses this value as the target to train a new weak classifier to fit and calculate the weights of this weak learner, and finally achieve the update of the model. GBDT is expressed as Equation(2.6). In contrast to RF's, the result of GBDT is a weighted sum of weak learners, each of which has a corresponding weight α.

$$F(w, X) = \sum_{i}^{N} \alpha_i f_i(w, X) \tag{2.6}$$

GBDT still uses CART as a weak learner. Unlike RF, the training of each weak learner in GBDT requires the result of the previous training as the optimization target and therefore cannot be computed in parallel. Therefore, many researchers have improved on this problem, providing algorithms such as XGBoost[18] and LightGBM[42] that can support parallel computation. Because the Boosting algorithm updates the weak learner based on errors, GBDT is more prone to overfitting compared to RF, even if we control the size of the weak learner.

2.1.4 Artificial Neural Networks for Regression

Researchers have discovered that neurons can collaborate with each other to process and transmit information, thus completing the process of thinking. Since the basis of thinking is neurons, if we can make artificial neurons, we can form Artificial Neural Networks(ANNs) to simulate thinking. The first artificial neurons were perceptrons, which were invented in 1943 by McCulloch and Pitts[23]. Figure 2.2 [1] shows the connection between human neurons and perceptron.

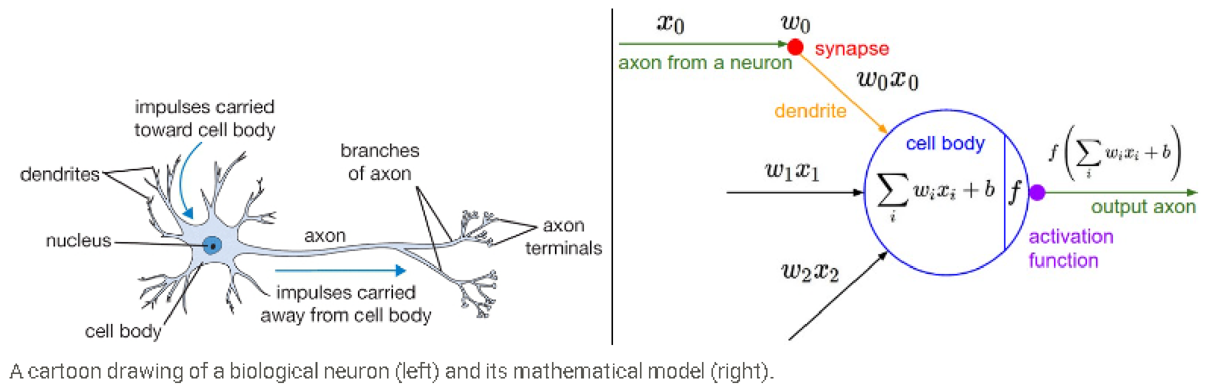


Figure 2.2: From Neurons to Perceptron (*extracted from Stanford class CS231n*)

Later on, researchers gradually combined and stacked several perceptrons together so that Multi-Layer Perceptrons(MLPs) emerged, which is the prototype of ANN[34]. As time went on, more artificial neurons were invented, such as convolutional kernels and recurrent kernels. As the researchers deepen the layers of the network, the ANN becomes a Deep Neural Network(DNN). The power of DNNs lies in their ability to capture a large amount of information through a large number of neurons, and therefore to tackle a wide range of problems such as pattern recognition, image recognition and regression analysis. So in order to investigate whether ANNs can accomplish what this project is trying to achieve, we have also selected a number of ANNs to compare and verify their strengths.

Convolution Neural Network

Convolutional neural networks (CNNs) are a powerful class of neural networks designed for multiple kinds of data especially for image data[52]. For simple MLPs, it may be good

[1]CS231n: Convolutional Neural Networks for Visual Recognition https://cs231n.github.io

for processing data of low dimensionality. But for data such as images, a single 100*100 channel can be considered a 10,000-dimensional vector, which makes MLPs considerably more difficult to compute. With the introduction of convolutional kernels, the original image information is transformed into a new feature map with a partial vision, so that models based on convolutional neural network architectures have come to dominate the field of computer vision.

In mathematics, the convolution between two functions (e.g. $f, g : \mathbb{R}^n \rightarrow \mathbb{R}$) is defined as:

$$(f * g)(\mathbf{x}) = \int f(\mathbf{z})g(\mathbf{x} - \mathbf{z})d\mathbf{z} \tag{2.7}$$

When it is a discrete item, the integral becomes a summation:

$$(f * g)(i) = \sum_a f(a)g(i - a) \tag{2.8}$$

We take the function g as the convolution kernel. The convolution of a function f at point i is a function g flipped ($g(a) \rightarrow g(-a)$), a translation to i ($g(-a) \rightarrow g(i - a)$) and then a superposition (summation or integration) over a certain range a. Through this series of transformations, we then map the original function f into a new function$f * g$. The process of training a CNN is in fact a continuous optimization of the convolution kernel to achieve the goal we want to achieve. A complete CNN usually consists of a convolutional layer for feature extraction, a pooling layer for integrating feature information within a region, and a fully-connected layer for mapping the extracted feature information to the result we want[52, 49].

When tackling time series problems, we usually use a 1-Dimensional convolutional layer. As shown in Figure 2.3, although it is a 1-Dimensional convolutional layer, it can still handle 2-dimensional multi-feature time series. The 1-Dimension here refers more to the fact that the kernel moves in only one dimension, which for a time series is in the time dimension. Each convolutional kernel scans the entire time series in the time direction, generating new features. The information contained in the new features comes from multiple points in time, depending on the size of the convolutional kernel. Also, we can add padding on both sides of the data. Padding has the nice property of controlling the size of the data (e.g. to keep the input and output timestamps are equal).

Research on 1-Dimensional convolutional layers in time series has mainly focused on signal processing, where researchers have often used them as filters and feature extractors[44,

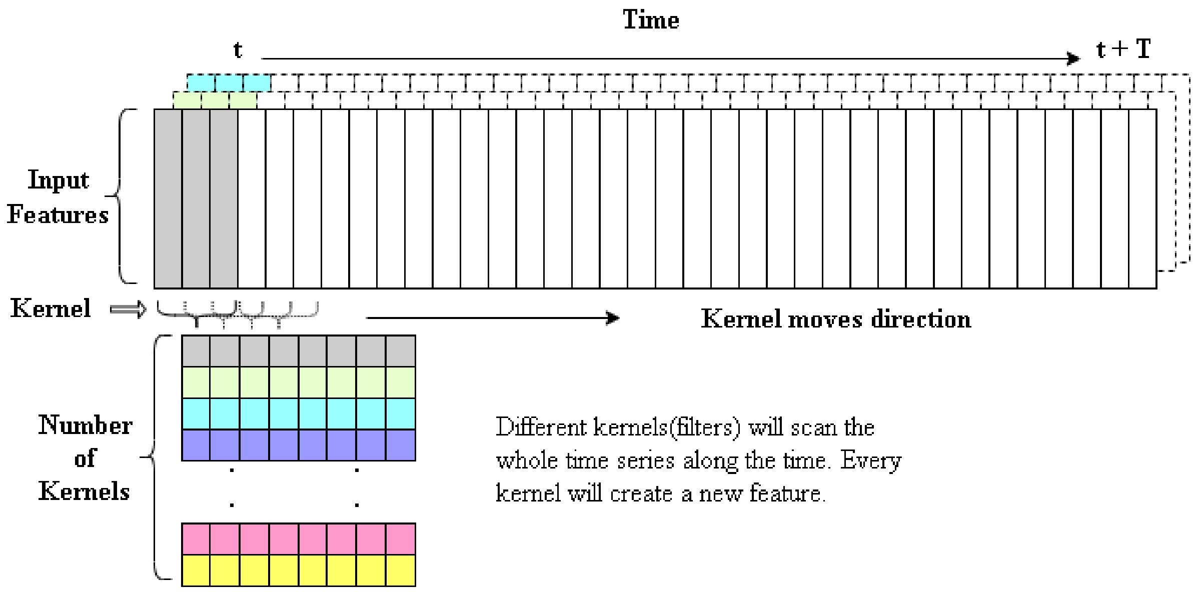

Time
t
t + T
Input Features
Kernel
Kernel moves direction
Number of Kernels
Different kernels(filters) will scan the whole time series along the time. Every kernel will create a new feature.

Figure 2.3: 1-Dimensional Convolutional Layer

45, 47, 52]. The use of 1D convolutional layers for time series prediction is still relatively unexplored. This is mainly because convolution is more like a filter to extract features from a time series, but we are not sure how the features relate to the future, so it seems unconvincing. Thus there have been many researchers combining CNNs and neural networks targeting time series, as will be explained next, to achieve the goal.

Recurrent Neural Network

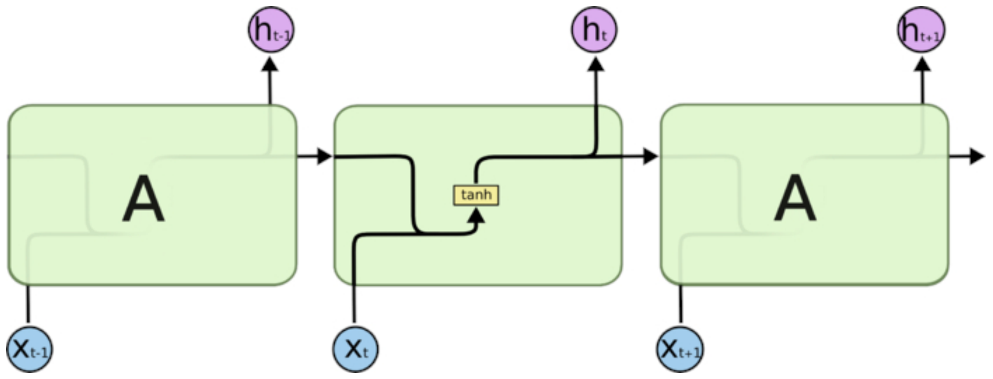


Figure 2.4: Structure pf RNNs (cite from [3])

In a traditional ANN such as a CNN, the information flows from the input layer to the hidden layer to the output layer, but the nodes within each layer are not connected to each other. Therefore their information is not shared, and when dealing with time series problems it also means that each neuron does not know what happened to the previous neuron. This is why Recurrent Neural Networks were(RNN) born[]. This network remembers the previous information and applies it to the current output calculation, which means that the nodes between the hidden layers are connected. As shown in Figure 2.4, the input to the hidden layer includes not only the output of the input layer (X_t) but also the output of the hidden layer (h_{t-1}) at the previous moment. In mathematical way, the out put of RNN can be written as:

$$O_t = F(w_o, h_t) \tag{2.9}$$

$$h_t = tanh(f(w_h, h_{t-1}) + g(w_x, X_t)) \tag{2.10}$$

The output O_t of this one cell is the functional mapping of the output h_t of the hidden layer. But the hidden layer output h_t is a weighted sum of the previous cell's output h_{t-1} and the current input X_t. In this way, the RNN connects the cells to each other. Each time a cell receives information it contains the previous information, so it is able to achieve the task better for data that is linked before and after. The disadvantage of RNNs, however, is that we do not know how much of the previous information we have received has been retained. For example, at timestamp t, it is not clear whether the information at timestamp $t-5$, $t-10$ or $t-100$ has been retained or not. This leads us to the next model, the long and short-term memory model.

Long-Short Term Memory Model

Long Short-Term Memory (LSTM) is still a type of RNN and was specifically designed to solve the problem of long-term dependency that exists in general RNNs[37]. In contrast to RNNs, LSTMs are suitable for processing and predicting important events with very long intervals and delays in time series. The structure of the LSTM is shown in Figure 33.

As you can see straight away from the diagram, there are two major differences in the LSTM, one is that more computation is added to the cell and the process is more complex. More importantly, the RNN only passes one value to the next cell, but in the LSTM there is another additional value, which is called long-term memory. We can say that the output of a cell consists of two parts, the output of the current cell to which long-term information has been added (we can say that it is short-term memory) and the long-term information

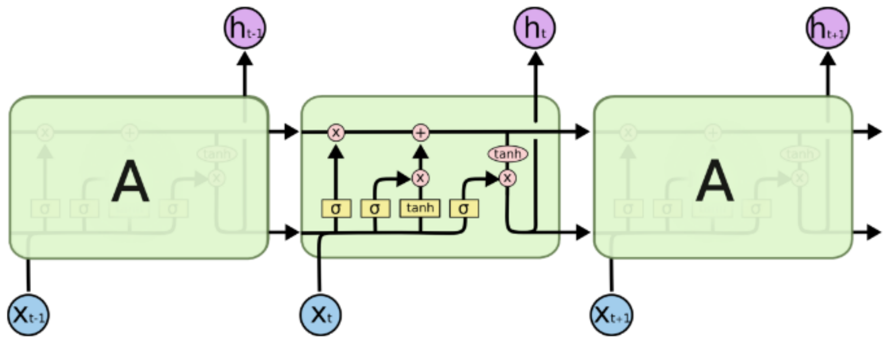
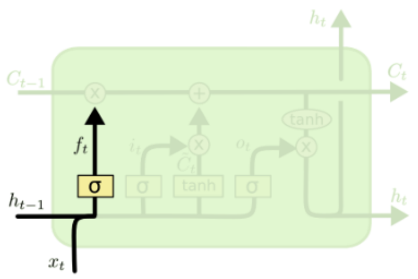


Figure 2.5: Structure pf LSTMs (cite from [3])

that has been processed by the current cell. This is where the name LSTM comes from. The cells of an LSTM contain four different gates implementing different functions.

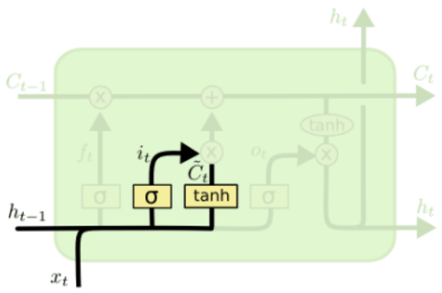


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Figure 2.6: Forget Gate (cite from [3])

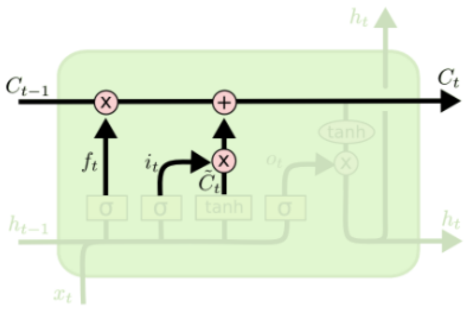
The first is the forget gate. In short, the forget gate determines how much information should be left behind in the long term. The current input X and the output of the previous cell are mapped to recent information between 0 and 1 by a weighting function and then by a *sigmoid* function. This result is then multiplied by the long-term memory and therefore determines whether the long-term memory is forgotten or not (0 means completely forgotten, 1 means completely remembered).

The input gate handles the fusion of the data from the current cell with the output from the previous cell. It uses two different activation functions, *tanh* and *sigmoid*. *tanh* maps the short-term information between -1 and 1, while the output of *sigmoid* is between 0 and 1. Since the outputs of the two are also multiplied, we can say that *tanh* is responsible for integrating the information, while *sigmoid* is responsible for giving each piece of information a weight to determine exactly what information is useful. Only then is it added to the long-term memory.



