CORE
Metadata, citation and similar papers at core.ac.uk
Provided by Scholarship@Western

Western Graduate&PostdoctoralStudies

Western University
Scholarship@Western

Electronic Thesis and Dissertation Repository

8-23-2019 3:00 PM

Forecasting Energy Consumption using Sequence to Sequence Attention models

Ljubisa Sehovac
The University of Western Ontario

Supervisor
Grolinger, Katarina
The University of Western Ontario

Graduate Program in Electrical and Computer Engineering
A thesis submitted in partial fulfillment of the requirements for the degree in Master of Engineering Science
© Ljubisa Sehovac 2019

Follow this and additional works at: https://ir.lib.uwo.ca/etd

Recommended Citation

Sehovac, Ljubisa, "Forecasting Energy Consumption using Sequence to Sequence Attention models" (2019). *Electronic Thesis and Dissertation Repository*. 6544.
https://ir.lib.uwo.ca/etd/6544

This Dissertation/Thesis is brought to you for free and open access by Scholarship@Western. It has been accepted for inclusion in Electronic Thesis and Dissertation Repository by an authorized administrator of Scholarship@Western. For more information, please contact wlswadmin@uwo.ca.

Abstract

To combat negative environmental conditions, reduce operating costs, and identify energy savings opportunities, it is essential to efficiently manage energy consumption. Internet of Things (IoT) devices, including widely-used smart meters, have created possibilities for sensor based energy forecasting. Machine learning algorithms commonly used for energy forecasting, such as FeedForward Neural Networks, are not well-suited for interpreting the time dimensionality of a signal. Consequently, this thesis applies Sequence-to-Sequence (S2S) Recurrent Neural Networks (RNNs) with attention for electrical load forecasting. The S2S and S2S-attention architectures commonly used for neural machine translation are adapted for energy forecasting. An RNN enables capturing time dependencies present in the load data, while the S2S RNN model strengthens consecutive sequence prediction by combining two RNNs: encoder and decoder. Adding the attention mechanism to these S2S RNNs alleviates the burden of connecting the encoder and decoder. Presented experiments compare a regular S2S model and four S2S attention models with two baseline models, the conventional Non-S2S RNN and a Deep Neural Network (DNN). Furthermore, each RNN model was evaluated with three different RNN-cells: Vanilla RNN, Gated Recurrent Unit (GRU) and Long Short-Term Memory (LSTM) cell. All models were trained and tested on one building-level electrical load dataset, with five-minute incremental data. Results showed that the S2S Bahdanau *et al.* attention model was the dominant model as it outperformed all other models for nearly all forecasting lengths.

Keywords: Deep Learning, Energy Load Forecasting, Recurrent Neural Networks, Sequence-to-Sequence, Gated Recurrent Units, Long Short-Term memory, Attention, Bahdanau Attention, Luong Attention

Lay Summary

Energy consumption has been continuously increasing due to the rapid expansion of high-density cities, and growth in the industrial and commercial sectors. To combat negative environmental conditions, reduce operating costs, and identify energy savings opportunities, it is essential to efficiently manage energy consumption. Internet of Things (IoT) devices, such as widely used smart meters, are capable of measuring and communicating data about energy use; thus, they have created opportunities for improved energy management as well as for energy forecasting. Machine learning techniques build a mathematical model based on this historical data in order to make predictions or perform a different task. The common machine learning algorithms used for energy forecasting are not well-suited for time series problems. Consequently, this thesis applies Sequence-to-Sequence (S2S) Recurrent Neural Networks (RNNs) with attention for electrical load forecasting. Specifically, S2S and S2S attention models from neural machine translation are adapted for energy forecasting. RNNs enable capturing time dependencies present in the consumption data, while the S2S RNN strengthens consecutive predictions by combining two RNNs: encoder and decoder. The first RNN (encoder) reads the data, passes information to the second RNN (decoder), and the second one predicts the energy consumption. Further, an attention mechanism was added to the overall model, which helps connect encoder and decoder RNN and allows the decoder RNN to pay attention to specific parts of the encoder RNN. By using these techniques, the proposed approach can improve energy consumption forecasts. Presented experiments compare seven models: a regular S2S model, four S2S attention models, and two baseline models, the conventional RNN and a Deep Neural Network (DNN). All models were trained and tested on one building-level electrical load dataset, with five-minute incremental data. Results showed that the S2S Bahdanau et al. attention model was the dominant model as it outperformed all other models for nearly all forecasting lengths.

Acknowlegements

First and foremost, I would like to express my sincerest gratitude to my supervisor Katarina Grolinger. Without her initial faith in me, none of this would have been possible. She continually offered her support and knowledge throughout these past two years, and always encouraged me to test my abilities, to keep striving for more. Above all, she is a very kind and understanding person, which made the stressful days a lot easier with her in my corner.

This thesis would also not have been possible without the support of my family. My brothers Nemanja, Srdjan, and Aleksandar inspire me every single day and this thesis is dedicated to them. Nemanja, you are my motivation, I just want to make you proud. Srdjan, you are the epitome of a good person, a good man. Aleks, your positivity and humour consistently brings me joy. I love you three more than anyone else in this world. My parents, Gordana and Zoran, thank you for caring so much, I know you two would do anything for us.

To my closest friends, Nechirwan, Dino, Ahmed, Austin, Mat, Dan, Andrew, Dejan, Marko, and Rhyce, thank you for your loyalty, trust, and love. You taught me to never lose sight of true happiness. To my lab collegues and friends, thank you for the generous ingsights, good laughs, and “intellectual” conversations we shared. To Norm, thank you for all your help, it was an honour to pick your brain and learn from you. To my roommate, Linda, thank you for putting up with me during my most stressful moments, you deserve some sort of award for that.

Lastly, I would like to thank Utilismart, the industry partner that I had the pleasure of working with. Thank you for giving me an opportunity to work with an outstanding team, who provided me with valuable experience and influenced me to pursue my dreams.

Contents

Abstract ii

Lay Summary iii

Acknowlegements iv

List of Figures vii

List of Tables x

List of Abbreviations xi

1 Introduction 1

- 1.1 Motivation 1
- 1.2 Contribution 4
- 1.3 Organization of Thesis 6

2 Background 8

- 2.1 Recurrent Neural Networks 8
- 2.2 Sequence to Sequence Recurrent Neural Networks 11
- 2.3 Vanilla RNN, LSTM and GRU Cells 13
- 2.4 Attention Mechanism 17

3 Literature Review 23

- 3.1 Energy Forecasting 23

3.2 Use of Sequence-to-Sequence Models 27
3.2.1 Energy Forecasting 27
3.2.2 Classification Problems 29
3.3 Use of Sequence-to-Sequence Attention Models 30
4 Methodology 33
4.1 Dataset and Evaluation Process 33
4.2 Sample Generation 35
4.3 S2S-prediction 38
4.4 S2S-prediction with Bahdanau Attention 41
4.5 S2S-prediction with Luong Attention 45
5 Evaluation 50
5.1 Experiments 50
5.2 Results and Discussion 55
5.2.1 Case 1: Input One Hour 56
5.2.2 Case 2: Input Four Hours 59
5.2.3 Case 3: Input Ten Hours 62
5.2.4 Case 4: Input Twenty-Four Hours 64
5.2.5 Discussion 68
6 Conclusion and Future Work 73
6.1 Conclusion 73
6.2 Future Work 75
Bibliography 78
Curriculum Vitae 85

List of Figures

2.1 A neural network node computes a nonlinear function of a weighted sum of inputs. 9

2.2 (a) ANN vs (b) DNN. A DNN contains more hidden layers than an ANN. . . . 10

2.3 Sample passed through the Non-S2S RNN. 11

2.4 Sample passed through the S2S RNN. 12

2.5 Vanilla RNN cell architecture [29]. 14

2.6 LSTM cell architecture [33]. 16

2.7 GRU cell architecture [35]. 17

2.8 The original Bahdanau *et al.* attention mechanism [17] for decoder time step t. Here, the attention weights $\alpha_{t,j}$ are shown as being multiplied by the encoder outputs $\vec{h}_j$, then this product is joined with the previous hidden state s_{t-1} to produce the current hidden state s_t, which ultimately leads to the final output y_t. 20

2.9 The original Luong *et al.* attention mechanism [18] for decoder time step t. Here, the current hidden state h_t is used to compute the attention weights α_t by one of three score functions. The context vector c_t is then obtained by the inner product of the attention weights α_t and encoder outputs $\bar{h}_s$. This context vector and the current hidden state are used in producing the attentional hidden state $\tilde{h}_t$, which leads to producing the final output y_t. 21

4.1 Usage data zoomed in to show breakdown between the train and test sets. . . . 34

4.2 Training set sample generation. Input and target samples generated from random index $i + 1$. 36

4.3 Test set sample generation. (a) Input and target samples generated from dataset at index $i' + 1$. (b) Sequential input and target samples generated from sliding window with overlap length N. 37

4.4 Detailed process of the S2S architecture. The left box shows the last two steps of E_T, while the right box shows the first two steps of D_N. 39

4.5 Detailed process of the BA mechanism at time step i. 43

4.6 Detailed process of the LA mechanism at time step i, using the *general* score function. 47

5.1 Comparing cell performance of each non-DNN model: input $T = 12$ to predict $N_i \in \vec{N}$. First row graphs show MAPE, while bottom row shows MAE. 57

5.2 Best achieved results, by each model: input $T = 12$ to predict $N_i \in \vec{N}$. First row graph shows MAPE, bottom row shows MAE. 58

5.3 (a) Results for the S2S-BA model where $T = 12,\ N = 12$. From left to right: Predicted vs. Actual of the last output sample, total train loss and total test loss computed at each epoch, respectively. (b) Results for the S2S-LA-concat model where $T = 12,\ N = 48$. (c) Results for the S2S-LA-dot model where $T = 12,\ N = 120$. (d) Results for the S2S-LA-general model where $T = 12,\ N = 288$. 59

5.4 Comparing cell performance of each non-DNN model: input $T = 48$ to predict $N_i \in \vec{N}$. First row graphs show MAPE, while bottom row shows MAE. 60

5.5 Best achieved results, by each model: input $T = 48$ to predict $N_i \in \vec{N}$. First row graph shows MAPE, bottom row shows MAE. 61

5.6 (a) Results for the S2S-LA-concat model where $T = 48,\ N = 12$. (b) Results for the S2S-BA model where $T = 48,\ N = 48$. (c) Results for the S2S-LA-dot model where $T = 48,\ N = 120$. (d) Results for the S2S-BA model where $T = 48,\ N = 288$. 62

5.7 Comparing cell performance of each non-DNN model: input $T = 120$ to predict $N_i \in \vec{N}$. First row graphs show MAPE, while bottom row shows MAE. . . . 63

5.8 Best achieved results, by each model: input $T = 120$ to predict $N_i \in \vec{N}$. First row graph shows MAPE, bottom row shows MAE. 64