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \;+\; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \;+\; b_C)$$

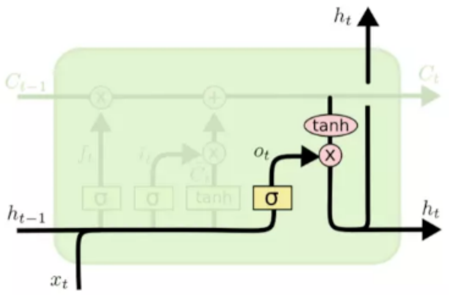
Figure 2.7: Input Gate (cite from [3])



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Figure 2.8: Update Gate (cite from [3])

The update gate then updates the results of the forget and input gates into long-term memory. It is important to note that these two operations are not computed in parallel. The first is to decide which of the previous memories need to be forgotten before the current information is added.



$$o_t = \sigma\left(W_o \; [h_{t-1}, x_t] \;+\; b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Figure 2.9: Output Gate (cite from [3])

The output gate, on the other hand, serves to integrate long-term memory and short-term input. Using the same idea as the input gate, the *sigmoid* function generates weights and the *tanh* function generates information. We can see that the information in the current cell comes mainly from long-term memory, because we have added the current information to the long-term memory in the input. The short-term input determines which of the information in long-term memory is needed for the current output.

So LSTM achieves better processing of time series data through the interplay of long-term memory and short-term input. Therefore LSTM also dominates research in various time-related fields such as finance, climate, and also electricity.

Attention Mechanism

The attention mechanism was first used in the field of computer vision, but gained increasing attention after Ashish et al. proposed the Transformer model[79, 10]. The attention mechanism is much like the human logic of looking at pictures, when we look at a picture, we do not see the whole picture, but focus on a particular focal point of the picture. The Attention mechanism has three very important vectors, a Query vector, a Key vector and a Value vector. We can form a pair of Key and Value, and after a certain relationship between Key and Query is satisfied, we take out the Value value corresponding to the key, i.e. the Attention value.

The relationship between Key and Query determines different kinds of attention mechanisms, called matching score. There are also different models depending on how the matching score is calculated, such as the Additive score proposed by Bahdanau[10]. The most used model is the Scaled Dot-Product score proposed by Vaswani[79]. It has been expressed as:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.11}$$

We use a set of Query(form matrix Q) and a set of Key(form matrix K) dot-products, then apply a $softmax$ to that result to get a set of weights, apply that weight to a set of Values(form matrix V), and we get our attention value. When dimension of the $K(d_k)$ is large, the dot product result will become large and its variance will also be large. This will make the gradient of the $softmax$ function unusually small, so dividing the dot product result by d_k will counteract this effect.

In Ashish et al.'s model, they also use a structure of attention called multi-head attention[79]. It produces several scaled dot product function with different parameters, which are integrated to give a more diverse attention method. In mathmetical way:

$$\text{MultiHead}(Q, K, V) = \text{Concat}\left(\text{head}_1, \ldots, \text{ head }_{\mathrm{h}}\right) W^O \tag{2.12}$$
$$\text{where head } = \text{Attention}\left(Q W_i^Q, K W_i^K, V W_i^V\right) \tag{2.13}$$

Where the projections are parameter matrices $W_i^Q \in \mathbb{R}^{d_{\text{maxde}} \times d_k}, W_i^K \in \mathbb{R}^{d_{\text{madel}} \times d_k}, W_i^V \in \mathbb{R}^{d_{\text{madd}} \times d_v}$ and $W^O \in \mathbb{R}^{h d_v \times d_{\text{mads}}}$. They are the parameters in the attention network that can be modified by training.

Next we should determine the source of the Key, Query, and Value. Typically we will use Self-Attention, which simply means that the Query and Value are generated internally by the model itself and are intrinsically linked. For example, in a Seq2Seq structure, the result produced by the Encoder can be considered a Query, while the result produced by the Decoder can be considered a Value. The Key, on the other hand, is generally equivalent to the Query. In the case of prediction tasks, we can use the known time series as the Query and the future series to be output as the Value.

The attention mechanism has a number of advantages. Firstly we note that all the computations are matrix operations and can therefore be computed in parallel as with CNNs. Secondly, the LSTM still suffers from memory being forgotten over time, but the attention mechanism brings the whole Query into the computation. So the attention mechanism solves the problem of long memory while still getting the global information.

2.2 Intermittent Demand Problem

Intermittent demand first originated in commodity production transactions, where certain items were found to have random demand with a large number of zero demand in most of time[74, 63, 48, 80]. Many of these problems can be converted to continuous demand in many cases, and such conversions are still relevant for practical problems, such as the sale of goods, where counting hourly and minute sales may be intermittent, while counting daily, weekly and monthly sales may be continuous. So this is precisely the first way of solving intermittent demand. But there are still many situations that cannot be converted into continuous demand. For example, if we estimate the demand for electrical appliances, such as ovens, which may only be used 1-2 times a day, our forecasts would be meaningless if they were converted to continuous data in terms of days.

There is a lack of research on intermittent demand and it is concentrated in areas such as production retailing and tourism management[56, 57, 9, 63]. As a result, the models for this problem are not as diverse as for ordinary regression analysis problems. The main approaches are mathematical models such as the Croston Method and deep learning methods such as N-BEATS.

2.2.1 Croston Method

The Croston Method is a sophisticated demand forecasting method based on Exponential Smoothing(ES). The ES is one of the important methods for forecasting continuous demand[39]. If one of the most rudimentary forecasting methods is followed, the forecast for the future is the data at the last point in time ($\hat{y}_{T+1|T} = y_T$). A further method is to use the average of all previous observations as a forecast ($\hat{y}_{T+1|T} = \frac{1}{T}\sum_T^t y_t$).

The ES method, on the other hand, is a combination of these two methods. Predicted values are calculated using a weighted average where the weights decrease exponentially with the time of the observation. It was expressed as:

$$\hat{y}_{T+1|T} = \alpha y_T + \alpha(1-\alpha)y_{T-1} + \alpha(1-\alpha)^2 y_{T-2} + \alpha(1-\alpha)^3 y_{T-3} + \cdots, \tag{2.14}$$

Such an idea has a high similarity to LSTM, where long-term memory is always incorporated into the calculation, even if only a small weight is given. But when the ES method deals with intermittent demand problems, he does not treat them specifically for long intervals. When the interval is large, we can find that the demand in the distant data will be erased because his weight will be very small. Thus the Croston Method was proposed[24, 80].

The Croston Method separates the demand interval from the quantity demanded and uses ES to calculate the demand interval and quantity demanded separately: if demand occurs, the estimates of the demand interval and quantity demanded are updated and then the quantity demanded is divided by the demand interval to obtain the average demand, which is used to forecast the average demand; if demand does not occur the original forecast is maintained and only the demand interval from the last time demand occurred to the present time is updated.

We let G represents the demand interval ES value, g represents the interval between current demand and last demand, D represents the demand ES value, x represents the current demand, and Y represents the prediction of the demand. If the demand on the current time step is not 0, then the method's prediction will be:

$$\begin{cases} D_{t+1} = \alpha X_t + (1-\alpha)D_t \\ G_{t+1} = \alpha g + (1-\alpha)G_t \\ Y_{t+1} = \frac{D_{t+1}}{G_{t+1}} \end{cases} \tag{2.15}$$

where α denotes the ratio of current and past data. If no demand has occurred, then it is the data from the previous point in time that is maintained, like:

$$\begin{cases} D_{t+1} = D_t \\ G_{t+1} = G_t \\ Y_{t+1} = Y_t \end{cases} \tag{2.16}$$

We can see that, compared to the normal ES, the Croston Method adds the interval between demands as information to the forecast. In practical terms, a demand may be higher after a long interval, and the opposite may be true for smaller intervals. So in practice, the Croston Method has also been used with better results. Improvements have also been proposed based on this method, for example Teunter proposes to perform state updates even when no demand occurs, in order to improve the robustness of the model [75, 80]. But like ES, the Croston Method still cannot decide how much the long time data actually plays a role in the later predictions, and therefore still has no advantage when dealing with long time series.

2.2.2 N-BEATS Model

N-BEATS is a network specifically for time series published by Element AI in 2020[65]. The fundamental concept behind N-BEATS is to decompose a time series through multiple full connected layers, each layer fitting a portion of the time series information. This partial information is derived from the residuals between the real input series and the estimated input series from the previous layer. This idea is very similar to that of GBDT, where the measured residual between the estimated value and the real input is the information left behind by the previous layer. In this way the next layer of the network can be trained according to the deviations of the previous network. Therefore, we could also regard N-BEATS as a way of ensemble learning implemented in deep learning.

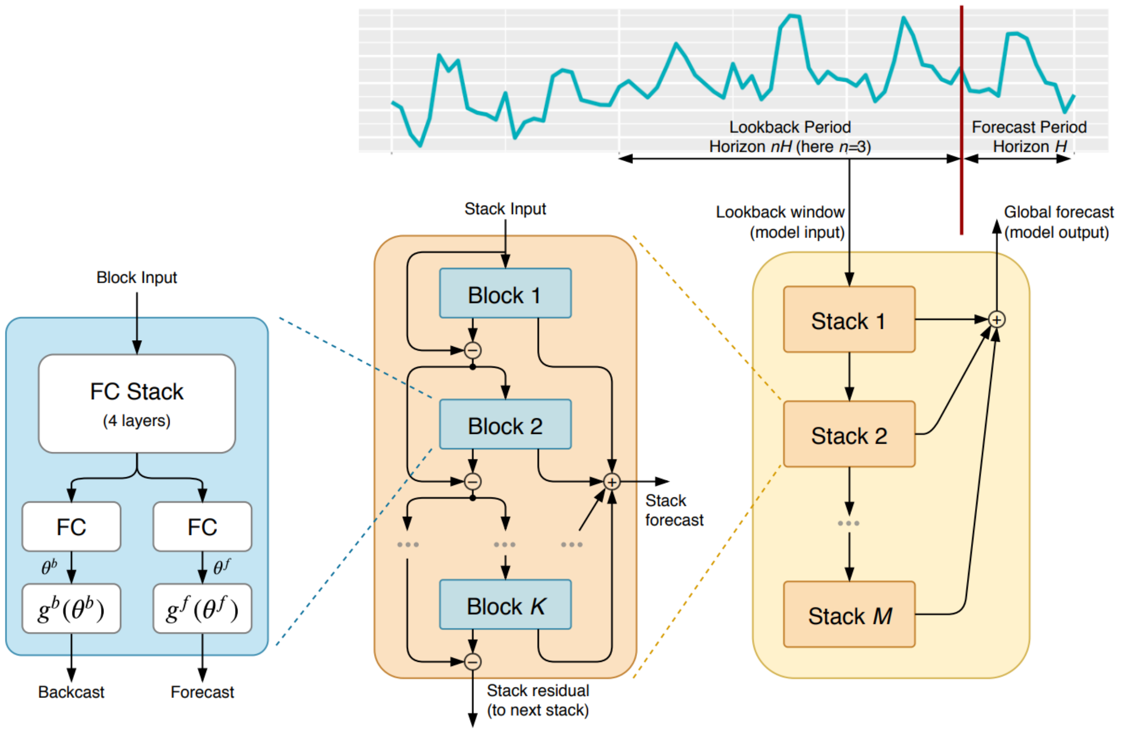


Figure 2.10: Structure of the N-BEATS model[65]

The structure of the N-BEATS is shown as Figure 2.10. Each block is the basic structure of N-BEATS and consists of several fully connected layers. Four of the fully connected layers act as feature extractors for the time series and are then placed into two different predictor. One predictor is responsible for fitting the extracted information to the input series, while the other is responsible for using the extracted information to predict future series. The residuals are used as input to the next block, and the predictions from each layer are summed to form the final prediction.

The success of N-BEATS is that, as stated in the paper, ensemble learning is a much more powerful regularization technique. The use of residuals as input to the weak learner preserves the original information and reinforces the missing information. On the other hand, ensemble's approach enriches the diversity of models and improves robustness. Since it was proposed, it has achieved many good results in several time series competitions[56, 57, 9]. Although N-BEATS has more parameters compared to LSTM, CNN etc., it is still able to complete inference with a very fast speed because all the basic structure are fully connected layers.

In the intermittent demand competition M5 Forecasting[58], there are a lot of academic who use the N-BEATS model. However, because it is not aimed at intermittent demand, it does not achieve top performance. However, some of these researchers have achieved good results with improved models based on N-BEATS, such as the model presented by Anderera et al[7]. In their model, the fully connected layer in the block is replaced with LightGBM and achieves very good results.

2.3 Electricity Consumption Estimation

Electricity consumption forecasting is an area of research that has been of interest for a long time. Proper forecasting of electricity demand can be particularly useful for large power supply corporations or nations in controlling energy distribution and reducing resource use[47, 64, 11, 68, 51, 5]. For households, electricity consumption estimation can help them save money. And with the spread of clean domestic energy sources such as domestic solar power systems, electricity consumption estimation can help to adjust the load between different systems[13, 8, 25, 43, 8, 1, 29]. And we could say that, electricity consumption estimation is a typical time series forecasting problem, which the consumption changes along with the time. So, many scholars' research revolved around the various regression models we have discussed above.

From a macro perspective, we focus on large electricity supply systems, and Vincenzo et al. take the annual electricity consumption of the whole of Italy as the subject of their study, using data from 1970 to 2007 as training data and using the simplest linear regression model to predict electricity consumption up to 2030[11]. The authors also used Gross Domestic Product (GDP) and population as features in their predictions. However, for their study, the change in electricity consumption on an annual basis is a stable and continuous growth model, so a simple model would achieve their goal. Srinivasa et al. go a further step by looking at monthly electricity consumption in India. Although there is generally a continuous increase in electricity consumption compared to annual consumption, the data is cyclical due to seasonal influences. They therefore also propose a more complex Multiplicative Seasonal Autoregressive Integrated Moving Average (MSARIMA) model for their study[68]. The study by Engle et al. published in 1989 combined short-term (monthly) and long-term (annual) data and achieved better results than a single model by using two different models in conjunction with each other[27]. The study by Abderrezak et al. focused on daily data and used a multiple mixture model for prediction. They used the morning and evening peaks of electricity consumption for the day as features for both data prediction and a clustering approach to determine the pattern of electricity usage for the day as another feature[51]. Identifying the most recent electricity usage patterns through an advance classification method is a very effective piece of information. Better results were achieved compared to a single model.

On the other hand, focusing on household electricity consumption, Ivana et al. used a deep residual network to predict daily electricity consumption by combining both short-term and long-term data like Engle's approach[43]. In addition, they used regional electricity consumption and the electricity consumption of other households in the region as supplementary information for the estimation of electricity consumption of individual households, which enabled the model to achieve good results. The study of Candanedo et al., targeting the electricity consumption of appliances, but was the sum of all appliances in a household[13]. However, the novelty of their study is that they used a number of environmental variables, such as temperature and humidity in a certain area of the house, to participate in the prediction. Their study showed that many of the environmental variables were highly correlated with household electricity consumption, and this was the key to their success. The study of Nicoleta et al., on the other hand, is very similar to our project, focusing on the estimation of daily electricity consumption of individual appliances[8]. In the same way as Abderrezak, they predict the probability of whether or not an appliance will be used, and then use this result as features to predict electricity consumption.