5.9 (a) Results for the S2S-LA-general model where $T = 120,\ N = 12$. (b) Results for the S2S-BA model where $T = 120,\ N = 48$. (c) Results for the S2S-BA model where $T = 120,\ N = 120$. (d) Results for the S2S-LA-dot model where $T = 120,\ N = 288$. 65

5.10 Comparing cell performance of each non-DNN model: input $T = 288$ to predict $N_i \in \vec{N}$. First row graphs show MAPE, while bottom row shows MAE. . . . 66

5.11 Best achieved results, by each model: input $T = 288$ to predict $N_i \in \vec{N}$. First row graph shows MAPE, bottom row shows MAE. 66

5.12 (a) Results for the S2S-BA model where $T = 288,\ N = 12$. (b) Results for the S2S-BA model where $T = 288,\ N = 48$. (c) Results for the S2S-o model where $T = 288,\ N = 120$. (d) Results for the S2S-o model where $T = 288,\ N = 288$. . 67

5.13 Analyzing model performance for varying input lengths T, by (a) MAPE and (b) MAE comparison. 72

List of Tables

4.1 Feature Data (before standardization). 34

5.1 Best achieved MAPE and MAE for each model to predict $N_i \in \vec{N}$. Input length, cell used, and hidden state size not taken into consideration. The respective standard deviation of the APE and AE for those specific test set samples is given in the brackets below. 70

List of Abbreviations, Symbols, and Nomenclature

ANN(s)	Artificial Neural Network(s)
AE	Absolute Error
APE	Absolute Percentage Error
BA	Bahdanau *et al.* Attention
BLEU	Bilingual Evaluation Understudy
BPTT	back-propagation through time
BTU	British Thermal Unit
CCAD-SW	collective contextual anomaly detection using sliding window
CNN(s)	Convolutional Neural Network(s)
CTC	connectionist temporal classification
DA-RNN	dual-stage attention-based RNN
DL	Deep Learning
DNN(s)	Deep Neural Network(s)
D_T	Decoder RNN
EIA	U.S. Energy Information Administration
ELM	Extreme Learning Machine
EMD	empirical mode decomposition
E_T	Encoder RNN
FFNN(s)	FeedForward Neural Network(s)
HI	health index
IESO	Independent Electricity System Operator
IMF(s)	intrinsic mode function(s)
GRU	Gated Recurrent Unit
kW	Kilowatt
LA	Luong *et al.* Attention

LSTM	Long Short-Term Memory
LVCSR	large vocabulary continuous speech recognition
MAE	Mean Absolute Error
MAP	mean average precision
MAPE	Mean Absolute Percentage Error
ML	machine learning
MSE	Mean Squared Error
NAR	Nonlinear Auto Regressive Neural Network
NE	Naive Encoders
NMT	Neural Machine Translation
PDRNN	pooling based deep recurrent neural network
PvA	Predicted versus Actual
RNN(s)	Recurrent Neural Network(s)
RUL	Remaining Useful Life
SD	similar days
SD-AE	standard deviation-AE
SD-APE	standard deviation-APE
SMT	statistical machine translation
STLF	short-term residential load forecasting
SVM	Support Vector Machines
S2S	Sequence-to-Sequence
S2S-BA	S2S with Bahdanau Attention
S2S-LA	S2S with Luong Attention
S2S-LA-concat	S2S-LA using concat score function
S2S-LA-dot	S2S-LA using dot score function
S2S-LA-general	S2S-LA using general score function
S2S-o	S2S one value, regular S2S model

Chapter 1

Introduction

1.1 Motivation

Population growth together with expanding industrial and commercial sectors requires a continuous increase in energy production. It is estimated by the EIA (U.S. Energy Information Administration) that the industrial and commercial sectors consume 50% of the total energy production [1]. In 1949, the energy consumption by the commercial and industrial sectors was 3,668.816 trillion BTU and 14,723.587 trillion BTU, respectively [2]. In 2018, these numbers rose to 18,607.581 trillion BTU and 32,617.767 trillion BTU [2], an increase of over 507% and over 221%, respectively. Energy production and use is the single biggest contributor to global warming, accounting for roughly two-thirds of human-induced greenhouse gas emissions [3]. Furthermore, the International Energy Agency estimates that a push for electric mobility, electric heating, and electricity access could lead to a 90% rise in power demand by 2040 [4]. Hence, efficient energy management in buildings is crucial to combat negative environmental hazards, such as degradation and carbon dioxide emission [5].

In addition, buildings with efficient energy systems present financial gain by reducing overall operating costs. It is common for commercial, industrial, and other large energy consumers to pay premium prices for high energy peaks. For example, in Ontario, the Independent Electricity System Operator (IESO) charges their large customers Global Adjustment fees based on the consumers' contribution to the top five province-wide peaks in a set period [6]. Another category of consumers is charged premiums based on their peak monthly consumption [6]. The higher their peak consumption during this monthly time frame, the greater are the endured costs. Therefore, it is important for respective parties to improve their energy consumption efficiency for a reduction in associated operating costs.

The development of smart meters has provided data that can be used to evaluate building energy consumption patterns. This smart meter data can be used to train machine learning (ML) models to predict the future energy consumption for a desired time frame ahead. If the model produces a predicted load pattern similar to the actual, the interested parties can make cost-effective decisions based on these predicted values. For example, if the model predicts that the consumption will be high during peak hours, energy saving measures can be taken to reduce overall energy cost. A predictive model can also be used for anomaly detection. If the actual usage is not within a desired threshold of the model predictions, these can be deemed as anomalies and interested parties can further analyze their root cause. In addition, a predictive model enables plant operators to optimize scheduling, such as maintenance work, as well as to improve energy conservation. For example, if the model predicts that the consumption will be low for a certain time frame ahead, plant operators can schedule maintenance work at this time since the plant will be in a low-usage state. Furthermore, if the model predicts consumption will be high for a certain time frame ahead, plant operators can limit energy usage, or caution workers, to potentially conserve energy usage. Overall, energy consumption forecasting leads to reduced costs, improved system reliability, and efficient use of energy resources. Thus, this work focuses on ML algorithms to successfully forecast building energy consumption.

Machine learning-based energy forecasting has been attracting significant research and industry attention due to the importance for energy consumers and producers. Whereas forecasting techniques in the past have mainly focused on annual or monthly energy forecasts for a country or region, recent years have shown that the interest has shifted to building-level energy forecasting in hourly, 15 min or smaller intervals. Moreover, the availability of smart meters and other sensors has enabled deeper insights and improved energy forecasting. As machine learning, and specifically deep learning, evolves rapidly [7] [8] [9], energy forecasting can benefit from this cutting edge research.

Feedforward neural networks (FFNN) are commonly used for load forecasting [10]; in FFNN, information moves through a collection of connected nodes from the input layer, through hidden layers, to the output layer. A Deep Neural Network (DNN) is a framework of Deep Learning (DL), a subset of ML, that involves processing data through many hidden layers, usually containing a high number of nodes. A DNN can be thought of as a much larger ANN, while the purpose of both is to learn the mathematical mapping that the input data takes to produce the output data.

A Recurrent Neural Network (RNN) is a form of DNN where the connections between nodes establish a directed graph along a sequence [11]. This architecture allows RNNs to consider the current input along with the previously received inputs, differing from FFNNs which only consider the current input. RNNs pass an internal hidden state through the graph which is used as a mechanism to remember information throughout a temporal sequence. As can be seen from recent works on load forecasting [12] [13] [14], RNNs have an advantage over FFNNs and other approaches in analyzing the temporal dynamic behaviour of a sequence [15]. However, the combination of an encoder and decoder RNN to form a Sequence-to-Sequence (S2S) RNN, has proved even more superior for time series related tasks [16]. The encoder RNN is tasked with encoding information into a fixed-length vector, which the decoder RNN uses to sequentially make predictions [16]. However, these S2S models pose potential issues since the encoder is burdened with compressing all necessary information into this fixed-length vector. For longer input sequences, this makes the task of the encoder increasingly difficult.

This issue was addressed for S2S models used in neural machine translation, which aims to maximize the language translation performance. Both Bahdanau *et al.* [17] and Luong *et al.* [18] applied their respective attention mechanisms to their S2S models. These mechanisms learned to align and translate jointly. Each time the model generated a word in the decoder, it soft searched the encoder outputs to find which positions, in the input sequence, held the most relevant information. The model then predicted a word using this information, along with previously predicted words. The most important feature of attention is that it alleviates the encoder from solely compressing all input information into a fixed-length vector.

These ML techniques have potential for energy forecasting because of the long term temporal dependencies they provide. Energy forecasting is a time-series related problem; RNNs use an internal state to remember information throughout a time series sequence. The S2S models use an encoder-decoder framework to better predict consecutive time steps ahead, while the attention based S2S models improve the Non-S2S architecture; the decoder is assisted at each prediction step with information from the encoder outputs. The S2S attention models showed improved state-of-the-art performance, over regular S2S models, for the time series task of neural machine translation [17] [18]. Therefore, these attention models hold potential to improve accuracy of regular S2S models in other time series prediction tasks such as energy forecasting.

1.2 Contribution

The primary objective of the presented work is to enhance energy forecasting methods through use of a novel adaptation of the S2S attention architectures. The use of a S2S attention model for energy consumption forecasting is novel to this domain. Previous S2S attention architectures have mainly been used in the classification problem domain of neural machine translation (NMT). Furthermore, the contributions that this thesis presents are:

- Input and output sample generation for the energy forecasting problem. This data preparation was required to facilitate use of S2S models from NMT in energy forecasting. For the training set the samples were uniformly randomized, for the test set the samples were generated using a sliding window technique [19]. The combination of randomized samples for the training set and sliding window samples for the test is the specific contribution.

- Adaptation of a S2S model for the energy forecasting domain. S2S models have proved to be powerful models for the task of language translation, which is a classification time series problem. Hence, applying S2S and comparing with other models for the task of load forecasting is the specific contribution.

- Adaptation of attention based architectures used in NMT problem domains for energy forecasting. Four different attention mechanisms were applied to a S2S model, and the application and comparisons of these models for load forecasting is the specific contribution.

- Evaluation of seven different models: Non-S2S RNN, regular S2S, four S2S-attention models, and a DNN model. The comparison of two strong baseline models (Non-S2S RNN and DNN) with the S2S and four S2S attention models is the specific contribution.

- Evaluation of three different RNN cells: Vanilla RNN, Gated Recurrent Unit (GRU), and Long Short-Term Memory (LSTM).

- Evaluation of each model across four prediction lengths: input length is fixed as prediction length varies. The specific contribution here is to see which model performs the best for a fixed number of input steps.

- Evaluation of each model across four input lengths: input length varies as prediction length is fixed. The specific contribution here is to see if increasing the number of input steps also increases the accuracy of the models.

The attention models were compared against a Non-S2S RNN model, a regular S2S models, and a DNN model. In total, four different attention mechanisms were used in the evaluation process. This was done to evaluate more than one attention mechanism versus the regular S2S model, as well as compare the attention mechanisms among themselves to see which one performs the best. The Non-S2S and DNN models were used to serve as baseline comparison models. All non-DNN models were evaluated with each of the following cells: Vanilla RNN, GRU, and LSTM.

In addition, three DNN models of different sizes were used in the evaluation process: small, medium, and large. Three sizes were chosen to provide fair comparisons with the S2S-attention based models and the regular S2S model. Moreover, all models were evaluated for four different input cases: input length was fixed at four different values and at each fixed value the prediction length was varied. This was done to analyze different cells at each input length, as well as to observe which model achieved the best results for fixed input length with varied prediction length. Likewise, the models were also evaluated with varied input length while the prediction length was fixed. This was done to analyze whether a shorter or longer input length improved the models accuracy in predicting a fixed number of time steps ahead.

1.3 Organization of Thesis

The rest of the thesis is organized as follows. Chapter 2 discusses the background, which includes Recurrent Neural Networks in Section 2.1, Sequence to Sequence Recurrent Neural Networks in Sections 2.2, the three different cells: RNN, GRU and LSTM, in Section 2.3, and is wrapped up with Section 2.4 which gives the general concept of the S2S attention architecture.

Chapter 3 covers the related work, which provides literature discussing energy forecasting methods in Section 3.1. Section 3.2 provides the use of S2S models in forecasting energy and other domains, while Section 3.3 concludes the chapter with S2S-attention based models.

Chapter 4 describes the core of the thesis: the methodology. Section 4.1 defines the feature selection and the evaluation process. Section 4.2 explains the sample generation process. Section 4.3 describes the algorithm of the regular S2S model, while Sections 4.4 and 4.5 describe the different S2S attention algorithms.

Chapter 5 provides the experiments and results. Section 5.1 discusses the experiments that were taken to produce the results. These results, along with comments and discussions, are then given in Section 5.2. Chapter 6 concludes the paper and discusses future work.

Chapter 2

Background

This chapter first introduces DNNs and RNNs in Section 2.1, and provides a general view of S2S RNNs in Section 2.2. Furthermore, it describes the functionality of the three RNN-based cells used in the S2S models in Section 2.3: the Vanilla RNN, Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) cells. Lastly, a general explanation of the two S2S attention mechanisms is provided in Section 2.4.

2.1 Recurrent Neural Networks

Deep Neural Networks are ML models that can be thought of as much larger ANNs, where an ANN is a neural network containing hidden layers between the input and output layer. Generally, a FeedForward Neural Network consists of a set of neurons, or nodes, and a set of directed edges between them. Associated with each node j is an activation function $a_j(\cdot)$. Associated with each edge, from node j' to node j, is a weight $w_{j'j}$, where the index denotes "from-to" notation. The output o_j from each node j is calculated by applying $a_j(\cdot)$ to a weighted sum of the outputs of the previous nodes, given as:

$$o_j = a_j\Big(\sum_{j'} w_{j'j} \cdot o_{j'}\Big) \tag{2.1}$$

Let us denote the weighted sum of outputs of previous nodes as s_j. The computation of o_j can be visually described as in Fig. 2.1, where the circles constitute nodes and the arrows represent the edges connecting them. Here, the inputs and outputs of previous nodes are denoted as $x_{j'}$ and $o_{j'}$ respectively for $j' \in 1, 2, 3$.

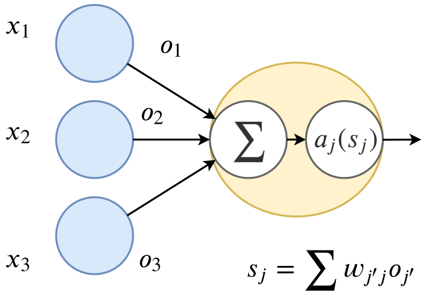


Figure 2.1: A neural network node computes a nonlinear function of a weighted sum of inputs.

Common choices for the activation function are the sigmoid $\sigma(z) = 1/(1 + e^{-z})$ and tanh function $\phi(z) = (e^z - e^{-z})/(e^z + e^{-z})$. However, for regression problems, it is common to omit an activation function and simply use the sum s_j as the output o_j.

An ANN is made up of these nodes and edges, with the architecture constituting of an input layer, hidden layers, and an output layer, as can be seen in Fig. 2.2 (a). In comparison, a DNN, as seen in Fig. 2.2 (b), is a larger ANN with more hidden layers and usually more nodes per each hidden layer.

DNNs have achieved success in solving many problems, such as speech recognition [20] [21], speech synthesis [22] [23], object detection in images [24] [25], mitosis detection in breast cancer histology images [26], and brain tumour segmentation [27]. Nonetheless, commonly used DNN architectures, such as FFNNs, are not well-suited for predicting a sequence consecutively, since DNNs directly predict the sequence. Meaning, if a time series vector is the desired predicted sequence, a feedforward DNN predicts this entire vector directly in the output layer, all at once. Hence, DNNs generate predictions based solely on the current input, irrelevant of any prior inputs, thus neglecting temporal dependencies present in time series problems. RNNs provide a solution by using an internal state to remember information in

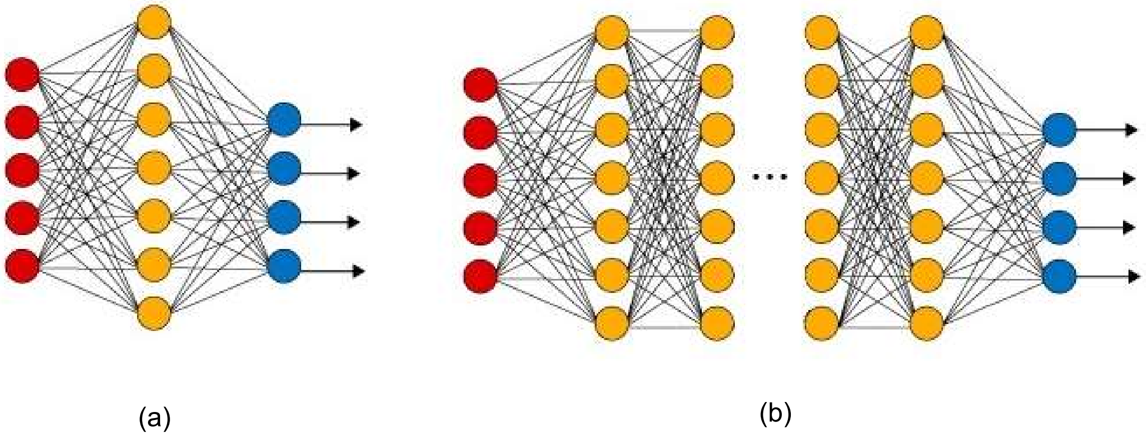


Figure 2.2: (a) ANN vs (b) DNN. A DNN contains more hidden layers than an ANN.

sequential time steps.

Non-S2S RNNs take a sequence of inputs $x_{[1]}, ..., x_{[T]}$, and previous hidden states $h_{[1]}, ..., h_{[T-1]}$, to compute a sequence of outputs $y_{[1]}, ..., y_{[T]}$, where j is a time step, $j \in 1, ..., T$. We can define $y_{[j]}$ as:

$$
\begin{aligned}
y_{[j]} &= f^{\circ}(\{x_{[1]}, ..., x_{[j-1]}, x_{[j]}\}, \{h_{[1]}, ..., h_{[j-1]}\}) && \text{(2.2a)} \\
&= f^{\circ}(x_{[j]}, h_{[j-1]}) && \text{(2.2b)}
\end{aligned}
$$

Equation (2.2a) refers to the sequence of inputs and hidden states needed, through time, to reach the computation of $y_{[j]}$, while equation (2.2b) denotes the specific input and hidden state needed at time j to compute $y_{[j]}$. Here, f° is a non-linear function, potentially combined of either a vanilla RNN, LSTM, or GRU cell, and some multi-layered network. As can be seen from Fig. 2.3, the inputs and hidden states used to generate $y_{[T-1]}$ are also used to generate $y_{[T]}$, but the reverse is not true.

These RNNs are convenient when trying to predict the next word in a sentence since the RNN will learn to predict the word “blue” if given the previous inputs “the, sky” and current

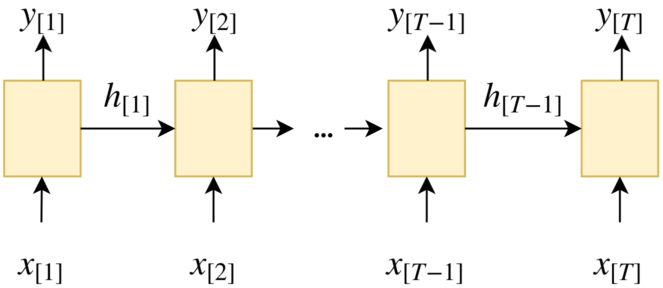


Figure 2.3: Sample passed through the Non-S2S RNN.

input "was". However, the method is weaker when trying to predict a consecutive sequence of words. For example, given the inputs "the, sky, was, blue", Non-S2S RNNs will struggle to predict the consecutive sequence "so, I, went, outside" and will be unable to predict a sequence of different length such as "so, I, went, outside, to, play, with, my, friends." Non-S2S RNNs are only able to output a prediction sequence that is less than or equal to the input sequence length. In terms of energy forecasting, if the usage inputs [400, 425, 450, 475] kW were passed to the Non-S2S RNN, it would adequately predict the next usage to be 500 kW. However, for the same usage inputs, the Non-S2S RNN will struggle to predict the consecutive sequence [500, 475, 450, 425] kW, and will be unable to predict a sequence of longer length, such as [500, 475, 450, 425, 400, 400, 405] kW.

2.2 Sequence to Sequence Recurrent Neural Networks

In comparison to Non-S2S RNNs, S2S RNNs contain two RNNs, an Encoder and Decoder RNN. The general idea, as shown in Fig. 2.4, is to first pass the input sequence, a sequence of vectors $x_{[1]}, ..., x_{[T]}$, into the Encoder RNN, one time step at a time, to obtain a context vector ($\vec{c}$). A common approach is to use an Encoder RNN such that:

$$h_{[j]} = f^*(x_{[j]}, h_{[j-1]}) \tag{2.3}$$

and

$$\vec{c} = q(\{h_{[1]}, ..., h_{[T]}\}) \tag{2.4}$$

where $h_{[j]}$ is the hidden state at time j, otherwise known as the encoder output at time j, and $\vec{c}$ is generated from the sequence of hidden states. Note that the outputs from equations (2.3) and (2.2a), $h_{[j]}$ and $y_{[j]}$ respectively, are different. Hence, $y_{[j]}$ is the final predicted output at time j, which is used to compute the loss and further train the model, while $h_{[j]}$ is simply the hidden state, or encoder output, computed at time j. Continuing, the functions f^* and q are generally non-linear functions such as in the work of Sutskever *et al.* [16], where f^* denoted an LSTM and $q = h_{[T]}$ was simply the last hidden state produced by the encoder.