Chapter 3

Model Development

In order to find the most suitable model for our objectives, several models were constructed for comparison. The first is a model with an LSTMs as the backbone. By controlling the parameters and the different decoders we were able to construct several variants. The other model is modelled on N-BEATS. First we still use the N-BEATS model, but we also construct an N-BEATS-LSTM model that replaces the fully connected layer with an LSTM. This model was used to investigate whether an ensemble model for computing residuals could achieve better results than a single LSTM based model. In this chapter, we will focus on describing the construction details of the model.

3.1 LSTM-Based Neural Network

According to related research, there is no neural network specifically designed for intermittent demand. However, for the LSTM model, its objects are data with time dependency, so we still chose the LSTM as the backbone network. And through analysis of relevant studies, we have found that trying to get good results using a single model is usually not feasible. Most scholars have used a mixture of models that work in conjunction with each other. Therefore, we have added various networks such as convolution layer and attention mechanism to the LSTM to form a hybrid model. The overall structure of the model is shown in Figure 3.1.

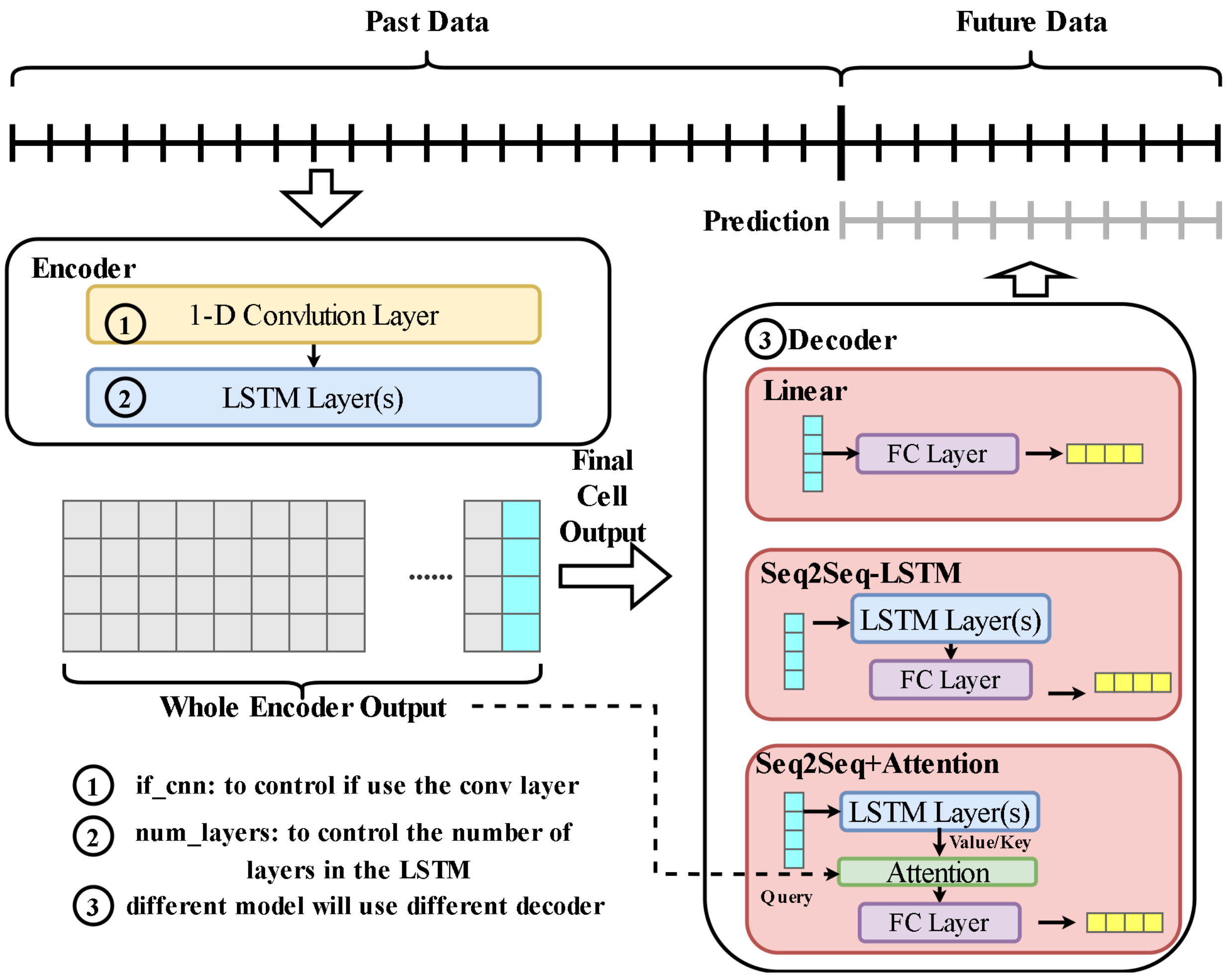


Figure 3.1: Structure of LSTM-Based Model

3.1.1 Encoder

The first part of the model is the encoder. The role of the encoder is to extract the features from the input sequence. The predictive model using the LSTM as the backbone uses the output of the last cell as the summative output by analysing the input sequence in the Encoder. The results of the analysis are then passed on to the next step[73, 2].

Once the data is in the encoder, it first passes through a convolutional neural network. There are many scholars who add a CNN network as an information filter by adding a layer on top of the LSTM network[46, 14, 72, 6, 15]. What a CNN network can bring is a much wider view than a single LSTM. As we said before, the LSTM analyses a time series by combining long-term and short-term memory. But an LSTM cell has as input only the data of the current timestamp, the output of the previous cell and the long-term memory. Because we cannot be sure how much information is actually in long-term memory, the most valid information is only that which is available within a segmented time. The convolutional layer, on the other hand, is able to complement the broader short-term memory, as the convolution kernel computes data with multiple timestamps centred on the current timestamp. So by setting the size of the convolutional kernel, we are able to make the input to the LSTM have information from multiple adjacent timestamps. Therefore, we add a convolutional layer to the top of the encoder, controlled by the parameter "if_cnn" means to be used or not. And at the same time, the size of the convolution kernel becomes a controllable parameter that we need to adjust.

Next comes the LSTM layer. When the convolutional layer has processed the time series it passes the processed time series to the LSTM network. In most models, the LSTM has only one layer, for example in many natural language models [73]. However, there are still many scholars who use multiple LSTMs stacked together to build models, such as in action recognition which works with more complex computer vision networks[81, 78, 82, 16]. So we can see that the multilayer LSTM is for more complex tasks in order to allow the encoder to capture more information. And for our task, the intermittent demand problem is a more complex problem than the normal time series problem. But as defined by the intermittent demand problem, the data in the intervals are mostly ineffective data. So it is more complex in terms of the task, but simpler in terms of the data. Thus, in our network, we set the parameter "num_layers" to control the number of layers of the LSTM. And it is considered as an adjustable parameter in the experiment to determine whether we need a multi-layer LSTM.

3.1.2 Decoder

Next, the encoder passes the output of the last LSTM cell to the decoder. Decoder processes the information from the encoder through a new network and produces the final prediction. A total of three different decoders, linear, seq2seq and seq2seq with attention, have been designed according different studies.

Linear

Linear decoder will only use full connected layer to map the output of the encoder into the final result. Using the fully-connected layer to get the results directly is the last step in most neural networks, and in time series prediction it is also. Decoder receives a matrix of size $[batch\_size, num\_features]$, and the fully-connected layer maps the $num\_features$ extracted features analyzed by encoder into a vector of size $out\_steps * out\_dims$. The mapped result is then reshaped into a matrix of shape $[batch\_size, out\_steps, out\_dims]$, which gives us our prediction results.

The problem with using this approach is that we can see that when decoding the encoder's information, the model simultaneously gets the results for each future timestamp. In this case the temporal characteristics of the output result are lost. This is because, as a time series, we should take the time before and after relationship into account in the prediction.

Seq2Seq

Seq2Seq is a model structure widely used in natural language models, and is used when the length of the output is uncertain, which is usually the case in machine translation tasks[73]. In the case of time series, we can also use this structure, sometimes called an Autoregressive model, even though our prediction series is of a certain length. The structure of the model is shown in the Figure 3.2.

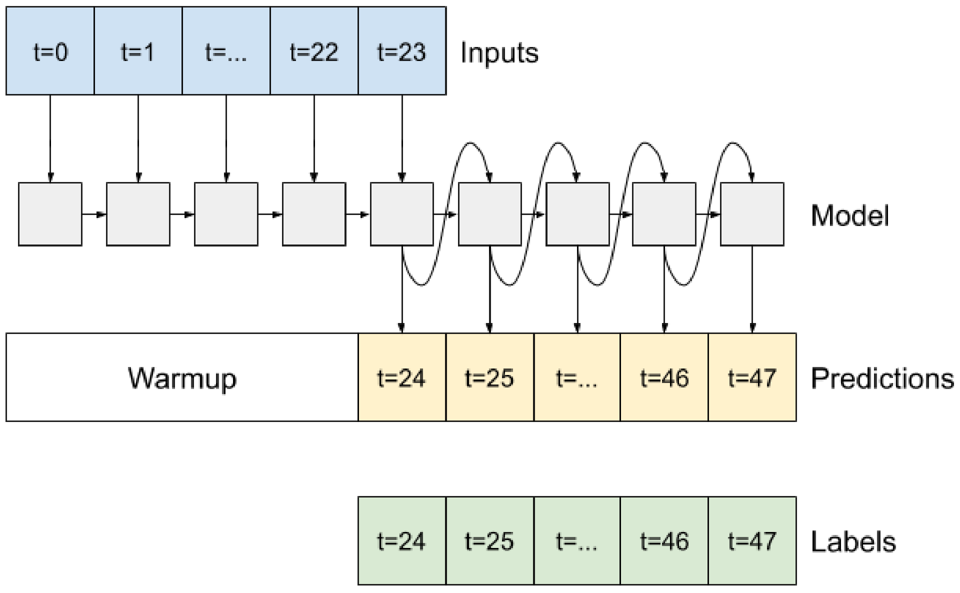


Figure 3.2: Structure of Seq2Seq(Autoregressive) model[2]

In the Seq2Seq model we usually use the same neural network for both decoder and encoder, in our case the LSTM. the first cell of the LSTM in decoder gets its input from the output and the hidden state (long-term memory and hidden state) of encoder. The input to each LSTM cell in deocoder is then the output of the previous cell, thus linking them in time to each other. By doing so, each LSTM cell outputs a feature vector at the prediction timestamp. We then use a fully connected network to map the feature vectors for each timestamp to the actual results.

Seq2Seq + Attention

Following the widespread use of the attention mechanism, many scholars have combined the Seq2Seq model with it and obtained good results. We can see that in the traditional Seq2Seq model, the connection between encoder and decoder is only dependent on the final output. Therefore, the encoder is directly responsible for whether the decoder gets the correct answer or not. Such a fragile design will of course result in a fragile model. The inclusion of an attention mechanism would solve this problem.

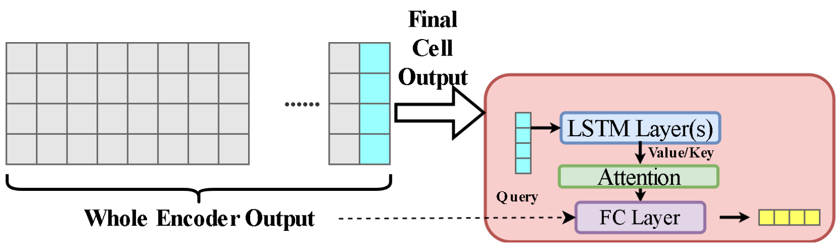


Figure 3.3: Structure of Seq2Seq+Attention Decoder

The decoder we have designed for Seq2Seq+Attention is shown in the Figure 3.3. The most important function of the attention mechanism is that it brings global information to a single output by computing a global attention score. In our model, the Query comes from the complete output of the encoder, while the Value/Key comes from the output of the decoder. According to the formula for the attention score (Formula 2.11), the calculation is equivalent to multiplying the output sequence by the attention score obtained as a weight using the analysis of the input sequence and the output sequence. This enables global information to be added to the output. In our model, because of the attention mechanism there are also various ways of calculating the attention score. Therefore, we chose two different ways of calculating Additive score and Multi-head as a comparison for our experiments.

3.2 N-BEATS-LSTM Model

N-BEATS and its various transformations are able to achieve good results [65, 7]. So we have also included the N-BEATS model in our model comparison. And the N-BEATS transformations we found usually change the base blocks to adapt different tasks. In the post-match statistics of the M5 competition, the organisers also highlighted the advantages of using a combination of models in the time prediction problem[58]. Therefore we also wanted to see if we can achieve our goal by changing the blocks.

3.2.1 Understanding N-BEATS

In an N-BEATS network, the smallest unit of computation is a block, and a single block can be treated as a predictive model. A block has two outputs, one for the prediction and one for the calculation of the residuals. When the data enters a block, it first enters a 4-layer fully connected layer and then enters two different predictors. The two predictors will also contain a fully-connected layer, but in the original paper they will have a smaller width than the previous four layers. This is a one vector mapping process, and this step is also similar to AutoEncoder, where the time series is mapped into a low-dimensional vector to preserve the core information, and then restored back[60, 55, 77]. Doing so facilitates the model to preserve to valid information. Immediately following this fully connected layer is a functional output function. In the original paper, in order to make the output interpretable, the authors used three different output functions, a general function, a trend function and a seasonal function[65]. In our implementation we have chosen the general function as the output function.

Another key factor that makes N-BEATS successful is its integration of a large number of basic networks. In the original paper, the authors used 30 stacks, each containing one block, so that 180 layers of fully connected networks were used in the original network. But the N-BEATS network was trained on a much larger dataset than the data we collected[65, 56, 57, 9]. Therefore, in our implementation, we use only one stack as the model, and change the size of the model by changing the blocks in the stack.

3.2.2 N-BEATS-LSTM Model

According to Anderera et al. they used the N-BEATS network with LightGBM as a block in the intermittent demand competition M5 and achieved good results[7, 58]. So in our implementation we have also chosen to improve the N-BEATS network for comparison.