This context vector is an encoded representation of the processed input sequence and is then passed to the Decoder RNN which extracts information at each unraveled time step to obtain the output sequence $\dot{y}_{[1]}, ..., \dot{y}_{[N]}$, where $i \in 1, ..., N$. The decoder is trained to predict the next output $\dot{y}_{[i]}$ from the sequence of previously predicted outputs $\{\dot{y}_{[1]}, ..., \dot{y}_{[i-1]}\}$, sequence of decoder hidden states $\{h^*{}_{[1]}, ..., h^*{}_{[i-1]}\}$, and information provided by $\vec{c}$. See that, from equations (2.3) and (2.4), $\vec{c}$ has already encoded information from the input sequence. Obtaining $\dot{y}_{[i]}$ is defined as:

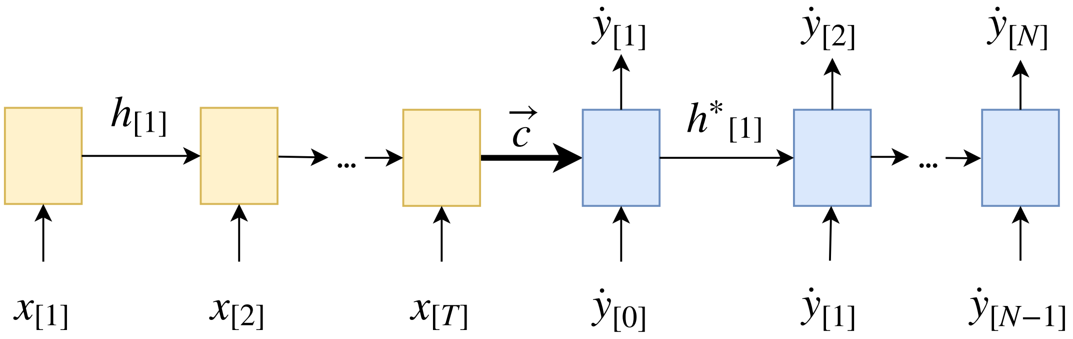


Figure 2.4: Sample passed through the S2S RNN.

$$\dot{y}_{[i]} = g^*(\{\dot{y}_{[1]}, ..., \dot{y}_{[i-1]}\}, \{h^*{}_{[1]}, ..., h^*{}_{[i-1]}\}, \vec{c}\,) \tag{2.5a}$$

$$= g^*(\dot{y}_{[i-1]}, h^*{}_{[i-1]}) \tag{2.5b}$$

where

$$\dot{y}_{[0]} = q^*(\vec{c}) \tag{2.6}$$

Here, equation (2.5a) denotes the needed variables to reach the computation of $\dot{y}_{[i]}$, while equation (2.5b) denotes the specific variables needed, at time i, to compute $\dot{y}_{[i]}$. Similarly as before, g^* is some non-linear function. See that $\dot{y}_{[0]}$ is the context value, derived from $\vec{c}$ as obtained in equation (2.6), and is used as the initial input for the Decoder RNN. Also, in the equations of (2.5), $\vec{c}$ is not explicitly used in generating any $\dot{y}_{[i]}$ other than $\dot{y}_{[1]}$, as seen in Fig. 2.4. The authors write $\vec{c}$ as a parameter of g^* in equation (2.5a) to imply the "extracting of information" from the input sequence.

Thus, the use of two RNNs strengthens consecutive sequence prediction, while also allowing the time dimensionality of inputs and outputs to vary. Hence, regardless of the input sample length T ([400, 425, 450, 475] kW = 4 steps), the output can be an arbitrary N time steps ([500, 475, 450, 425, 400, 400, 405] kW = 7 steps). This feat is unattainable for Non-S2S RNNs as they can not produce a prediction sequence that is longer than the input sequence.

2.3 Vanilla RNN, LSTM and GRU Cells

This subsection provides a brief overview of the algorithms in vanilla RNN, Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) cells. Vanilla RNN cells were first proposed by Elman [28] in an attempt to tackle time series problems and their long-term temporal dependencies. To obtain the current output, Vanilla RNN cells not only consider the current input, but also the output of RNN cells preceding it. The computations of carrying memory forward at arbitrary time step t and for input $x_{[t]}$ is given as:

$$h_{[t]} = \tanh(W_{xh}x_{[t]} + b_{xh} + W_{hh}h_{[t-1]} + b_{hh}) \tag{2.7}$$

In Vanilla RNN cells, their is only one calculation that occurs to produce the current hidden state at time t, otherwise known as the output at time t, which can be seen in Fig. 2.5. Here, h is the size of the hidden state, tanh is the activation function, $W_{xh} \in \mathbb{R}^{h \times x}$ is the input-hidden weight matrix, and $W_{hh} \in \mathbb{R}^{h \times h}$ is the hidden-hidden weight matrix. These weight matrices are the parameters to be learned during training. Similarly, $b_{xh} \in \mathbb{R}^{h}$ is the input-hidden bias and $b_{hh} \in \mathbb{R}^{h}$ is the hidden-hidden bias parameters to also be learned during training.

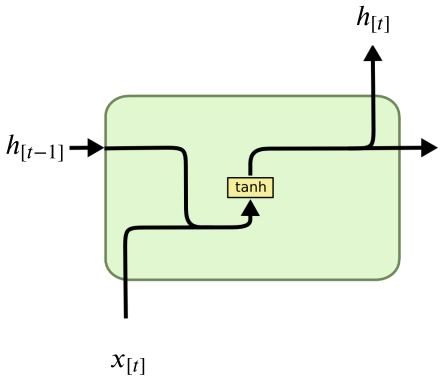


Figure 2.5: Vanilla RNN cell architecture [29].

The parameters can be thought of as how much value to assign to the current input and past hidden state. The error that the parameters produce will return via back-propagation [30] and be used to adjust the weights until the error has reached a minimum; this is deemed convergence. The activation function tanh is used to “squash” the input, which is a standard tool for condensing very large or very small values into a $[-1, 1]$ domain, it also helps to achieve non-exploding gradients during back-propagation. In machine learning, the gradient is simply a vector which gives the direction of the maximum rate of change, computed from the partial derivatives of the error function.

However, even with the tanh activation function, using traditional RNN cells for longer sequences will lead to the vanishing gradient problem [31]. Hence, as a network becomes longer, the calculated partial derivatives used to compute the gradient become increasingly smaller as back-propagation reaches the early layers of the network. Since the gradients dictate how much the network learns during training, very small or zero gradients will lead to little or no training, causing poor predictive performance.

Thus, LSTM networks [32] are a form of RNNs that were designed to specifically overcome the problem of vanishing gradient, producing a model able to store information for longer periods of time. These LSTMs are comprised of cells, which contain internal mechanisms called gates that perform actions on the flow of information. The general idea behind these gates is that they learn which data in the given sequence is meaningful and should be kept, and which data can be forgotten. By doing so, relevant information can be passed down through longer sequences and, hence, the model can make better predictions.

The LSTM cell architecture, as shown in Fig. 2.6, contains three gates (input i, forget f and output o), an update step g, a cell memory state c, and a hidden state h. The computations in a single LSTM cell at time t, for input x, are given as [32]:

$$i_{[t]} = \sigma(W_{xi}x_{[t]} + b_{xi} + W_{hi}h_{[t-1]} + b_{hi}) \tag{2.8a}$$

$$f_{[t]} = \sigma(W_{xf}x_{[t]} + b_{xf} + W_{hf}h_{[t-1]} + b_{hf}) \tag{2.8b}$$

$$g_{[t]} = \tanh(W_{xg}x_{[t]} + b_{xg} + W_{hg}h_{[t-1]} + b_{hg}) \tag{2.8c}$$

$$o_{[t]} = (W_{xo}x_{[t]} + b_{xo} + W_{ho}h_{[t-1]} + b_{ho}) \tag{2.8d}$$

$$c_{[t]} = f_{[t]} \odot c_{[t-1]} + i_{[t]} \odot g_{[t]} \tag{2.8e}$$

$$h_{[t]} = o_{[t]} \odot \tanh(c_{[t-1]}) \tag{2.8f}$$

Here, σ is the sigmoid activation function, tanh represents the hyperbolic tanh activation function, and the $\odot$ stands for element-wise multiplication. The W_x's are the input-hidden weight matrices, and W_h's are the hidden-hidden weight matrices parameters learned during training. Similarly, the b_x's and b_h's are the biases learned during training.

The forget gate in equation (2.8b) decides what information to keep and what information to forget. It does so by passing the current input $x_{[t]}$ and previous hidden state $h_{[t-1]}$ through a sigmoid function, which assigns a value between 0 and 1, to be element-wise multiplied to the cell state in equation (2.8e). A value closer to 1 means "keep this information", while a value closer to 0 means "forget". The input and output gates work similarly.

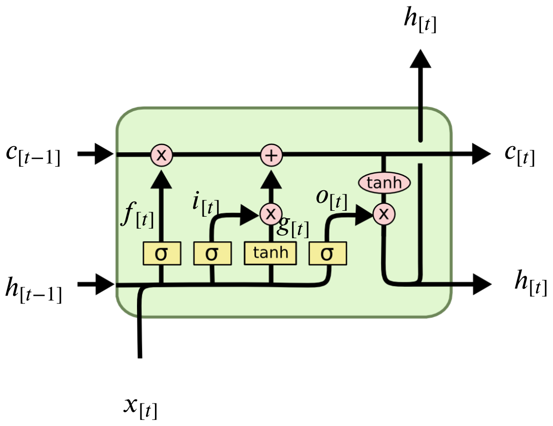


Figure 2.6: LSTM cell architecture [33].

The GRU model [34] was recently introduced to simplify the LSTM model, while maintaining similar functionality. The GRU cell architecture is provided in Fig. 2.7. GRU cells differ from LSTM cells by merging the cell memory state and hidden state into one all-purpose hidden state h and also by combining the input and forget gates into a single update gate z. Introduced is the reset gate r, which moderates the impact of the previous hidden state on the new hidden state, as can be seen in the update step k in equation (2.9c). The computations in a single GRU cell are given as [34]:

$$r_{[t]} = \sigma(W_{xr}x_{[t]} + b_{xr} + W_{hr}h_{[t-1]} + b_{hr}) \tag{2.9a}$$

$$z_{[t]} = \sigma(W_{xz}x_{[t]} + b_{xz} + W_{hz}h_{[t-1]} + b_{hz}) \tag{2.9b}$$

$$k_{[t]} = \tanh(W_{xk}x_{[t]} + b_{xk} + r_{[t]} \odot (W_{hk}h_{[t-1]} + b_{hk})) \tag{2.9c}$$

$$h_{[t]} = (1 - z_{[t]}) \odot k_{[t]} + z_{[t]} \odot h_{[t-1]} \tag{2.9d}$$

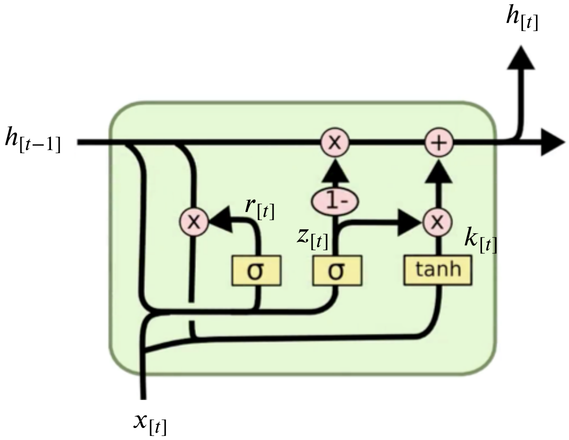


Figure 2.7: GRU cell architecture [35].

where the σ, tanh, and $\odot$ are equivalently used as in the LSTM cell. Note that there is one less learnable input-hidden weight matrix W_x as compared to the LSTM cell. This is also true for W_h, b_x, and b_h. Thus, GRUs have fewer tensor operations, less parameters, and they omit an internal cell state. This means that training and convergence are achieved faster on GRUs, while nonetheless, they contain enough gates and hidden state dimension for long-term retention.

2.4 Attention Mechanism

Attention was introduced to S2S RNNs by Bahdanau *et al.* [17] to improve against the underlying concerns the S2S models possess. Namely, the Encoder RNN is responsible for compressing all significant information of an input sequence into a single, fixed-length, context vector ($\vec{c}$). For longer input sequences, this task of compressing becomes a burden on the Encoder, and some valuable information may be lost. Furthermore, with only a single context vector being passed from Encoder to Decoder, and for longer prediction lengths, the Decoder is burdened by trying to make predictions using information from this single vector, notably in the latter ends of the decoding process. Thus, a mechanism called "attention" is added to the S2S model, with the main objective of storing information from the input sequence, then using this information to aid in the decoding process. Bahdanau *et al.* [17], followed by Luong *et al.* [18], applied attention-based architectures to S2S models for Neural Machine Translation (NMT) in 2014 and 2015 respectively.

The Bahdanau *et al.* attention (BA) mechanism, as originally shown in Fig. 2.8, was the first form of attention applied to S2S models for NMT. Since NMT is ultimately a classification problem, this section will present the *general idea* of applying BA to S2S regression problems. Hence, as seen previously in section 2.2, the S2S architecture without BA predicts the output $\dot{y}_{[i]}$ by the equations in (2.5). With BA applied, the new architecture first obtains the current decoder hidden state, at time i, by:

$$h^b_{[i]} = f^b([\dot{y}_{[i-1]}; c^b_{[i]}], h^b_{[i-1]}) \tag{2.10}$$

Then the predicted value is obtained by:

$$\dot{y}_{[i]} = g^b(\dot{y}_{[i-1]}, c^b_{[i]}, h^b_{[i]}) \tag{2.11}$$

See that the superscript b is used to denote the variables used in the Bahdanau attention mechanism. Also, the subscript notation, where i is used to denote decoder variables and j is used to denote encoder variables. Here, f^b takes the previously predicted output $\dot{y}_{[i-1]}$, the attention context vector $c^b_{[i]}$, and the previous hidden state $h^b_{[i-1]}$. Note that $\dot{y}_{[i-1]}$ and $c^b_{[i]}$ in equation (2.10) are concatenated and we use the $[\cdot]$ to denote concatenation. We let f^b

denote either a vanilla RNN, LSTM, or GRU cell and g^b some non-linear function. Hence, g^b is arbitrary and should be used according to the specific problem domain. For example, g^b could be a DNN with many hidden layers and nodes, or it could be one fully-connected layer. Continuing, the vector $c^b_{[i]}$, no relation to the cell state as in equation (2.8e), depends on the sequence of encoder outputs $h_{[1]}, ..., h_{[T]}$ as obtained in equations (2.3) and (2.4). Each encoder output $h_{[j]}$ contains information regarding the j-th part of the input sequence. The vector $c^b_{[i]}$ is computed as a weighted sum of these encoder outputs:

$$c^b_{[i]} = \sum_{j=1}^{T} \alpha^b_{[ij]} h_{[j]} \tag{2.12}$$

where the attention weight $\alpha^b_{[ij]}$ of each encoder output j is computed by:

$$\alpha^b_{[ij]} = \frac{\exp(e^b_{[ij]})}{\sum_{k=1}^{T} \exp(e^b_{[ik]})} \tag{2.13}$$

And the attention energies $e^b_{[ij]}$ are computed as:

$$\begin{aligned} e^b_{[ij]} &= S(h^b_{[i-1]}, h_{[j]}) && \text{(2.14a)} \\ &= v^\intercal \tanh(W[h^b_{[i-1]}; h_{[j]}]) && \text{(2.14b)} \end{aligned}$$

Here, S is seen as an *alignment model* which scores how well the inputs around time j and the output at time i match. The model uses the previous decoder hidden state and the j-th encoder output of the input sequence. In equation (2.14b), S consists of the tanh activation function, a fully-connected layer W that takes the concatenation of $h^b_{[i-1]}$ and $h_{[j]}$, and is parametrized by a transposed variable v to allow S to be jointly trained with the rest of the system. The alignment model directly computes a soft alignment, which allows the gradient of the cost function to be back-propagated through; meaning S and the entire model can be jointly trained by this gradient.

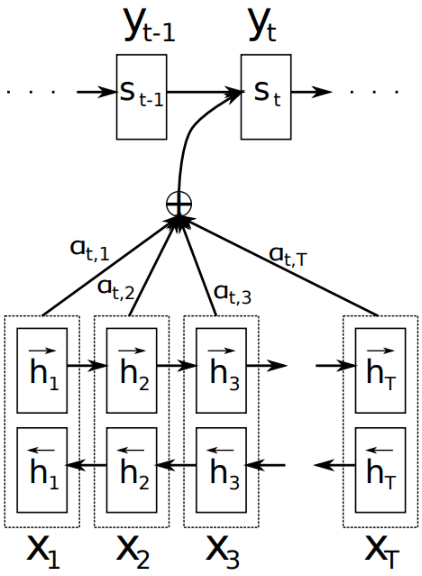


Figure 2.8: The original Bahdanau *et al.* attention mechanism [17] for decoder time step t. Here, the attention weights $\alpha_{t,j}$ are shown as being multiplied by the encoder outputs $\vec{h}_j$, then this product is joined with the previous hidden state s_{t-1} to produce the current hidden state s_t, which ultimately leads to the final output y_t.

We can understand $\alpha^b_{[ij]}$ as the probability that the target output $\dot{y}_{[i]}$ is aligned to, or most derived from, the input vector $x_{[j]}$. Specifically, the decoder, at the i-th step, will pay a percentage of attention to the j-th encoder output in producing the target output. For example, $\alpha^b_{[35]} = 0.09$ means for the 3rd decoder step, the target output $\dot{y}_{[3]}$ will have paid 9% attention to the 5th encoder output, since this was the alignment weight computed. Hence, the attention weight $\alpha^b_{[ij]}$, and its associated energy $e_{[ij]}$, reflect the importance of each encoder output $h_{[j]}$ in generating the next hidden state $h^b_{[i]}$ and prediction value $\dot{y}_{[i]}$. This allows the decoder to pay attention to specific parts of the input sequence; which in turn, relieves the encoder of having to encode all information into a single fixed length vector.

Furthermore, Luong *et al.* [18] developed both global and local attention-based models for NMT, differing whether the attention is concentrated on a few input positions (local) or on all (global). For the remainder of this thesis, Luong *et al.* attention (LA) will refer to the global model, as originally given in Fig. 2.9. Similarly as for BA, this work will present the *general* *idea* of applying LA to S2S regression problems. With LA applied, the system first computes an attentional decoder hidden state:

$$\hat{h}^l_{[i]} = \tanh(W[h^l_{[i]}; c^l_{[i]}]) \tag{2.15}$$

Then the predicted value is obtained by passing $\hat{h}^l_{[i]}$ through some non-linear function:

$$\dot{y}_{[i]} = g^l(\hat{h}^l_{[i]}) \tag{2.16}$$

In equation (2.15), the current hidden state $h^l_{[i]}$ is concatenated with the attention context vector $c^l_{[i]}$, then passed through a fully-connected layer W and the tanh activation function. The attentional hidden state $\hat{h}^l_{[i]}$ is only used to compute the final output, while the current hidden state $h^l_{[i]}$ is the hidden state that is passed to the next cell. Note the main difference between BA and LA: BA applies the attention mechanism before the variables are passed through the

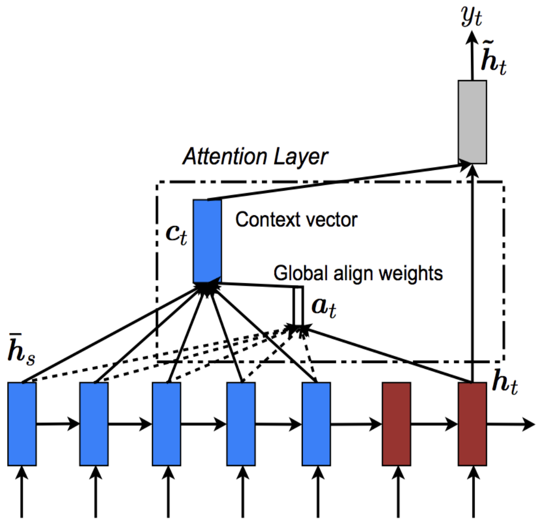


Figure 2.9: The original Luong *et al.* attention mechanism [18] for decoder time step t. Here, the current hidden state h_t is used to compute the attention weights α_t by one of three score functions. The context vector c_t is then obtained by the inner product of the attention weights α_t and encoder outputs $\bar{h}_s$. This context vector and the current hidden state are used in producing the attentional hidden state $\tilde{h}_t$, which leads to producing the final output y_t.

respective RNN cell, while LA applies the mechanism to the outputs of that respective cell. BA also generates the next hidden state from their attention mechanism in equation (2.10), while the next hidden state in LA is generated directly from the respective RNN cell.