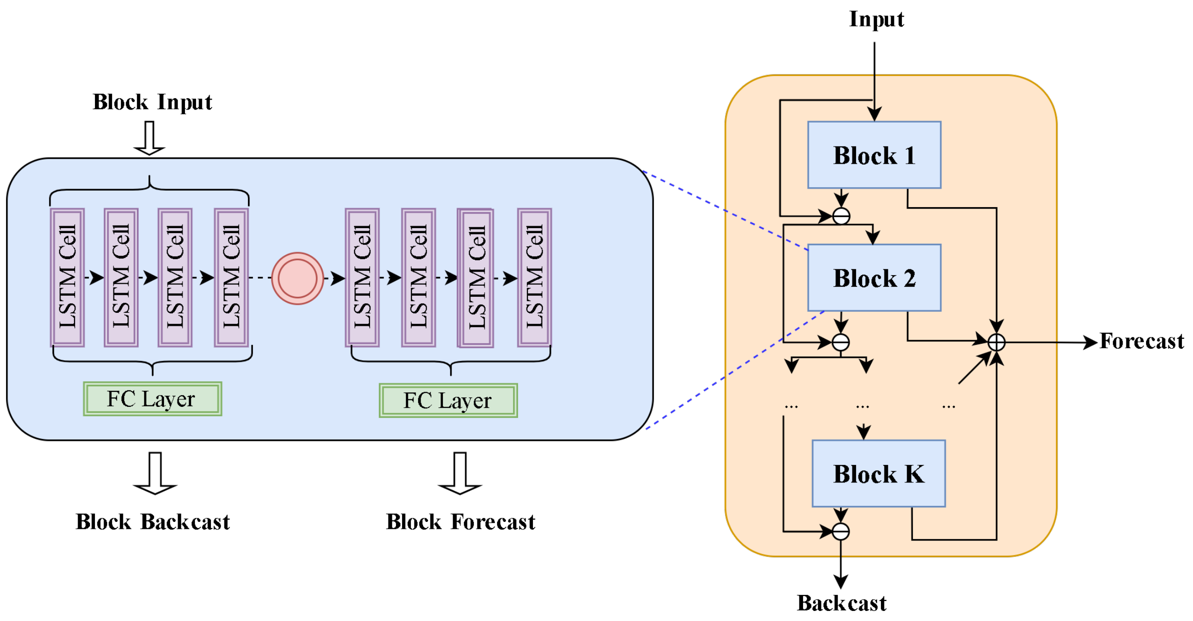


Figure 3.4: Structure of N-BEATS-LSTM model

As shown in the Figure 3.4, we still kept the structure of the N-BEATS network, but used only one stack as the model. In the block we use the LSTM network with Seq2Seq structure as the base network. In the original NBEATS, a block has two different predictors outputting forecast and backcast, which correspond to the outputs of encoder and decoder in the Seq2Seq model. Each cell of the encoder corresponds to a timestamp, so it is straightforward to consider the output of the encoder as a backcast of the input sequence.

For the structure of the LSTM, we have created various models in the previous section. So in the N-BEATS-LSTM we can also choose different models as weak learners. In our experiments we also designed different kinds of N-BEATS-LSTMs for comparison. But according to the characteristics of ensemble learning, the performance of the weak learner should not be too good, otherwise it will lose the purpose of integration and get a worse model instead[32, 70]. So in choosing the basic network of N-BEATS-LSTM, we should also a choose small but good model.

3.3 Summary of Models

Based on what has been said above, we are able to produce many combinations with different models. The table below (Tabel 3.1) shows all the models we have:

Model Name	Details
sl_lstm	Single Layer LSTM
ml_lstm	Multiple Layers LSTM
sl_lstm_s2s	Single Layer LSTM with Seq2Seq Structure
ml_lstm_s2s	Multiple Layers LSTM with Seq2Seq Structure
cnn_sl_lstm_s2s	CNN filter + Single Layer LSTM with Seq2Seq Structure
cnn_ml_lstm_s2s	CNN filter + Multiple Layers LSTM with Seq2Seq Structure
additive_sl	*cnn_sl_lstm_s2s* + Additive Attention
additive_ml	*cnn_ml_lstm_s2s* + Additive Attention
multihead_sl	*cnn_sl_lstm_s2s* + Multi-head Attention
multihead_ml	*cnn_ml_lstm_s2s* + Multi-head Attention
nbeats	General N-BEATS Model
nbeats_lstm	N-BEATS Structure using *cnn_sl_lstm_s2s* as blcok
nbeats_add_lstm	N-BEATS Structure using *additive_sl* as blcok
nbeats_mh_lstm	N-BEATS Structure using *multihead_sl* as blcok

Table 3.1: Models we Created for Comparison

3.4 Loss Function and Metrics Design

In machine learning, the difference between the predicted and true values of a model is usually referred to as the loss; the smaller the loss, the better the model, and the function used to calculate the loss is called the loss function. The loss function is often used as a learning criterion in connection with an optimisation problem, where the model is solved and evaluated by minimising the loss function. On the other hand, Metric methods are the counterpoint of machine learning loss functions and are used to evaluate the actual effectiveness effect of a model, giving us a quantitative perception of how good or bad the current model is.

3.4.1 Loss Function

In section 2.1.1 we mentioned the mean squared error function, the most common loss function in regression problems. The MSE is easy to compute and the calculation of the gradient when updating the parameters is simple. The error $(y - \hat{y})$ grows squarely as seen in Equation (2.3), so a model using MSE will give more weight to the outliers. However, this function is still chosen as the basic loss function in our training. This is because in intermittent demand, a large amount of data is not demanded, so the demanded data become anomalies instead. If we use another loss function that is resistant to anomalies, we lose the detection of demand points instead.

But we did not use MSE as a loss function directly. After examining a large amount of data we found that our data was more specific than the regular intermittent demand data. Each appliance is used less than 3% of the time, which means that the occurrence of zero values in the data takes up almost the entire data. For this feature of a large number of 0 values we designed **SlicedMSE** as a new loss function. Its design principle is shown in the Figure 3.5.

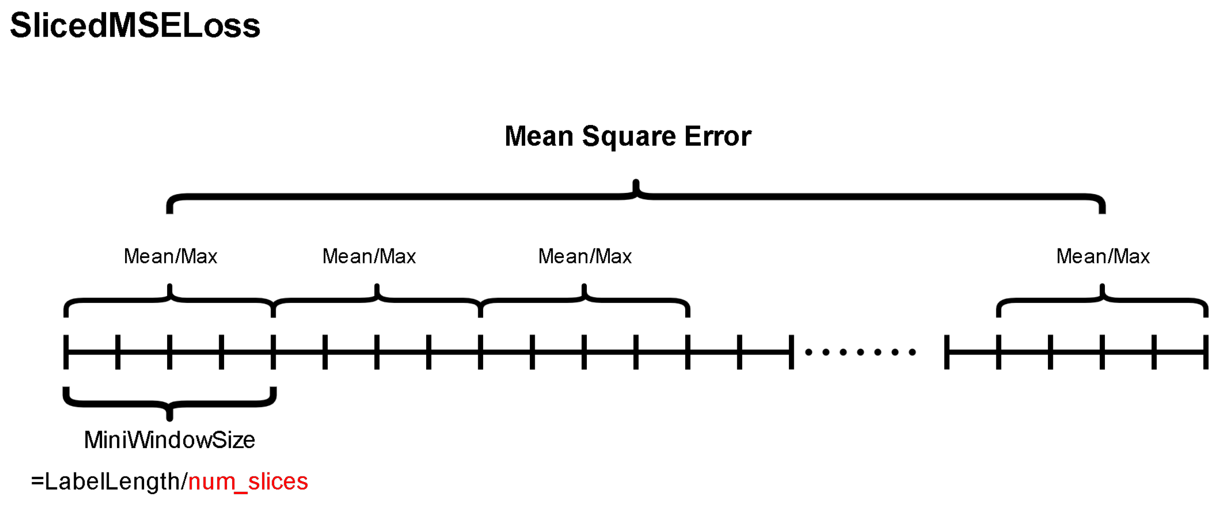


Figure 3.5: Illustration of SlicedMSE

In SlicedMSE we do not calculate in units of a single timestamp, but in units of multiple timestamps (called MiniWindow). The average or maximum value of this MiniWindow is calculated in both predicted and actual values, and then the MSE is calculated on MiniWindows as basic units. By doing so, the MiniWindow calculation simplifies the calculation by allowing a large number of 0-valued data to become a single data point compared to calculating the MSE for each data point.

3.4.2 Metrics

The choice of metric should reflect the goodness of the model. We have a number of options for evaluating regression models, and in our experiments we have chosen Mean Absolute Error(MAE) as the metric. The MAE is calculated as:

$$MAE = \frac{1}{N}\sum_{i}^{N}|Y_i - \hat{Y}_i| \tag{3.1}$$

MAE is a direct measure of the distance between predicted and observed values. Unlike MSE, because MAE is linear in relation to error, no greater weight is given to data points with high error. Thus MAE is a metric that measures the overall performance of the model.

Because our task is very difficult, we have also designed the **SlicedMAE**, by calculating the sum of the values in the MiniWindow to calculate the MAE, which means that we are willing to let the forecast be valid for a certain period of time. Also, for our task, what we actually want to know more than anything else is whether the model can predict the demand when it occurs. So we have also designed **ActiveMAE** to only calculate the MAE at the active state time steps. In this way by pooling the different MAE calculations, we are able to evaluate the performance of the model in a more global way.

Chapter 4

Experiments

In this section, we will first present the data we have collected and talk about how this data is pre-processed. We will then design benchmarks and compare various neural network models in our experiments. We then take the best performing pooled models and train them individually, tuning the parameters. We will then focus on the performance of the models, comparing them with traditional machine learning models and validating our assumptions.

4.1 Preparatory Works

Typically, once we have collected the data we need to analyse it and preprocess it to obtain the data that is more suitable for the model, which called Data Engineering. Data Engineering can make model training easier and, to some extent, improve the performance of the model. In addition, because we collect our own data, we also need to construct reasonable benchmarks to evaluate the performance of the model. Finally, we will introduce our experiment environment.

4.1.1 Datasets

Our data comes from a household where we recorded 8 commonly used appliances over a 2-year period. The electricity consumption of each appliance is collected individually and at a time resolution of 10 min, i.e. each timestamp is the sum of the electricity consumption of the appliance within 10 minutes. Here is a data sample in Figure 4.1.

As we can see from the sample of the data, the data we have collected is clearly intermittent demand data. A large number of appliances are in an inactive state for

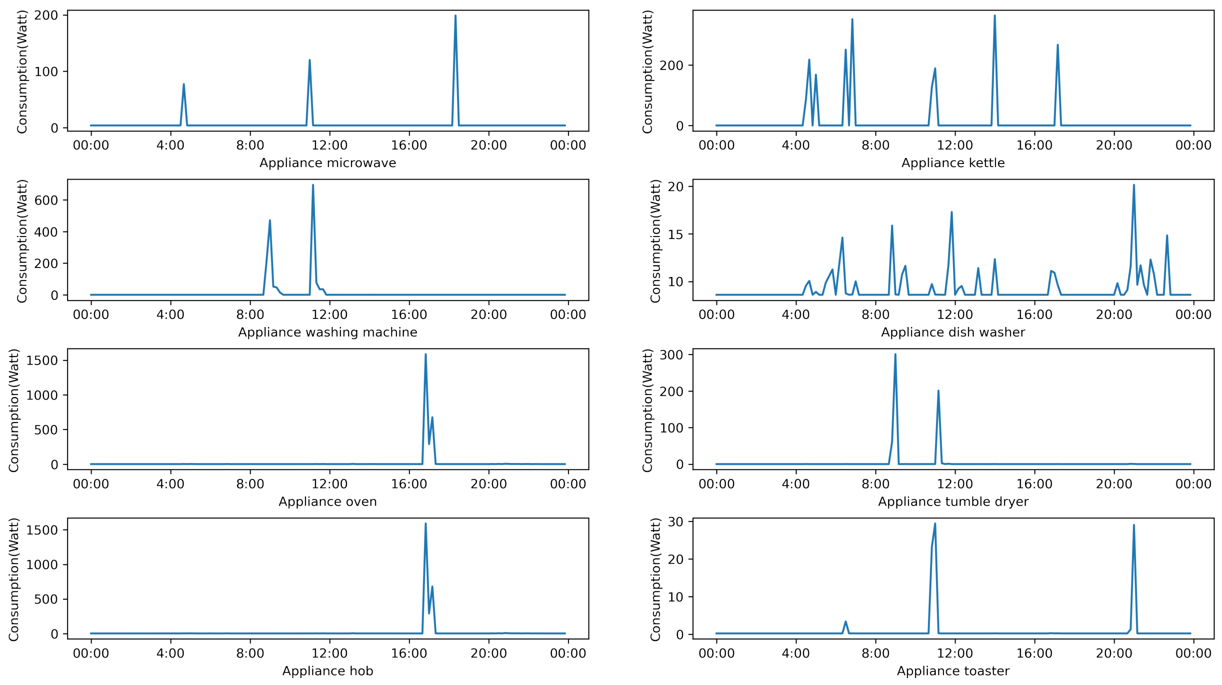


Figure 4.1: Samples of the Data within one day

much of the time. We have therefore counted the data by taking the gap between the active duration and active status of all appliances (as shown in the Table 4.1). According to the table, each appliance is only active for about 10% of the time. Also, the average gap and duration in the data are very uneven. There are some appliances with the longest interval even reaching 2500 timestamps (around 18 days), but the longest duration is only 49 timestamps (around 8 hours). This is a serious data imbalance for our model training and presents a serious obstacle to our training.

4.1.2 Preprocessing

Intermittent Normalization

According to the approach of many mathematical models of intermittent demand, the most important thing in dealing with intermittent demand problems is to eliminate the effects of long intervals, for example by transforming them into continuous problems, treating intervals as valid information, etc.[24]. Thus we propose the method of normalising demand.

First we analyse the distribution of all appliances used during the day (as shown in

	Microwave	**Kettle**	**Washing Machine**	**Dish Washer**
Total Time	116928			
Active Time	4637	10937	8983	11541
Active Percentage	3.96%	9.35%	7.68%	9.87%
Max Gap	757	2532	1167	1365
Avg. Gap	36.14	22.09	19.60	17.21
Max Duration	11	49	15	15
Ayg. Duration	1.47	2.20	1.56	1.79
	Oven	**Tumble Drier**	**Hob**	**Toaster**
Total Time	116928			
Active Time	26981	10534	27573	11164
Active Percentage	23.07%	9.01%	23.58%	9.54%
Max Gap	482	599	482	2532
Avg. Gap	9.89	16.98	9.57	21.40
Max Duration	31	18	31	45
Ayg. Duration	2.70	1.60	2.68	2.18

Table 4.1: Data Gap and Duration Statistics

the Figure 4.2). We can see that almost every appliance has one or more peaks of use, and that the use of appliances is almost always distributed around the peaks. So we then thought that by distributing the demand for one demand over multiple time stamps in close proximity, we would be able to reduce the interval and increase the active time.

The normalisation demand is able to accept a parameter **std**, which represents the timestamp we want to extend before and after the timestamp. We have used a normal distribution to normalise the demand. Using the timestamp t at which the demand occurs as the mean point, a normal distribution is created between $t - std$ and $t + std$ such that the values in this region are summed and equal to the actual demand at timestamp t. The processed data is shown in Figure 4.3.

In this way, we connect the intervals between requirements by spreading them out. In this way we want to transform intermittent demand problems into continuous demand problems as far as possible. In our experiments, we chose to set std to 5, which means that the electricity consumption for one demand timestamp is distributed over 10 adjacent timestamps. This also means that we allow for an error of 2 hours or less in the electricity demand.