Continuing with LA, the vector $c^l_{[i]}$, along with attention weights $\alpha^l_{[ij]}$, are similarly computed as in equations (2.12) and (2.13) respectively. However, the attention energies $e^l_{[ij]}$ for LA are computed by:

$$e^l_{[ij]} = S(h^l_{[i]}, h_{[j]}) \tag{2.17}$$

where the score function S has three different alternatives:

$$S(h^l_{[i]}, h_{[j]}) = \begin{cases} {h^l_{[i]}}^\intercal h_{[j]} & \textit{dot} \\ {h^l_{[i]}}^\intercal W(h_{[j]}) & \textit{general} \\ v^\intercal \tanh(W[h^l_{[i]}; h_{[j]}]) & \textit{concat} \end{cases} \tag{2.18}$$

Another difference between BA and LA: BA only utilizes one score function, the *concat* product, while LA utilized all three in equation (2.18). Lastly, the computation path to obtain the output $\dot{y}_{[i]}$ is marginally simpler for LA. Hence, $\mathrm{GRU}(\cdot) \rightarrow h^l_{[i]} \rightarrow \alpha^l_{[ij]} \rightarrow c^l_{[i]} \rightarrow \hat{h}^l_{[i]} \rightarrow \dot{y}_{[i]}$, as compared to that of BA, $h^b_{[i-1]} \rightarrow \alpha^b_{[ij]} \rightarrow c^b_{[i]} \rightarrow \mathrm{GRU}(\cdot) \rightarrow h^b_{[i]} \rightarrow [\cdot] \rightarrow \dot{y}_{[i]}$. Namely, BA is slightly more complicated in the latter part of the mechanism, as the variables are concatenated before being passed to the function g^b. Ultimately, LA serves the same purpose as BA; to relieve the encoder of having to extract all information from the input sequence into a single fixed-length vector.

Chapter 3

Literature Review

This chapter provides an overview of the literature relevant to the work presented in this thesis. Section 3.1 presents related work that is applicable to energy forecasting. Sections 3.2 and 3.3 discuss works that have incorporated S2S models and S2S attention models, respectively. Note that Section 3.3 mainly focuses on problems in the classification domain as to the best of our knowledge, only one other work has adapted a S2S attention model for a regression problem.

3.1 Energy Forecasting

Energy forecasting can be classified into three main categories: short, medium, and long-term [36][37]. Firstly, it is important to note that the forecasting lengths are directly defined with regards to the sample rate, the increment time between data entries [38]. Hence, datasets with hourly or daily increments will define forecasting lengths differently as compared to minute-increment datasets. For example, a distribution company that supports revenue projection may define short term as one to five years ahead, and long term as 20 years ahead [38]. In addition, the operations group in an Independent System Operator may define short term as a few hours ahead, medium term as five days ahead, and long term as two weeks ahead [38].

Thus, in order to properly compare different forecasting models, the sample rate of the datasets would have to be equivalent, along with the total number of time steps predicted ahead.

Nonetheless, since the presented work is concerned with five minute-increment data, we define the different forecasting lengths as:

- *short-term*: predict next thirty minutes to an hour ahead
- *medium-term*: predict next few hours to half a day ahead
- *long-term*: predict next day ahead

There are many approaches to load forecasting (physics, statistics, and machine learning-based), but this section focuses on machine learning-based models as our work belongs to this category. The purpose of these algorithms is to learn the mathematical relationship amongst the variables which affect energy consumption.

Traditional machine learning methods, such as ANNs and Support Vector Machines (SVM), have been used to forecast energy consumption. The work by Jetcheva *et al.* [39] proposed an ANN model to forecast day-ahead building-level energy, with an ensemble approach to select model parameters. Ferlito *et al.* [40] proposed a Nonlinear Auto Regressive Neural Network (NAR), a form of ANN, to forecast building energy demand for forecasting lengths 3, 6 and 12 months ahead. Their results show that the NAR model achieved, in terms of root mean square percentage error, 15.7%, 17.97%, and 14.59% for the three different forecasting lengths. It is important to note that the aforementioned study used a dataset of monthly energy consumption. The work by Chae *et al.* [41] considered the use of an ANN model, with Bayesian regularization algorithm, to predict short-term building energy usage. On a 15-minute interval dataset, the results demonstrated that the model, with adaptive training methods, outperformed Linear Regression, SVM, and Gaussian process regression. Accordingly, Araya *et al.* [42] proposed an ensemble learning framework for anomaly detection in building energy consumption. Their work developed a new pattern-based anomaly classifier, the collective contextual anomaly detection using sliding window (CCAD-SW), then implemented an ensemble framework that consisted of pattern-based and prediction-based anomaly classifiers. Hence, part of the framework consisted of ANN and SVM models to first predict energy consumption, then the predicted values were compared to the actual usage values to distinguish if an anomaly had occurred or not. The results showed that the CCAD-SW improved the true positive rate of the CCAD (without the sliding window) by 15% and reduced the false positive rate by 8%. Moreover, the ensemble framework further improved the true positive rate of the CCAD-SW by 3.6% and reduced the false positive rate by 2.7%.

A review of ML models for estimation of building energy consumption is presented by Seyedzadeh *et al* [43]. Specifically, they review how other researchers have used the three approaches: ANN, SVM, and Gaussian process regression, for energy forecasting. They do not state results, but rather provides a discussion of the benefits and drawbacks of each model.

The work by Naji *et al.* [44] predicted building energy consumption by applying an Extreme Learning Machine (ELM) method. The ELM method was applied to the data regarding building material thickness and their thermal insulation capability. The results showed that the ELM method was superior to genetic programming and ANNs, and that ELMs are faster in learning speed compared to traditional feedforward network algorithms, such as the BPTT algorithm. In comparison, the work by Tasfi *et al.* [45] focused on prediction-based anomaly detection with a model that consisted of a Convolutional Neural Network (CNN) Encoder-Decoder framework. The results showed their network outperformed SVM and Autoencoder variants in anomaly detection ability. The work by Amarasinghe [46] also used a CNN model to forecast energy consumption. Their results showed that the CNN architecture produced superior results to SVMs, while having comparable results to ANNs and other deep learning approaches. Mocanu *et al.* [37] assessed their newly developed stochastic models on a benchmark dataset consisting of almost four years of one-minute increment consumption data. Their results showed that the Conditional Restricted Boltzmann Machine and Factored Conditional Restricted Boltzmann Machine outperformed ANN, SVM, and regular non-S2S RNNs for short-term prediction lengths.

While the aforementioned works contribute to load forecasting in their respective ways, the presented work differs by focusing on both S2S-RNN based and S2S-RNN attention based models. These S2S-RNN models offer a stronger analysis in time series problems, since their internal hidden state is passed through a directed graph along a sequence. For this reason, regular S2S-RNN models are able to retain information in sequential data better than traditional ANNs, SVMs and CNNs [16]. Moreover, adding an attention mechanism to these (already superior) S2S-RNN models only holds potential in improving their ability to learn relationships in the same temporal data.

Other works have explored the use of non-S2S RNN models for load forecasting. The work by Kong *et al.* [47] proposed an LSTM based regular RNN model for short-term residential load forecasting (STLF). The model was applied to 69 datasets, that consisted of 30-minute increment readings. The results showed that the model achieved an average mean absolute percentage error (MAPE) of 44.06% when predicting 12 time steps ahead. In contrast, Shi *et al.* [48] worked with 30-minute increment residential datasets, where they applied a novel pooling based deep recurrent neural network (PDRNN) for STLF. The methodology consists of two stages: Stage one consisting of Load Profile Pooling and Stage two including STLF with a deep non-S2S LSTM based RNN. The results showed the PDRNN model outperformed ARIMA by 19.5%, Support Vector Regression by 13.1% and classical RNN by 6.5% in terms of root mean squared error. Moreover, non-S2S LSTM based RNN models were used for prediction-based anomaly detection in different time series domains, one of which was power demand [49]. These non-S2S LSTM models demonstrated that stacked LSTM networks outperformed stacked vanilla RNN networks and also learned higher-level temporal patterns without prior knowledge of the pattern duration. Moreover, Bouktif *et al.* [50] used a standard LSTM model, coupled with a genetic algorithm, for short to medium term aggregate load forecasting. The genetic algorithm was used to find optimal time lags and the number of layers for the LSTM model to improve prediction performance. The dataset had a sample rate of 30-minutes, and the models were trained to forecast a few days to months ahead. The results showed that the model achieved a mean absolute error (MAE) of 251 for the 2-week forecasting scenario, and a MAE of 208 for the 3-4 months scenario.

Although these four studies used LSTM based RNN models for load forecasting, their focus was on standard non-S2S prediction, rather than S2S prediction as in our work. The main difference, as described in section 2.2, is that standard non-S2S LSTM models use the outputs from the Encoder as predictions, while S2S models combine the Encoder and Decoder to use the sequential outputs from the Decoder as predictions. In addition, neither of the aforementioned works applied any sort of attention mechanism to their architectures.

3.2 Use of Sequence-to-Sequence Models

3.2.1 Energy Forecasting

This section reviews three different works that present models which are most comparable to our S2S non-attention model. Firstly, the work by Marino *et al.* [36] used standard LSTM and LSTM-based S2S models to forecast residential-level energy consumption on one hour and one-minute time step datasets. The S2S model performed well on both datasets, and produced comparable results with other deep learning methods [37]. While our work also uses S2S models, it varies by means of: overall different algorithm, sample generation, and longer prediction sequence length. Furthermore, in the aforementioned work, an important step in the S2S approach of using the previous Decoder output as next input was not utilized for their respective LSTM-S2S model.

Secondly, Zheng *et al.* [51] proposed a hybrid algorithm that combined similar days (SD) selection, empirical mode decomposition (EMD), and LSTM neural networks to construct a prediction model denoted SD-EMD-LSTM for STLF. The extreme gradient boosting-based weighted k-means algorithm was used to calculate the similarity between historical and forecasting days. Then, the EMD method was used to decompose the SD load to numerous intrinsic mode functions (IMFs) and residuals. From here, LSTM S2S-based networks were employed to forecast each IMF and residual. Once the forecasting values were produced from each LSTM model, they were passed through a series reconstruction phase to obtained the forecasted results. The dataset used was in one-hour increments, and the model was tested to forecast the next 24 hours (day-ahead) and 168 hours (week-ahead). The results showed the SD-EMD-LSTM model obtained an average MAPE of 1.08% and 1.59% for day-ahead and week-ahead respectively. Also, while this work proposed a unique hybrid model, the specific S2S LSTM-based model is identical to the one used by Marino *et al* [36]. The only difference is that Zheng *et al.* used a different input feature vector for their decoder network. Hence, our work differs by: not clustering based on similar days, and passing the usage data to the LSTM S2S model, not the IMF data and residuals. In addition, the work by Zheng *et al.* does not pass the previous Decoder output as the next input, as in our S2S approach.

Lastly, Rahman *et al.* [52] developed two S2S LSTM-based models to make medium-to-long term predictions, i.e. time horizon greater than one week, for commercial and residential datasets at one-hour sample rate. Model "A" has an encoder-decoder base, with the final encoder step outputting the context vector, similar to our work. The main difference between their work and ours: their work uses this encoded context vector as the input for each step in the decoder process, while our work does not. Hence, their decoder would take as inputs the context vector and the previous hidden state, while the decoder in our work takes as inputs the previously predicted output and the previous hidden state. These differences are identical for their respective model "B". What is interesting about this work is that they concatenate the original encoder inputs with the outputs of the decoder, and perform an operation, respective to model A or B, on this newly formed concatenation to achieve the forecasted sequence. The results showed that the proposed S2S LSTM-based models A and B, in general, performed better than a 3-layer ANN model in the case of electric load forecasting in commercial buildings. To conclude, even though the work by Rahman *et al.* is comparable to ours, it does not utilize the complete S2S approach as proposed for the task of NMT [16], a re-occurring theme for literature mentioned in this subsection.

3.2.2 Classification Problems

While our work primarily focuses on using S2S models for a regression problem, such as load forecasting, other works have used S2S models for tackling classification problems. The S2S architecture was first developed, by Cho *et al.* [34], to achieve improved performance in the field of statistical machine translation (SMT). Interestingly, this same work not only proposed a novel model architecture, but also a novel RNN cell structure, later to be known as the GRU unit. The model was evaluated on the task of translating from English to French. The results showed that the RNN encoder-decoder, with GRU units, improved the overall translation performance in terms of Bilingual Evaluation Understudy (BLEU) scores, obtaining a score of 34.64. Furthermore, the work by Sutskever *et al.* [16] introduced a slight variation to the RNN S2S framework for the same task of translating from English to French. While Cho *et al.* used a GRU cell, Sutskever *et al.* used an LSTM cell. The main technical difference between these two works: as was done by Cho *et al.* Sutskever *et al.* did not pass the encoded representation vector, the context vector, to the decoder at each time step. Nonetheless, the main result of their work was the following: on the WMT'14 English to French translation task, the proposed LSTM S2S model obtained a BLEU score of 34.81, which was the best achieved result up to that point. One could argue that the model by Sutskever *et al.* achieved better results because they used an LSTM cell, or that Cho *et al.* had not optimized the novel methodology at the time. Nevertheless, both works revolutionized how deep learning is used for problems in the domain of sequence prediction.

Since their origin, other works have explored the use of S2S RNN models for classification problems. The work by Venugopalan *et al.* [53] use an LSTM-based S2S model to generate descriptions of real-world videos. The architecture is slightly different than that of Cho *et al.* and Sutskever *et al.* as the raw video frames are first passed through a CNN, then these outputs are used as the encoder sequence inputs. In addition, their work [53] used a stacked layer approach, and the current decoder output of the last RNN layer was used as the next input **only** for that last layer. The first layer of the decoder was only passed the previous hidden state. The proposed model achieved state-of-the-art performance on a standard YouTube corpus dataset, and outperformed related work on two large and challenging movie-description datasets. Similarly, S2S RNN models have found success in producing vector representations of fixed dimensionality for variable-length audio segments [54]. Hence, an audio segment is first converted to data, then used as the input sequence of the encoder RNN. The remaining decoder architecture is as in Sutkever *et al.* where the predicted output is passed as the next input. The results showed that the proposed models were able to outperform different baseline Naive Encoders (NE) in terms of mean average precision (MAP).

Furthermore, an LSTM encoder-decoder network was used to improve the estimation of Remaining Useful Life (RUL) of a machine [55]. The proposed model obtained a health index (HI) for a system using multi-sensor time-series data. The model was trained to reconstruct the time-series corresponding to the healthy state of the system. The reconstruction error was used to calculate the HI, which is then used to estimate RUL. The results, obtained from using a real-world industry dataset, showed a strong correlation between the HI produced from the LSTM encoder-decoder and actual maintenance costs. Lastly, the work by Park *et al.* [56] explored the use of LSTM-based S2S models for real-time vehicle trajectory sequence prediction. The encoder was used to analyze the pattern of the past trajectory, while the decoder was used to generate the future trajectory sequence. The overall process used a beam search technique to keep the K most likely trajectory candidates from the decoder output. From highway traffic scenarios, the results showed the proposed model obtained improved prediction accuracy over traditional trajectory prediction techniques.

3.3 Use of Sequence-to-Sequence Attention Models

While RNN encoder-decoder models have found success in the problem domain of sequence prediction, the underlying issue of burdening the encoder to condense all information into one context vector is still present. Thus, an attention mechanism, otherwise known as an “alignment model", was added to these models by Bahdanau *et al*. [17]. Their work focused on using this novel architecture to maximize neural machine translation (NMT) performance. The proposed model was evaluated on the WMT'14 dataset, for the task of English to French translation. For sentence prediction length of up to 30 words and 50 words, the results showed the attention based model outperformed the Non-attention encoder-decoder model [34] [16] in terms of BLEU score. Shortly after, Luong *et al*. [18] proposed a new S2S attention mechanism for the same task of improving NMT performance. The technical differences between Bahdanau *et al*. attention (BA) and Luong *et al*. attention (LA) is given in Section 2.4. To summarize, LA introduced two new score functions, *dot* and *general*, on top of BA's *concat* function, and LA introduced local attention in addition to BA's global attention. The proposed model was tested on the WMT translation tasks between English and German in both directions. Similar to BA, their attention mechanism achieved better results over the regular S2S models [34] [16] in terms of BLEU score. Their ensemble model established new state-of-the-art results for both WMT'14 and WMT'15, which outperformed existing best systems by more than 1.0 BLEU. As described in Section 2.4, our work incorporates both the BA and LA attention mechanisms for the regression problem of predicting energy consumption.

The work by Qin *et al*. [57] proposed a dual-stage attention-based RNN (DA-RNN) to predict the NASDAQ 100 Stock for input lengths of 3, 5, 10, 15, and 25 minute time steps. The first stage is the Input Attention Mechanism which converts the original inputs to newly computed "attention inputs" to be used by the encoder RNN. The second stage is the Temporal Stage Mechanism, which incorporates the BA mechanism in their S2S model. The results showed that DA-RNN achieved state-of-the-art results, outperforming regular S2S models and S2S-attention based models. It is important to note that the model was used to only predict the next minute value, and that the decoder was passed the previous actual values throughout the entire decoder process, a technique otherwise known as Teacher Forcing. The proposed model achieved state-of-the-art performance in terms of MAPE. Our work differs as: no conversion was done to the original inputs, our S2S-attention models were tested for much longer prediction lengths, and no Teacher Forcing was done during the decoder process.

Other works have used the S2S-attention architecture for classification problems. The work by Prabhavalkar *et al.* [58] compared several S2S models for the task of speech recognition. The models included connectionist temporal classification (CTC), the RNN transducer, an attention-based model, and an augmented RNN transducer with attention mechanism. Their results showed that, on the large vocabulary continuous speech recognition (LVCSR) task, the baseline outperformed all models for the voice-search test. Nonetheless, the S2S attention-based models significantly outperformed the baseline on the test set of numeric entities, which require the model to map utterances from the spoken to the written domain. Furthermore, S2S attention based models have also been used for the task of Grapheme-to-Phoneme conversion [59]. Results showed that regular S2S models obtained comparable, but not better, performance to S2S attention based model. However, it was found that a bi-directional LSTM combined with an attention mechanism, as presented by Bahdanau *et al.* [17], significantly outperformed previous baseline methods for the Grapheme-to-Phoneme conversion task. Lastly, Zhang *et al.* [60], proposed a character-level S2S attention model for the task of understanding subtitles, where the applied attention mechanism was that by Bahdanau *et al.* [17]. Results showed that, on the task of English-to-Chinese subtitle translation, the proposed model achieved performance comparable to that of the most competitive word-based model, namely the Bahdanau *et al.* attention model. In addition, the model by Zhang *et al.* delivered results close to the well-established phrase-based system, Moses.

While the aforementioned works found success using attention based S2S models in their respective classification domains, our work differs as it adapts a S2S attention mechanism to a regression problem, namely that of energy consumption forecasting.

Chapter 4

Methodology

This section discusses the applied approach. First, the dataset and evaluation process are introduced in Section 4.1, followed by the sample generation in Section 4.2. Section 4.3 describes the S2S algorithm, while Sections 4.4 and 4.5 give the specific BA and LA attention-based S2S algorithms respectively.

4.1 Dataset and Evaluation Process

Smart meters are digital electricity meters that are able to measure how much electricity is used and when it is used. Typically, data obtained from smart meters contains the meter location, the time stamp, and usage recorded at each reading interval. Ultimately, for this thesis, one building-level smart meter raw data was processed and the dataset used for machine learning consists of readings with 9 features, as given in Table 4.1, where three readings from the dataset are randomly given to show how the data can vary. The holiday feature was added using the Ontario holiday calendar, while the remaining time features were engineered from the time stamp. Gray encoding could be used to deal with cyclical features, such as hour, in order to address the issue of rolling over from maximum to minimum (hour 23 to 0). Using gray encoding holds potential to further improve results obtained in our study.

The entire dataset was divided into a training set and test set. The first 80% of all readings

Table 4.1: Feature Data (before standardization).

Index	Month	Day of Year	Day of Month	Weekday	Weekend	Holiday	Hour	Season	Usage (kW)
0	7	187	5	1	0	0	0	3	405.8743
23470	9	268	24	5	1	0	18	4	332.7853
104275	7	184	3	0	0	1	7	3	297.4848

were assigned as the training set and the last 20% as the test set. Fig. 4.1 shows where the train set ends and the test set begins for the usage feature. This validation process was chose to ensure that the model is built using older data and tested on newer data. As seen in Fig. 4.1, the usage data describes a building that experiences a regular work-week. Hence, the spikes represent the typical Monday-Friday work-week, while the drops represent lower consumption during the weekend.

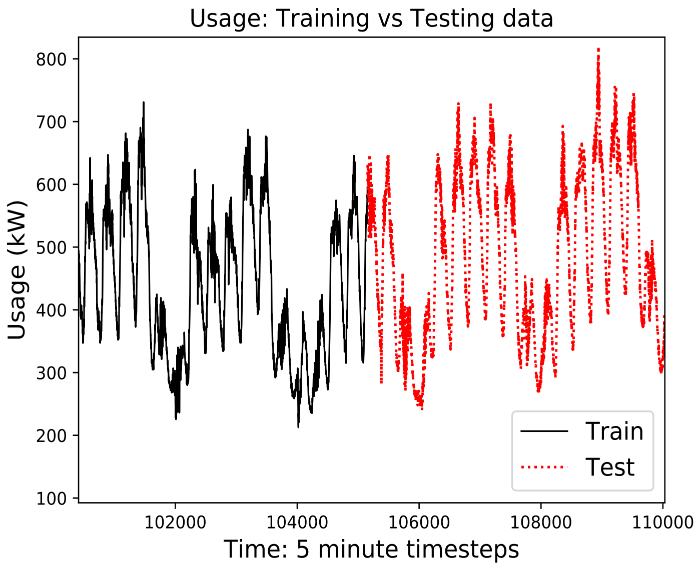


Figure 4.1: Usage data zoomed in to show breakdown between the train and test sets.

The data was normalized using standardization. Hence, the values of each feature in the data were transformed to have zero-mean and unit-variance:

$$\tilde{x} = \frac{x - \mu}{\sigma} \tag{4.1}$$

Here, x is the original feature vector, μ is the mean of that feature vector, σ is its standard deviation, and $\tilde{x}$ represents the feature vector after normalization.