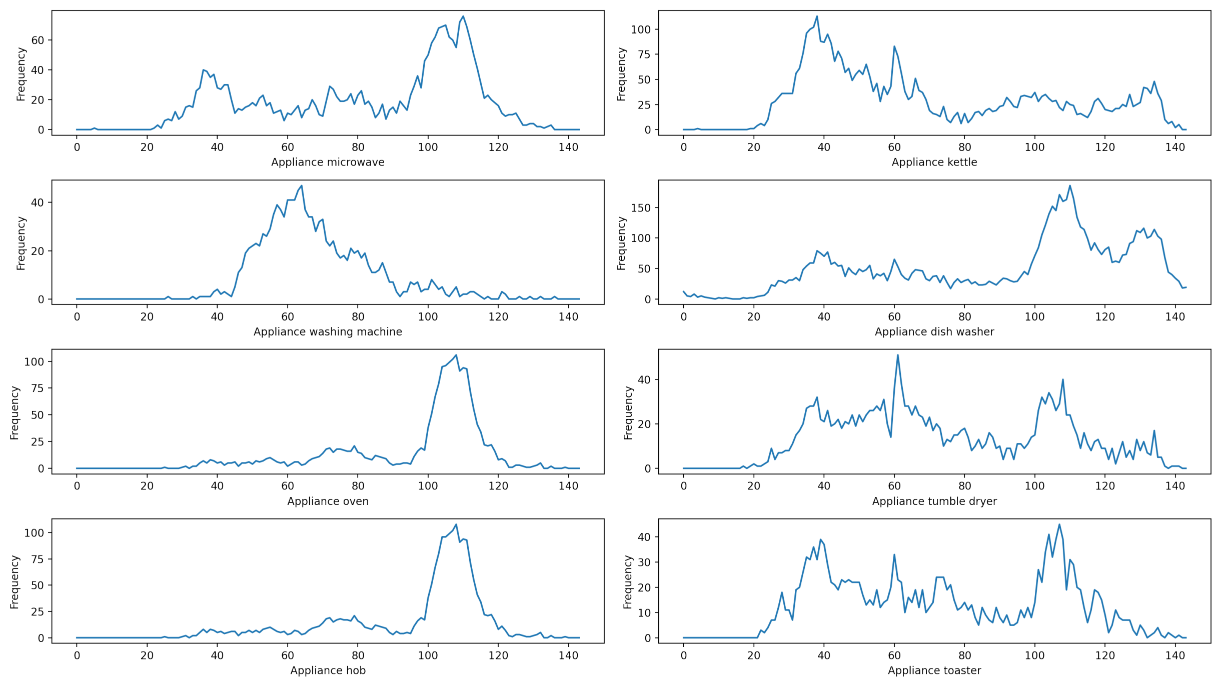


Figure 4.2: Usage Frequency Within One Day

Min-Max Normalization

In practice, the power of various appliances varies, so that the electricity consumption of each appliance is also of different orders of magnitude. Some low-powered appliances, such as the toaster, only use a maximum of 531kWh, while high-powered appliances, such as the oven, use a maximum of 3500kWh. Because our model needs to predict the electricity consumption of each appliance at the same time, when the electricity consumption of each appliance differs significantly, a direct calculation using the raw values would highlight the role of appliances with higher values in the model and relatively weaken the role of appliances with lower levels of values. Therefore, in order to ensure the reliability of the results, the raw indicator data needs to be standardised.

Since the output of the LSTM is made from tanh through the tanh function, a value between -1 and 1 for the output would be convenient for the model to calculate. So we scaled the data using the scaled Min-Max regularisation method, which is calculated as Equation (4.1):

$$x^* = \frac{x - \min}{\max - \min} * scale + shift \tag{4.1}$$

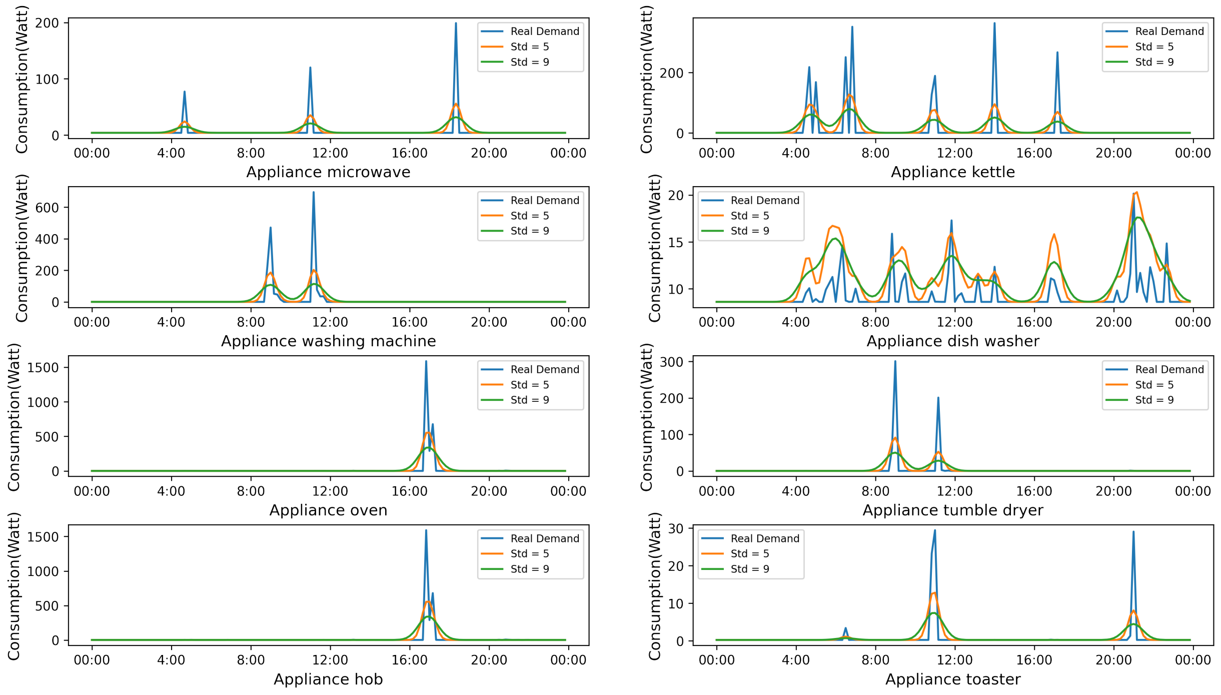


Figure 4.3: Demand Normalization with Different std

The standard Min-Max regularisation $(\frac{x-\min}{\max-\min})$ scales the data to $[0, 1]$ and then multiplies by *scale* change the range to $[0, scale]$. Then add *shift* to get a range of $[shift, scale + shift]$. By choosing $scale = 2, shift = -1$ we have scaled the data to the range [-1,1].

4.1.3 Model Inputs and Outputs

Our objective is a model that, having obtained the electricity consumption of all appliances over a time period, also predicts the respective electricity consumption of these appliances in a future period. As with most time series problems, a sliding window approach is used to generate the inputs and outputs. Sliding window works on our data along with timestamps. There will be a total window size contain the input data with length *input_steps* and the future data with length *output_steps*. *input_steps* plus *output_steps* will equal to the total window size, which means the first *input_steps* data will be the inputs and last *output_steps* data will be the label. And they will not overlap each other. Then the sliding window will scan the whole datasets with a stride values. Every sliding window is a sample of our model. In our experiments we set the input to 144 timestamps and the output to 36 timestamps, i.e. using 24 hours of data to predict the next 6 hours of data. Several sliding windows together will form a batch and the size of a batch is also an important adjustable parameter in the training of neural network models.

4.1.4 Training Set and Validation Set Split

Considering the time span of our mobile phone data and considering that there are seasonal variations. If we directly slice our data into a before-and-after part as a training set and a validation set, it is possible that the appliance usage patterns in the validation set never appear in the training set. Therefore, we adopted a week-based segmentation approach.

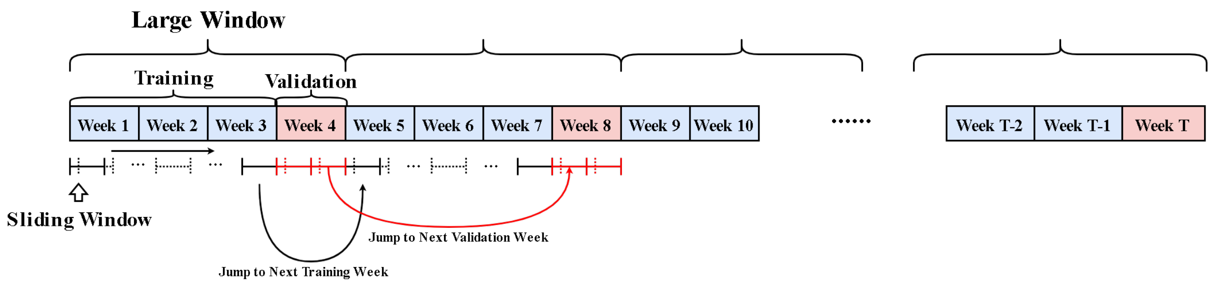


Figure 4.4: Training Validation Split Strategy

As shown in the Figure 4.4. We have a large window of four weeks, with the first three weeks as the training set and the last week as the validation set. The sliding window will only move during the first three weeks when fetching data from the training set and will not move to the validation set. After fetch from this large window, it will skip the validation week and continue fetching in next large window. Even though the week as the validation set will be sandwiched between the two training sets, we don't have to worry about data leakage because each segment is independent. In this way we can obtain an essentially identically distributed training and validation set.

4.1.5 Baseline

Because the data is collected ethically, there is not a ready-made benchmark to use as a comparison, so we need to construct our own benchmark. When introducing the ES method, we said that the most rudimentary method of forecasting is to use the value of the previous data point as a forecast. So our benchmark follows this line of thought. But we cannot use the data from the immediately preceding sliding window as a prediction, because the use of some appliances cannot happen consecutively. So we followed human habits, where people's actions are usually measured in days, and the same thing can happen in the same time period on different days. The data in the sliding window is itself a time period, so we only need to find data from the same time period on another day as a label and become the benchmark. We therefore constructed three different benchmarks, one day old data($baseline\_1$), two days old data($baseline\_2$) and one week old data($baseline\_7$).

4.1.6 Environment

Our model was built using the Tensorflow framework, version 2.8[4]. The model was trained using a remote server with Debian 5.10.120.1 operating system and an Nvidia Gefore T4 GPU. Some of the model testing and evaluation was performed on a mobile computer, model Inspiron 13 7000 2-in-1, operating system Window10 21H2, without GPU acceleration.

4.2 Model Compare

In this section, we compare the various models created above with the baseline. By comparing these models we will use the best performing models for individual training and parameters tuning. We will analyse the metrics we chosen to compare performances of the models.

As the Table 4.2 shows, most of our models were able to do better relative to the benchmark. One of the $multihead\_ml$ almost dominates the $LSTM-based$ models. Compared to the benchmark, the loss value of $multihead\_ml$ dropped by about 32% and the MAE dropped by about 6%. In contrast, the best performer in the $N-BEATS-based$ model is $nbeats\_add\_lstm$, with a drop of about 45% in loss value and 6% in MAE compared to the benchmark. So in terms of MAE, the difference between $multihead\_ml$ and $nbeats\_add\_lstm$ is not that great.

From the table we are also able to draw the following conclusions:

1. **Multi-Layer better than Single Layer** Comparing all the single-layer models with the multi-layer models shows that the multi-layer LSTM is better. The $N-BEATS-based$ model, because it is an ensemble model, also has more LSTM layers to some extent. This also demonstrates the improvement in model performance that comes from increasing the model complexity to some extent. And for the choice of the number of layers we will describe in the next section.

Model	SlicedMSE ↓		MAE ↓	
	Training	Validation	Training	Validation
baseline_1	**0.00643**	0.00696	**0.02231**	**0.02041**
baseline_2	0.00659	0.00675	0.02261	0.02056
baseline_7	0.00671	**0.00632**	0.02275	0.02081
sl_lstm	0.00613	0.00577	0.23187	0.23128
ml_lstm	0.00570	0.00534	0.22944	0.22890
sl_lstm_s2s	0.00421	0.00384	0.02693	0.02600
ml_lstm_s2s	0.00418	0.00380	0.02339	0.02245
cnn_sl_lstm_s2s	0.00529	0.00495	0.04575	0.04494
cnn_ml_lstm_s2s	0.00419	0.00381	0.02506	0.02414
addictive_sl	0.00460	0.00419	0.02588	0.02502
addictive_ml	0.00437	0.00397	0.02311	0.02209
multihead_sl	0.00433	0.00394	0.02180	0.02087
multihead_ml	**0.00417**	**0.00378**	**0.02045**	**0.01948**
nbeats	0.00358	0.00327	0.04331	0.04225
nbeats_lstm	0.00364	0.00330	0.02437	0.02334
nbeats_add_lstm	**0.00343**	0.00327	**0.02036**	**0.01957**
nbeats_mh_lstm	0.00346	**0.00322**	0.02193	0.02078

Table 4.2: Model Compare through Different Metrics

2. **Seq2Seq better than Linear** Compare *sl_lstm* and *ml_lstm* to *sl_lstm_s2s* and *ml_lstm_s2s*. In this comparison, we could say that Seq2Seq model can be better than pure linear model, especially the MAE. As we said before, the pure linear model does not take into account time dependence in the decoder stage. It is therefore also worse than the Seq2Seq model when dealing with multi-step predictions. This is why we have used the Seq2Seq structure in all of our other models.

3. **Better with CNN as filter** Compare *sl_lstm_s2s* and *ml_lstm_s2s* to *cnn_sl_lstm_s2s* and *cnn_ml_lstm_s2s*. When there is a CNN as a filter, as we said before, this gives the LSTM a broader view. And the experimental results verified our hypothesis that the model with CNN filter was superior to the model without CNN filter.

4. **Attention better on MAE** The model with the attention mechanism was not much different in terms of loss values compared to the model without it, but made progress in MAE.

5. **Multihead Attention better than Additive Attention** As stated by Vaswani et al[79]. the introducing of Multi-head gave the model more angular attention scores and increased the variance of the model. So it is clear from the experimental results that Multi-

head is superior to Additive. However, Multi-head increases the accuracy of the model, but this reduces the residuals in the $N-BEATS$ structure, resulting in insufficient input information for the following blocks. Therefore, Additive can do better than Multi-head in $N-BEATS-based$ models.

4.3 Hyper-Parameter Tuning

In this section, we will target *multihead_ml* and *nbeats_add_lstm*, which are representing the $LSTM-based$ models and $N-BEATS-based$ models. A series of experiments will be run for each of the modifiable parameters discussed while developing models to identify the best ones and produce the best model.

4.3.1 Batch Size

As shown in Table 4.3, the choice of batch size did not have a significant impact on the final performance of the model for *multihead_ml*. As for the *nbeats_add_lstm* models, we found in our experiments that they suffer from overfitting, so we increased the batch size and conducted experiments. By taking into account the training time and other factors, we finally chose a batch size of 64 for both $LSTM-based$ models and $N-BEATS-based$ models.

	Batch Size	SlicedMSE ↓		MAE ↓	
		Training	Validation	Training	Validation
multihead_ml	32	**0.00413**	0.00385	0.02407	**0.01675**
	64	**0.00413**	**0.00382**	**0.02405**	0.01748
	96	0.00416	0.00385	0.02426	0.01986
	128	0.00416	0.00385	0.02428	0.01915
nbeats_add_lstm	64	**0.00341**	**0.00328**	**0.02257**	**0.01976**
	128	0.00358	0.00329	0.02334	0.02136
	256	0.00377	0.00346	0.02328	0.02131

Table 4.3: SlicedMSE and MAE w.r.t Batch Size

4.3.2 Learning Rate

When adjusting for the learning rate we found that the $N-BEATS-based$ model would have a smaller learning rate than the $LSTM-based$ model. So we separate them for the study.

For *multihead_ml*, as shown in the Table 4.4, we set five different learning rates and fixed the number of iterations to 140, conducting the experiments while keeping other parameters constant. We can find that the model achieves good performance with learning rates equal to $1e-4$ and $1e-5$. However, by looking at the curves of MAE and SlicedMSE with epoch (Appendix B), we find that the model works well at the beginning for these two values, but after 10 epochs the model over-fits and the MAE increases. In contrast, the model does not overfit in 140 iterations when the learning rates are $5e-6$ and $2e-6$ and $1e-6$, with the best performance being obtained in the fewest number of training sessions when the learning rate is $5e-6$. Therefore, we choose a learning rate of $5e-6$ as the optimal learning rate for $LSTM-based$.

Model	Learning Rate	SlicedMSE ↓		MAE ↓		Epochs
		Training	Validation	Training	Validation	
multihead_ml	1e-4	**0.00381**	**0.00351**	**0.02323**	0.01915	140
	1e-5	0.00409	0.00383	0.02410	**0.01705**	140
	5e-6	**0.00413**	0.00382	**0.02404**	**0.01748**	50
	2e-6	0.00415	0.00388	0.02417	0.01892	70
	1e-6	0.00416	**0.00381**	0.02443	0.01853	140
nbeats_add_lstm	1e-6	0.00350	0.00331	0.02288	0.02045	140
	1e-7	0.00346	**0.00332**	**0.02250**	**0.02019**	140
	1e-8	**0.00345**	0.00326	0.02396	0.02172	140

Table 4.4: SlicedMSE and MAE w.r.t Learning Rate (Model *multihead_ml*)

From the experimental results, it can be seen that 1e-7 is the optimal learning rate for the *nbeats_add_lstm* model. But compared to *multihead_ml*, the loss value is smaller but the MAE is larger. And by looking at the training curve in Appendix B, the loss value and MAE of the *nbeats_add_lstm* model fluctuate downwards within a certain range. Compared to *multihead_ml*, *nbeats_add_lstm* is not a stable model. And for this finding we will also discuss in section 4.4

4.3.3 MiniWindow Size in SlicedMSE

Because changing the size of the Mini-window changes the size of the loss value, we will focus on the MAE. Firstly, we need to be clear that using the Mini-window approach to calculate the MSE means that we will allow for error, as the error changes from a single timestamp to a period of time. Then, as can be seen from the Table4.5, the MAE of the model is reduced by using the SlicedMSE method, whether it is *multihead_ml* or *nbeats_add_lstm*. And enlarging the size of the Mini-window also brings some performance gains. However, since we have already allowed for a 2h error when we used the method of demand normalisation, and if the Mini-window keeps increasing it means that the error range is larger. So we ended up setting the size of the Mini-window to 6.

Model	Mini-window Size	MAE ↓	
		Training	Validation
multihead_ml	1 (Pure MSE)	0.02419	0.01877
	3	**0.02405**	0.01759
	4	0.02407	0.01826
	6	**0.02405**	**0.01748**
nbeats_add_lstm	1 (Pure MSE)	0.02101	0.02130
	3	0.02145	0.02089
	4	0.02119	0.02043
	6	**0.02036**	**0.01957**

Table 4.5: MAE w.r.t Mini-window Size

4.3.4 Number of Layers in Multilayer Models

For the number of layers in the LSTM, we can already see in the model comparison that a multi-layer LSTM is better than a single layer. After increasing the number of layers we find that the optimal number of layers is 3. After changing the number of layers to 4, the loss value does not change, but the MAE increases, which we believe is due to a small overfitting. On the other hand, since we use a single-layer LSTM network as a block in the $N-BEATS-based$ model, we do not need to adjust the number of layers.

Layers	SlicedMSE ↓		MAE ↓		#PARAMS
	Training	Validation	Training	Validation	
2	0.00415	0.00395	0.02422	0.01980	7,546
3	**0.00414**	**0.00382**	**0.02405**	**0.01787**	9,658
4	0.00414	0.00383	0.02417	0.01800	11,770

Table 4.6: SlicedMSE and MAE w.r.t Number of Layers in LSTM (Model *multihead_ml*)

4.3.5 Number of LSTM units

As the Table 4.7 shows, the performance of the model improves as the number of units in the LSTM increases. However the loss values did not improve with units=32 compared to units=16. Although the MAE is reduced a little, the parameters of the model are four times larger. So we decided to set units to 16. And we use the same units number in $N - BEATS - based$ model.

Units	SlicedMSE ↓		MAE ↓		#PARAMS
	Training	Validation	Training	Validation	
8	0.00586	0.00498	0.04311	0.03610	2,794
12	0.00415	0.00384	0.02412	0.01847	5,714
16	0.00413	0.00382	0.02404	0.01753	9,658
32	0.00413	0.00384	0.02391	0.01699	35,674

Table 4.7: SlicedMSE and MAE w.r.t Number of Units in LSTM (Model *multihead_ml*)

4.4 Analysis of *nbeats_add_lstm*

Because the $N - BEATS - based$ model is a new model in our experiments. and we found it to have good performance in our experiments. In this section, we compare the *multihead_ml* and *nbeats_add_lstm* models, both of which are the best performing models in our experiments, and discuss the practicality of both using the actual prediction results and real data.

4.4.1 Compare of *multihead_ml* and *nbeats_add_lstm*

We expand the metrics of the two comparisons by introducing the SlicedMAE and ActiveMAE that we introduced earlier.The SlicedMAE is designed to measure the absolute error between the estimated sum and the actual sum of electricity consumption over a time period, with SlicedMAE-6 being the sum of 6 timestamps and SlicedMAE-9 being the sum of 9 timestamps. AcitveMAE, on the other hand, calculates the absolute error at the timestamps of demand occurrence only. Both of these new metrics better reflect the higher weight we give to demand, as this represents what we really want to get out of the forecast.

As shown in table 4.8, *nbeats_add_lstm* exceeds *multihead_ml* in every judging method when calculating the actual demand. And we can see why this is the case from the actual

Model	SlicedMAE-6		SlicedMAE-9E		ActiveMAE	
multihead_ml	0.12018	0.11909	0.17533	0.17370	0.03326	0.02714
nbeats_add_lstm	0.11660	0.11364	0.16584	0.16151	0.03013	0.02439
Change Ratio	↓ 2.9%	↓ 4.5%	↓ 5.4%	↓ 7.0%	↓ 9.4%	↓ 10.1%

Table 4.8: Compare of *multihead_ml* and *nbeats_add_lstm*

forecast results.

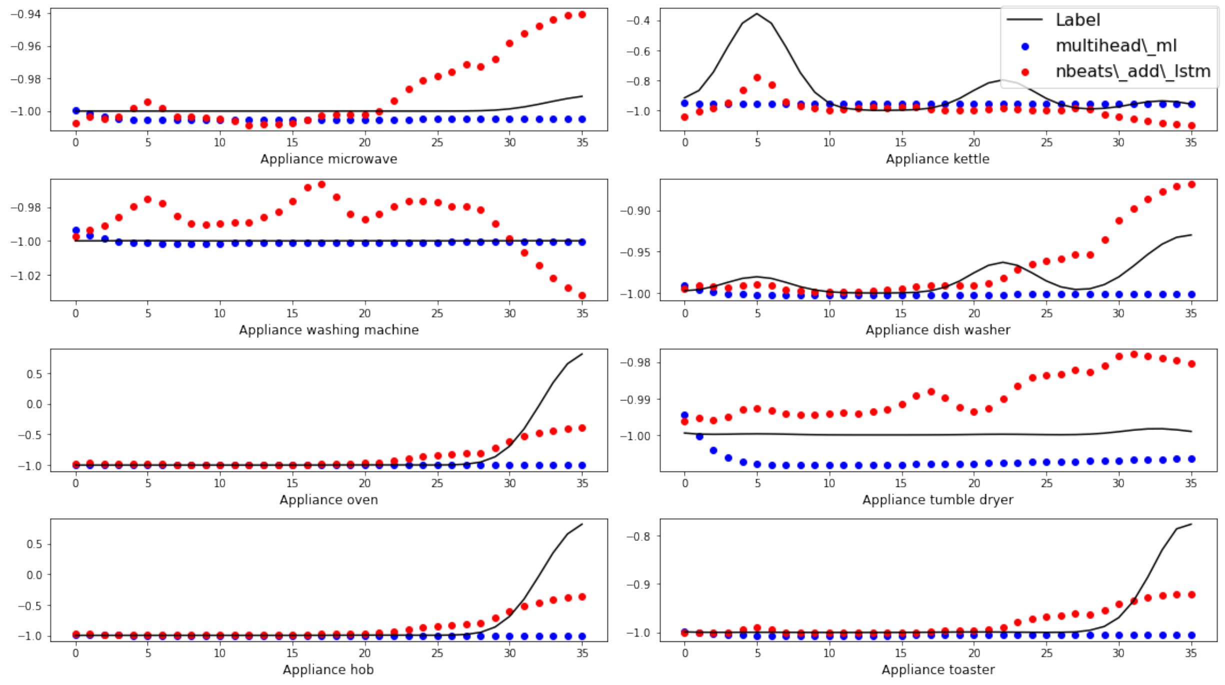


Figure 4.5: Prediction of *multihead_ml* and *nbeats_add_lstm*

It can be seen from the prediction results(Figure 4.5) that *multihead_ml* tends to fit the data in the inactive state more. Even though he carries a lot of information in the first few timestamps, the model gradually weakens it and fits the predicted values to the undemanding state, which is nearly a straight line. The prediction of *nbeats_add_lstm*, on the other hand, will be more volatile than *multihead_ml*, and will fluctuate with the true value. For example, in the prediction results of Appliance Kettle, the pulse around the 5th timestamp is fitted. But with this comes fluctuations in the inactive state as well, for example still in Appliance Kettle's prediction results, the 22-27 timestamp down is predicted to the 30-35 timestamp. This explains why *multihead_ml* performs better than *nbeats_add_lstm* on pure MAE. Since most of the timestamps in our data are inactive, a model like *multihead_ml* that predicts inactive states will be able to achieve better performance on MAE.

4.4.2 Exploring the Success of *nbeats_add_lstm*

The *nbeats_add_lstm* prediction results are divided into two parts, forecast and backcast. We can see from Figure 4.6 that the backcast part will be a better fit than the prediction. This suggests that the missing information is also better obtained when the residuals are calculated internally in the model. This is because the calculation of the residuals is only valid if the results of the backcast are good.

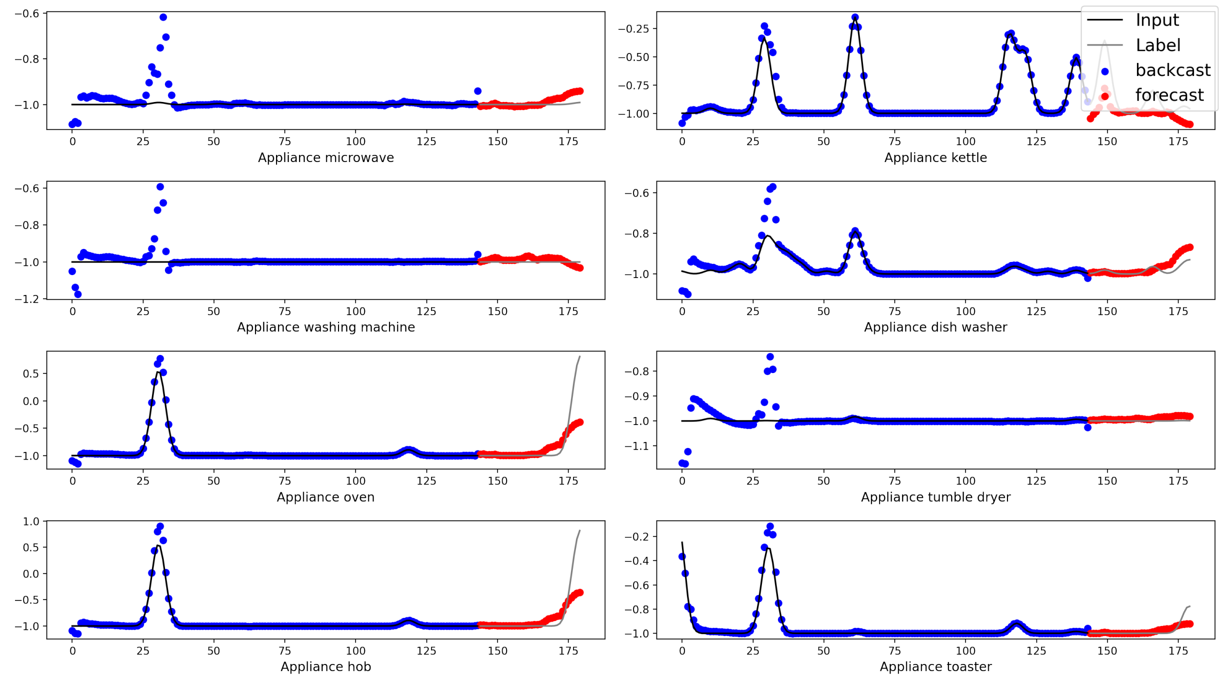


Figure 4.6: Backcast and Forecast of *nbeats_add_lstm* of all Appliances in the Same Period

But it is also the reason for the instability of *nbeats_add_lstm*. We can see from the backcast that the individual appliances are given high correlations with each other within the model. For example, around the 25 timestamp, the microwave, washing machine and tumble dryer should all be inactive, but their backcasts follow the other appliances.

According to Figure 4.7(a) we can see that after the first block gives the wrong backcast, the calculated residuals have a pulse at the 25th timestamp, which also leads to this pulse being remembered in the second block and being successfully fitted to the final backcast. Although it did not have a large impact on the prediction in this one test case, it does illustrate that the performance of *nbeats_add_lstm* is heavily influenced by the output of

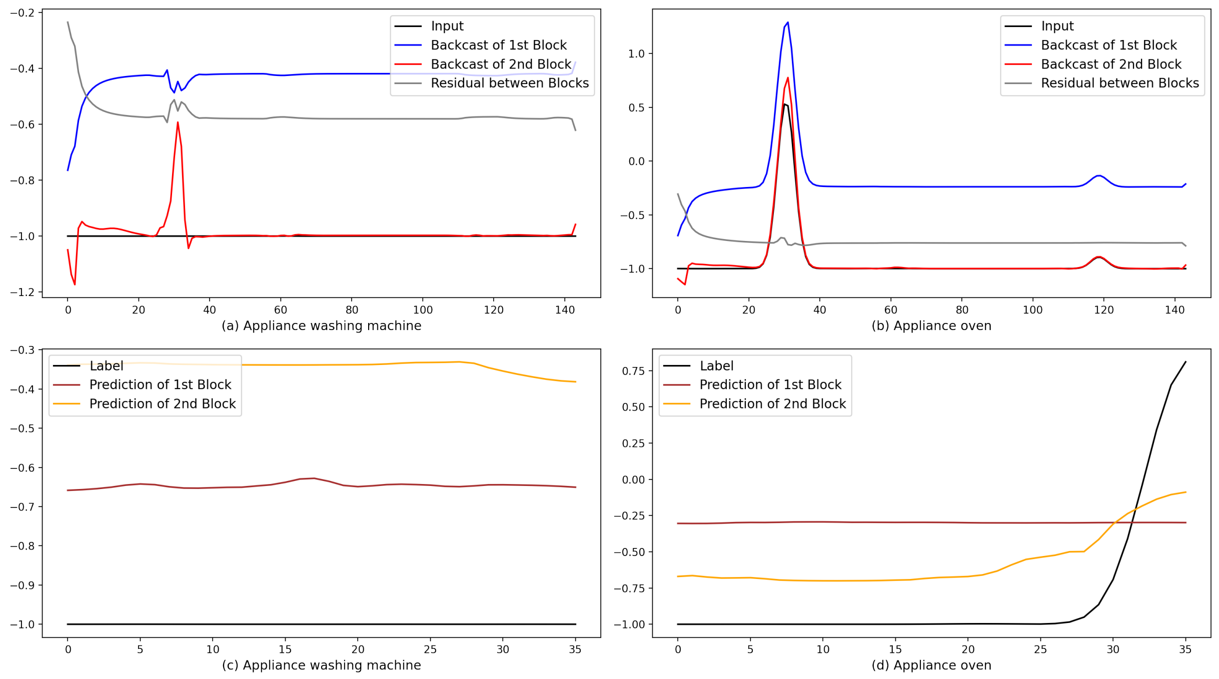


Figure 4.7: Analysis of Backcast and Forecast of Blocks in the Same Period

the first block. And when both blocks are fragile models, the predictions will fluctuate. This is what causes the model to be unstable in the curve of the MAE and loss value.

And from Figure 4.7(d) we can also see the advantages of the residual ensemble model. The brown line shows the prediction of the first block and we can see that it will be a straight line just like *multihead_ml*. The orange line shows the output of the second block, which makes two adjustments to the output of the first block. Firstly, it considers the first prediction to be high, so it shifts the whole thing down. And secondly, in the tail of the prediction it detected the change and have an impulse so its prediction raises up. The two are added together to get the final result. And as to why the second block has an upward trend in the prediction, we guess it's because the residuals are almost a straight line in the calculation, and this is because in the first block we were able to capture the demand in the input. And the fact that the residuals continue to be constant may bring temporal information, implying that this appliance may be coming up to his time period of use, hence the prediction will have the pulse.

Chapter 5

Conclusion

5.1 Summary of Our Works

First we took a closer look at the data we had collected. By examining a lot of relevant literature, we determined that the problem we were dealing with was a special case of a time prediction series, which helped us not to spend more time on unrelated models. In addition we decided, based on many research methods, to use a normalised demand approach to transform the difficult intermittent demand problem into a continuous problem where possible. Although there are still a large number of nulls in our data, by this method we effectively extracted the impulse signal from the demand.

Secondly, in the comparison of time series models, we have drawn several conclusions of interest through a series of experiments. The superiority of the Seq2Seq structure, the CNN filter combined with the RNN and the attention mechanism, is demonstrated.

Third, for intermittent demand data, we propose an ensemble model based on residual computation. This model is based on the N-BEATS network, modified to use the LSTM as a weak learner. It enables N-BEATS, a model widely used for time series forecasting, to target the problem of intermittent demand. We have also analysed the results of the model in depth to understand the role played by the residual calculation in the ensemble model and to learn more fully about the Boosting method.

5.2 Drawbacks

Because of time limitation, we did not look more deeply into the relationship between the complexity of the ensemble model and the performance of the model. We achieved good results using only two blocks, but it was not stable. We should rather ensemble models with more weak learners(more blocks), otherwise we would not be able to define such a model as a valid integration model. So we can only prove that our model is successful with the available data and training parameters, and although there will be fluctuations in performance, it does predict changes in demand.

Using the collected data for training, we do not have a reliable, widely recognised benchmark which we can refer. All our work is based on comparing to the baseline we have created, and although there are three different baselines, we also know that using the last time as a prediction is the simplest forecasting method. And again, because our model uses a multi-step prediction approach, it is also difficult for us to use some of the classical machine learning algorithms as comparisons, and most of the tree-based models we have mentioned can only achieve single-step predictions. So we can only say that our model has had limited success compared to our proposed benchmark and other deep learning models that we have created.

Even though our data resolution is 10min per timestamp, the methods we use, whether SlicedMSE or demand normalisation, have a consensus that the range of permissible errors will be extended. The parameters we have chosen indicate that for errors of estimated sum of consumption within two hour period are allowed . On this basis we cannot fully predict the actual demand, but in most cases we can only predict the pulse of demand. So we are still relatively weak in the face of more precise use.

5.3 Future Work

To address the first drawback, the first experiment we need to perform is to increase the number of blocks to demonstrate the usability of the ensemble. We can do this by increasing the number of blocks and decreasing the number of LSTM units in the blocks in order to achieve a comparison with constant model parameters.

There are various attempts we can make to address the third drawbacks. Firstly, we can explore a loss function for the data imbalance of the regression problem (intermittent demand is also a data imbalance) and achieve better demand forecasting by, for example, giving more weight to demand points. On the other hand we, we can retreat to the next best thing by converting the regression problem into a classification problem, converting demand into demand points and predicting the occurrence or otherwise of demand points. For categorical problems there are good solutions to data imbalance such as FocalLoss[54].

In addition, our models are all very basic, with no hierarchical structure and no multi-feature extraction. We could try more complex models such as graph neural networks(GNN)[20], etc.

Bibliography

[1] Energy prices and their effect on households - office for national statistics.

[2] Time series forecasting — tensorflow core.

[3] Understanding lstm networks – colah's blog.

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

[5] Ahmad, A. S., Hassan, M. Y., Abdullah, M. P., Rahman, H. A., Hussin, F., Abdullah, H., and Saidur, R. A review on applications of ann and svm for building electrical energy consumption forecasting. *Renewable and Sustainable Energy Reviews 33* (5 2014), 102–109.

[6] Alayba, A. M., Palade, V., England, M., and Iqbal, R. A combined cnn and lstm model for arabic sentiment analysis. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 11015 LNCS* (2018), 179–191.

[7] Anderer, M., and Li, F. Hierarchical forecasting with a top-down alignment of independent level forecasts.

[8] Arghira, N., Hawarah, L., Ploix, S., and Jacomino, M. Prediction of appliances energy use in smart homes. *Energy 48* (12 2012), 128–134.

[9] Athanasopoulos, G., Hyndman, R. J., Song, H., and Wu, D. C. The tourism forecasting competition. *International Journal of Forecasting 27* (7 2011), 822–844.

[10] Bahdanau, D., Cho, K., and Bengio, Y. Neural machine translation by jointly learning to align and translate, 2014.

[11] Bianco, V., Manca, O., and Nardini, S. Electricity consumption forecasting in italy using linear regression models. *Energy 34* (9 2009), 1413–1421.

[12] Bishop, C. M., and Nasrabadi, N. M. *Pattern recognition and machine learning*, vol. 4. Springer, 2006.

[13] Candanedo, L. M., Feldheim, V., and Deramaix, D. Data driven prediction models of energy use of appliances in a low-energy house. *Energy and Buildings 140* (4 2017), 81–97.

[14] Cao, J., Li, Z., and Li, J. Financial time series forecasting model based on ceemdan and lstm. *Physica A: Statistical Mechanics and its Applications 519* (4 2019), 127–139.

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

[16] Chen, C., Hua, Z., Zhang, R., Liu, G., and Wen, W. Automated arrhythmia classification based on a combination network of cnn and lstm. *Biomedical Signal Processing and Control 57* (2020), 101819.

[17] Chen, S. T., Kuo, H. I., and Chen, C. C. The relationship between gdp and electricity consumption in 10 asian countries. *Energy Policy 35* (4 2007), 2611–2621.

[18] Chen, T., and Guestrin, C. Xgboost: A scalable tree boosting system. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

[19] Cheng, C. C., and Lee, D. Smart sensors enable smart air conditioning control. *Sensors 2014, Vol. 14, Pages 11179-11203 14* (6 2014), 11179–11203.

[20] Cheng, D., Yang, F., Xiang, S., and Liu, J. Financial time series forecasting with multi-modality graph neural network. *Pattern Recognition 121* (2022), 108218.

[21] Cho, K., van Merriënboer, B., Bahdanau, D., and Bengio, Y. On the properties of neural machine translation: Encoder-decoder approaches. *Proceedings of SSST 2014 - 8th Workshop on Syntax, Semantics and Structure in Statistical Translation* (9 2014), 103–111.

[22] Cortes, C., Vapnik, V., and Saitta, L. Support-vector networks. *Machine Learning 1995 20:3 20* (9 1995), 273–297.

[23] Cowan, J. D. Discussion: Mcculloch-pitts and related neural nets from 1943 to 1989. *Bulletin of Mathematical Biology 1990 52:1 52* (1 1990), 73–97.

[24] Croston, J. D. Forecasting and stock control for intermittent demands. *Operational Research Quarterly (1970-1977) 23* (9 1972), 289.

[25] Dong, F., Yu, J., Quan, W., Xiang, Y., Li, X., and Sun, F. Short-term building cooling load prediction model based on dwdadam-ilstm algorithm: A case study of a commercial building. *Energy and Buildings 272* (10 2022), 112337.

[26] Drucker·, H., Burges, C. J. C., Kaufman, L., Smola··, A., and Vapoik, V. Support vector regression machines.

[27] Engle, R. F., Granger, C. W., and Hallman, J. J. Merging short-and long-run forecasts: An application of seasonal cointegration to monthly electricity sales forecasting. *Journal of Econometrics 40* (1 1989), 45–62.

[28] Fadaeenejad, M., Saberian, A. M., Fadaee, M., Radzi, M. A., Hizam, H., and Abkadir, M. Z. The present and future of smart power grid in developing countries. *Renewable and Sustainable Energy Reviews 29* (1 2014), 828–834.

[29] Fischer, C. Feedback on household electricity consumption: A tool for saving energy? *Energy Efficiency 1* (1 2008), 79–104.

[30] Freund, Y., and Schapire, R. E. A decision-theoretic generalization of on-line learning and an application to boosting. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 904* (1995), 23–37.

[31] Goodfellow, I., Bengio, Y., and Courville, A. *Deep learning.* MIT press, 2016.

[32] Hadjout, D., Torres, J. F., Troncoso, A., Sebaa, A., and Martínez-Álvarez, F. Electricity consumption forecasting based on ensemble deep learning with application to the algerian market. *Energy 243* (3 2022), 123060.

[33] Hastie, T., Rosset, S., Zhu, J., and Zou, H. Multi-class adaboost. *Statistics and Its Interface 2* (2009), 349–360.

[34] Hastie, T., Tibshirani, R., Friedman, J. H., and Friedman, J. H. *The elements of statistical learning: data mining, inference, and prediction*, vol. 2. Springer, 2009.

[35] Ho, T. K. Random decision forests. In *Proceedings of 3rd international conference on document analysis and recognition* (1995), vol. 1, IEEE, pp. 278–282.

[36] Ho, T. K. The random subspace method for constructing decision forests. *IEEE transactions on pattern analysis and machine intelligence 20*, 8 (1998), 832–844.

[37] Hochreiter, S., and Schmidhuber, J. Long short-term memory. *Neural Computation 9* (11 1997), 1735–1780.

[38] Hoerl, A. E., and Kennard, R. W. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics 12* (1970), 55–67.

[39] Hyndman, R., and Athanasopoulos, G. *Forecasting: Principles and Practice.* OTexts, 2014.

[40] Kammen, D. M., and Pacca, S. Assessing the costs of electricity. *http://dx.doi.org/10.1146/annurev.energy.28.050302.105630 29* (10 2004), 301–344.

[41] Kan, X., Reichenberg, L., and Hedenus, F. The impacts of the electricity demand pattern on electricity system cost and the electricity supply mix: A comprehensive modeling analysis for europe. *Energy 235* (11 2021), 121329.

[42] Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., and Liu, T.-Y. Lightgbm: A highly efficient gradient boosting decision tree. *Advances in Neural Information Processing Systems 30* (2017).

[43] Kiprijanovska, I., Stankoski, S., Ilievski, I., Jovanovski, S., Gams, M., and Gjoreski, H. Houseec: Day-ahead household electrical energy consumption forecasting using deep learning. *Energies 13* (5 2020).

[44] Kiranyaz, S., Avci, O., Abdeljaber, O., Ince, T., Gabbouj, M., and Inman, D. J. 1d convolutional neural networks and applications: A survey. *Mechanical Systems and Signal Processing 151* (4 2021), 107398.

[45] Kiranyaz, S., Ince, T., Abdeljaber, O., Avci, O., and Gabbouj, M. 1-d convolutional neural networks for signal processing applications. *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings 2019-May* (5 2019), 8360–8364.

[46] Kong, W., Dong, Z. Y., Jia, Y., Hill, D. J., Xu, Y., and Zhang, Y. Short-term residential load forecasting based on lstm recurrent neural network. *IEEE Transactions on Smart Grid 10* (1 2019), 841–851.

[47] Koprinska, I., Wu, D., and Wang, Z. Convolutional neural networks for energy time series forecasting. *Proceedings of the International Joint Conference on Neural Networks 2018-July* (10 2018).

[48] Kourentzes, N. Intermittent demand forecasts with neural networks. *International Journal of Production Economics 143* (5 2013), 198–206.

[49] Koushik, J. Understanding convolutional neural networks.

[50] Krarti, M., and Aldubyan, M. Review analysis of covid-19 impact on electricity demand for residential buildings. *Renewable and Sustainable Energy Reviews 143* (6 2021), 110888.

[51] Laouafi, A., Mordjaoui, M., Laouafi, F., and Boukelia, T. E. Daily peak electricity demand forecasting based on an adaptive hybrid two-stage methodology. *International Journal of Electrical Power Energy Systems 77* (5 2016), 136–144.

[52] Lecun, Y., Bengio, Y., and 4g332, R. Convolutional networks for images, speech, and time-series.

[53] Lesic, V., Bruin, W. B. D., Davis, M. C., Krishnamurti, T., and Azevedo, I. M. Consumers' perceptions of energy use and energy savings: A literature review. *Environmental Research Letters 13* (3 2018), 033004.

[54] Lin, T.-Y., Goyal, P., Girshick, R., He, K., and Dollár, P. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision* (2017), pp. 2980–2988.

[55] Liou, C. Y., Cheng, W. C., Liou, J. W., and Liou, D. R. Autoencoder for words. *Neurocomputing 139* (9 2014), 84–96.

[56] Makridakis, S., and Hibon, M. The m3-competition: results, conclusions and implications. *International Journal of Forecasting 16* (10 2000), 451–476.

[57] Makridakis, S., Spiliotis, E., and Assimakopoulos, V. The m4 competition: Results, findings, conclusion and way forward. *International Journal of Forecasting 34* (10 2018), 802–808.

[58] Makridakis, S., Spiliotis, E., and Assimakopoulos, V. M5 accuracy competition: Results, findings, and conclusions. *International Journal of Forecasting* (1 2022).

[59] Markandya, A., and Wilkinson, P. Electricity generation and health. *The Lancet 370* (9 2007), 979–990.

[60] Meng, Q., Catchpoole, D., Skillicom, D., and Kennedy, P. J. Relational autoencoder for feature extraction. *Proceedings of the International Joint Conference on Neural Networks 2017-May* (6 2017), 364–371.

[61] Nardelli, P. H., Rubido, N., Wang, C., Baptista, M. S., Pomalaza-Raez, C., Cardieri, P., and Latva-aho, M. Models for the modern power grid. *The European Physical Journal Special Topics 2014 223:12 223* (7 2014), 2423–2437.

[62] Natekin, A., and Knoll, A. Gradient boosting machines, a tutorial. *Frontiers in Neurorobotics 7* (2013), 21.

[63] Nikolopoulos, K. We need to talk about intermittent demand forecasting. *European Journal of Operational Research 291* (6 2021), 549–559.

[64] Nogales, F. J., Contreras, J., Conejo, A. J., and Espínola, R. Forecasting next-day electricity prices by time series models. *IEEE Transactions on Power Systems 17* (5 2002), 342–348.

[65] Oreshkin, B. N., Carpov, D., Chapados, N., and Mila, Y. B. N-beats: Neural basis expansion analysis for interpretable time series forecasting.

[66] Ozkan, H. A., and Aybar, A. A smart air conditioner in smart home. *EEEIC 2016 - International Conference on Environment and Electrical Engineering* (8 2016).

[67] Payne, J. E. A survey of the electricity consumption-growth literature. *Applied Energy 87* (3 2010), 723–731.

[68] Rallapalli, S. R., and Ghosh, S. Forecasting monthly peak demand of electricity in india—a critique. *Energy Policy 45* (6 2012), 516–520.

[69] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. Learning representations by back-propagating errors. *Nature 1986 323:6088 323* (1986), 533–536.

[70] Schapire, R. E. The strength of weak learnability. *Machine Learning 1990 5:2 5* (6 1990), 197–227.

[71] Shalev-Shwartz, S., and Ben-David, S. *Understanding machine learning: From theory to algorithms.* Cambridge university press, 2014.

[72] Siami-Namini, S., Tavakoli, N., and Namin, A. S. A comparison of arima and lstm in forecasting time series. *Proceedings - 17th IEEE International Conference on Machine Learning and Applications, ICMLA 2018* (1 2019), 1394–1401.

[73] Sutskever, I., Vinyals, O., and Le, Q. V. Sequence to sequence learning with neural networks. *Advances in Neural Information Processing Systems 4* (9 2014), 3104–3112.

[74] Syntetos, A. A., and Boylan, J. E. *Intermittent demand forecasting: Context, methods and applications.* John Wiley & Sons, 2021.

[75] Teunter, R. H., Syntetos, A. A., and Babai, M. Z. Intermittent demand: Linking forecasting to inventory obsolescence. *European Journal of Operational Research 214* (11 2011), 606–615.

[76] Tibshirani, R. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological) 58* (1 1996), 267–288.

[77] Tschannen, M., Zurich, E., Google, O. B., Team, B., Lucic, M., and Ai, G. Recent advances in autoencoder-based representation learning.

[78] Ullah, A., Muhammad, K., Ser, J. D., Baik, S. W., and Albuquerque, V. Activity recognition using temporal optical flow convolutional features and multi-layer lstm. *IEEE Transactions on Industrial Electronics* (2018).

[79] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Łukasz Kaiser, and Polosukhin, I. Attention is all you need. *Advances in Neural Information Processing Systems 2017-December* (6 2017), 5999–6009.

[80] Waller, D. Methods for intermittent demand forecasting.

[81] Wang, C., Olugbade, T. A., Mathur, A., De, A. C., Williams, C., Lane, N. D., and Bianchi, N. Chronic-pain protective behavior detection with deep learning.

[82] Zhang, S., Yang, Y., Xiao, J., Liu, X., Yang, Y., Xie, D., and Zhuang, Y. Fusing geometric features for skeleton-based action recognition using multilayer lstm networks. *IEEE Transactions on Multimedia 20* (9 2018), 2330–2343.

Appendix A

Code and Output Data

Github: https://github.com/informetis/uk-ucl-2022

Appendix B

SlicedMSE and MAE Graph

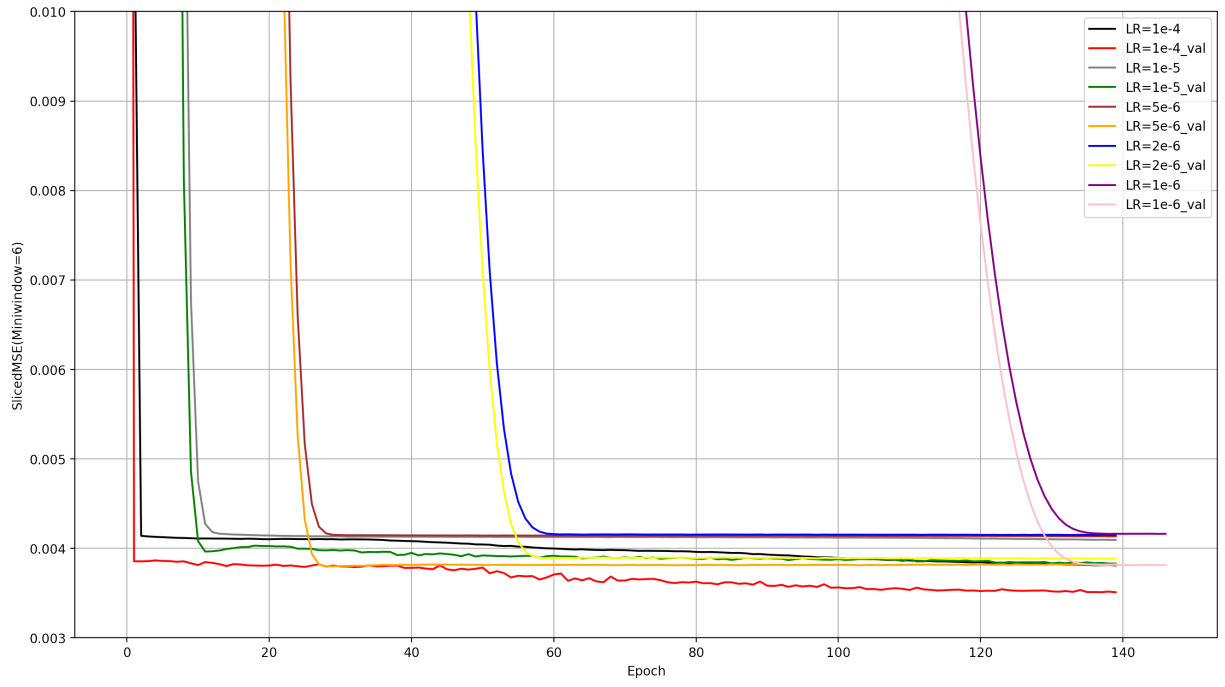


Figure B.1: SlicedMSE graph w.r.t Epochs (LSTM-based Model)

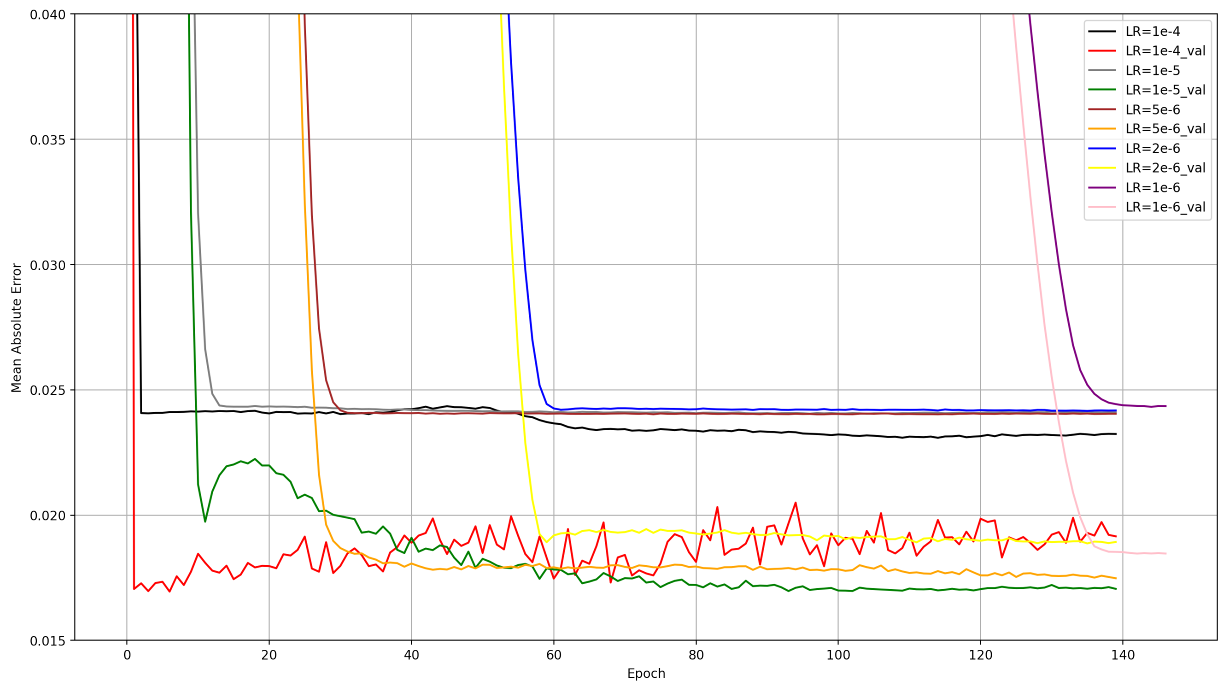


Figure B.2: MAE graph w.r.t Epochs (LSTM-based Model)

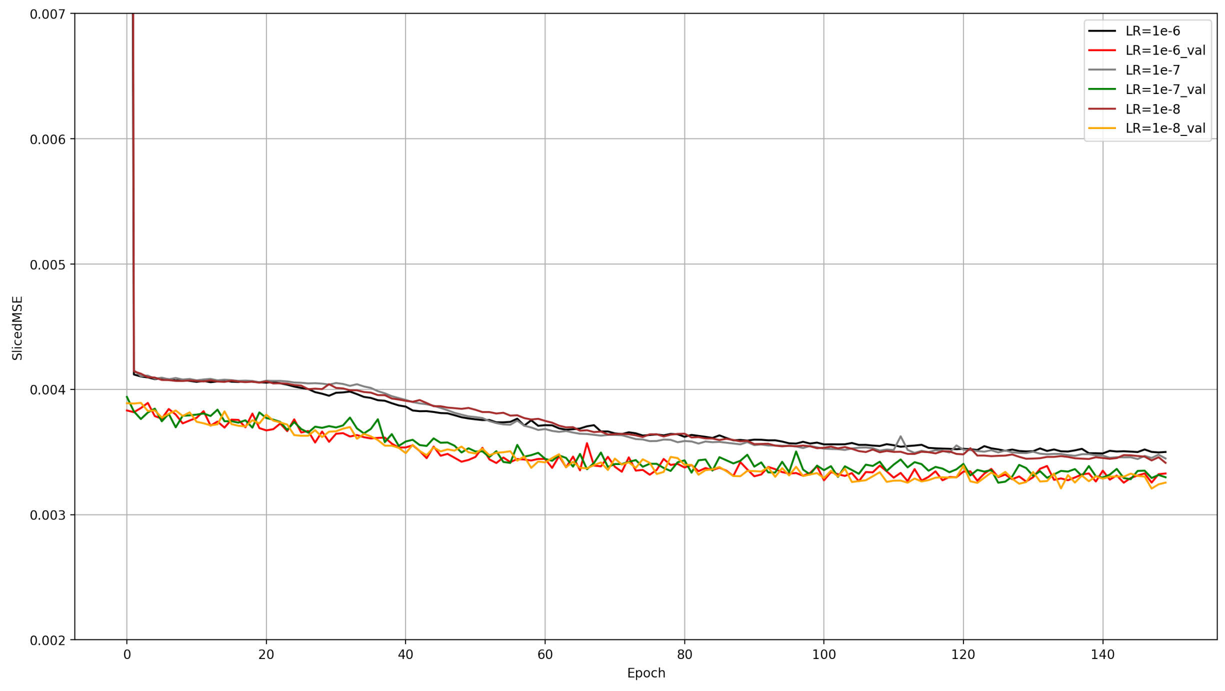


Figure B.3: SlicedMSE graph w.r.t Epochs (N-BEATS-LSTM)

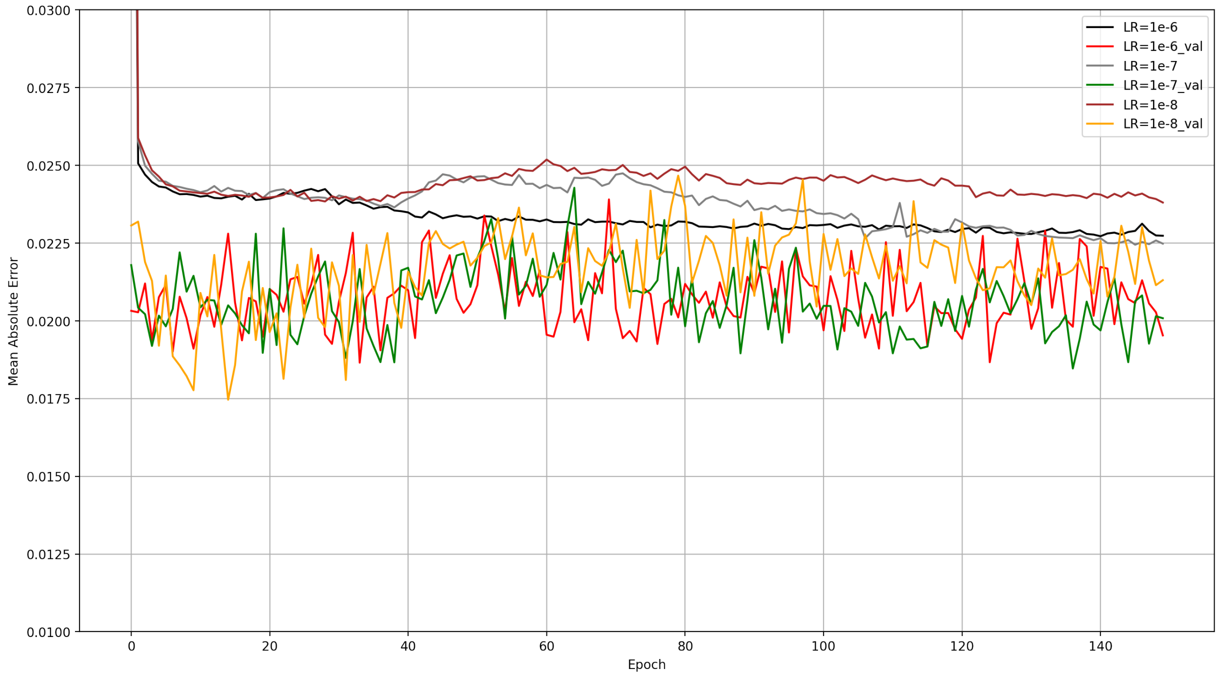


Figure B.4: MAE graph w.r.t Epochs (N-BEATS-LSTM)