4.2 Sample Generation

To enable the use of S2S models, the training and testing datasets need to be prepared in a different way as compared to traditional Feedfoward neural networks. More specifically, each target sample needs to be obtained *from* the end of each input sample since we want to train the model to predict a desired length ahead. Let one input sample be represented as a matrix, $X \in \mathbb{R}^{T \times f}$, where T is the number of input time steps and f is the number of features. As defined in the previous subsection, the number of features is $f = 9$, where each row of the matrix X is defined as:

$$\begin{aligned} x_{[j]} = [&\text{Month}_{[j]}\ \text{DayOfYear}_{[j]}\ \text{DayOfMonth}_{[j]} \\ &\text{Weekday}_{[j]}\ \text{Weekend}_{[j]}\ \text{Holiday}_{[j]}\ \text{Hour}_{[j]} \\ &\text{Season}_{[j]}\ \text{Usage}_{[j]}] \end{aligned} \tag{4.2}$$

where $j \in 1, ..., T$. For each input sample, one target sample was generated, represented by a vector $y \in \mathbb{R}^{N \times 1}$, where N represents the number of predicted time steps *from* T. The target vector represents the actual usage vector, which is given by:

$$y = [\text{Usage}_{[1]},\ ...\ ,\ \text{Usage}_{[i]},\ ...\ ,\ \text{Usage}_{[N]}] \tag{4.3}$$

where $y_{[i]}$, with subscript $[i]$ notation, denotes the actual usage value at time step i, $i \in 1, ..., N$. We use this target vector to compare with the predicted usage vector, $\dot{y} \in \mathbb{R}^{N \times 1}$.

Thus, the input and target samples were generated using the technique demonstrated in Fig. 4.2. Here, $i + 1$ denotes the index (of the dataset) where the window starts, T denotes the length of the input window, and N denotes the length of the target vector. Note that the use of notation i will refer to the index for the rest of this section, hence while only discussing sample generation. As an example, to predict the next hour of energy consumption using the previous four hours (in 5-minute increments), T and N would be set to $T = 48$ and $N = 12$. It is important to note that the **indices** (minus the last $T + N$) of the **training set** were first randomized, not the data itself, then the input and target samples were generated. This way, data in a single input sample represents consecutive time steps. The process for sample generation of the training set is given as:

1. Uniformly randomize indices i from $\mathcal{U}(1, i_{last})$. Here, $i \in 1, ..., i_{last}$ and i_{last} denotes the total length of the training set minus the last $T + N$ indices. If i was chosen to be $i_{last} + 1$, the input sample would still be length T, but the target sample would now be length $N - 1$, since it is not possible to exceed the available index of the training set.

2. For each i:

    (a) Append, from the training set, consecutive data from indices $i + 1$ to $i + T$ as the input sample. Hence, obtaining $X \in \mathbb{R}^{T \times f}$.

    (b) Append, from the training set, $i + T + 1$ to $i + T + N$ as the target usage vector. Hence, obtaining $y \in \mathbb{R}^{N \times 1}$.

Obtaining the input and target samples for the **test set** was slightly different. Instead, as seen in Fig. 4.3, a sliding window technique was used; the window shifted sequentially with

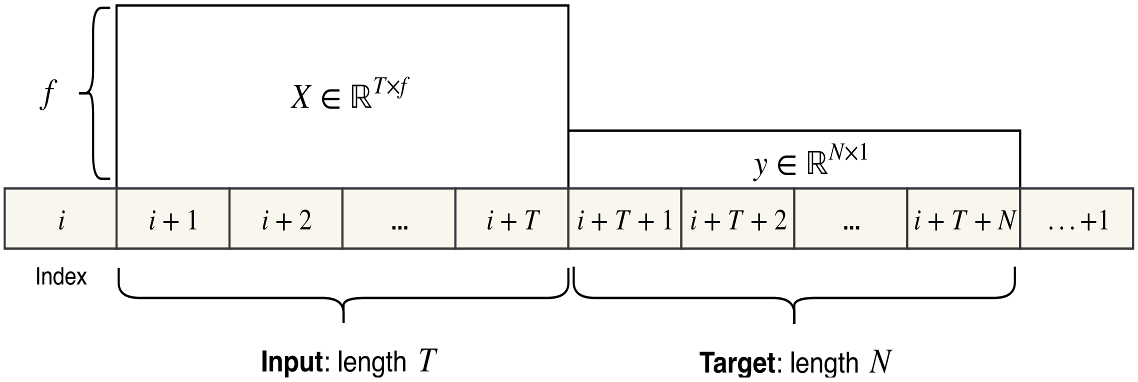


Figure 4.2: Training set sample generation. Input and target samples generated from random index $i + 1$.

the overlap size equivalent to the target length, N. Hence, if we obtain one test input sample starting at index $i' + 1$, the target sample still ends at $i' + T + N$, similar to what is done in Fig. 4.2 and the training set. However, the indices are not randomized for the test set, and the next input sample *slides* N time steps and starts at $i' + 1 + N$, with the target sample ending at $i' + T + 2N$. The process for sample generation of the test set is given as:

1. For each i', s.t. $i' \in 1, 1 + N, 1 + 2N, ..., i'_{last}$, where i'_{last} denotes total length of the test set minus the last $T + N$:
   (a) Append, from the test set, consecutive data from indices $i' + 1$ to $i' + T$ as the input sample. Hence, obtaining $X' \in \mathbb{R}^{T \times f}$.
   (b) Append, from the test set, $i' + T + 1$ to $i' + T + N$ as the target usage vector. Hence, obtaining $y' \in \mathbb{R}^{N \times 1}$.

Note that the process above accounts for the sliding window technique by starting at index

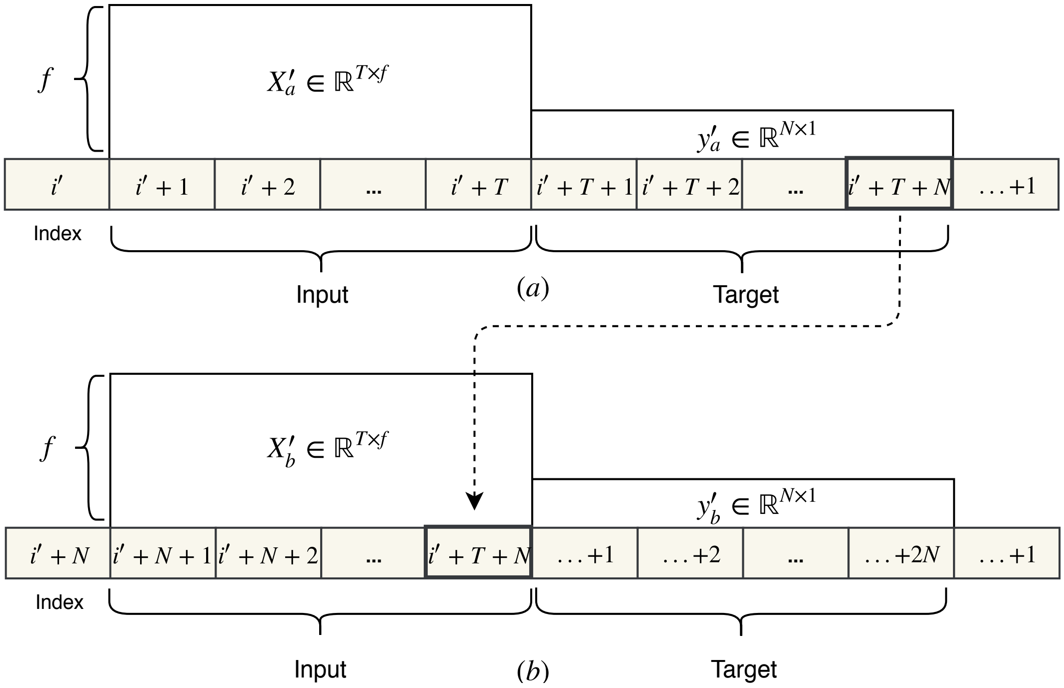


Figure 4.3: Test set sample generation. (a) Input and target samples generated from dataset at index $i' + 1$. (b) Sequential input and target samples generated from sliding window with overlap length N.

1 and adding N each time. Doing so allows for an overlap in input test samples, but **no** overlap in target test samples. This was done to enable **concatenating** the predicted usage vectors together into one vector which has equivalent length as the total actual usage vector. Thus, from here, comparisons between the predicted and actual usage can be done by means of accuracy measures.

4.3 S2S-prediction

This section gives a detailed breakdown of the S2S algorithm used for this work. As briefly described in section 2.2, each S2S model has an Encoder RNN (E_T) of length T and Decoder RNN (D_N) of length N. The inputs, as obtained in the previous subsection, are passed through E_T, one time step at a time. Hence, Fig. 2.4 shows E_T unrolled, where a vector, as in equation (4.2), is passed to it at each time step. Each cell takes the previous hidden state and the current input at time step j, performs the actions discussed in section 2.3 (equations (2.7), (2.8), (2.9) for a Vanilla RNN, LSTM or GRU cell, respectively), and outputs a hidden state. Note that Vanilla RNN and GRU cells output one hidden state, while LSTM cells output both a cell state and a hidden state.

Fig. 4.4 shows how E_T and D_N connect in the S2S approach. Once the entire input has been processed by E_T, the output produced is a hidden state commonly referred to as the context vector [16], since it encodes context from the entire input sequence. This context vector, shown as $h_{[T]}$ in Fig. 4.4, is then used as the initial hidden state for D_N. The context vector is also passed through a fully-connected layer to produce the feature context **value**, shown as $\dot{y}_{[0]}$, which is the initial input of D_N. Let us denote this fully-connected layer as $W^{h\to 1}$, where h represents the hidden state dimension, or the size of the hidden state vector that is passed through each cell. Note that the E_T cell took $f = 9$ features as input, while the D_N cell only takes $f = 1$ feature, hence the previously predicted usage value $\dot{y}_{[i-1]}$. Therefore, this $W^{h\to 1}$ transforms the output vector from dimension h to a single value of dimension 1, allowing it to be used as the next input value for D_N. This approach differs from others in literature as it passes the previously predicted output as the next decoder input; a novel approach to building-level load forecasting. Thus, let us denote these models with a "-o" notation, representing the distinguishing characteristic of the applied architecture for energy load forecasting; *one* predicted value computed at time step i is passed as the next decoder input. Hence, GRU S2S will be denoted as GRU S2S-o [61].

Continuing, each output produced from the remaining D_N cells is passed through the same fully-connected layer. This transforms the output $\bar{h}_{[i]}$ to a single predicted usage value, $\dot{y}_{[i]}$, at time step i. Thus, after N time steps we have obtained the predicted usage vector, and we can compute the loss from the target usage vector as:

$$L = \frac{1}{i}\sum_{i=1}^{N}(y_{[i]} - \dot{y}_{[i]})^2 \tag{4.4}$$

Where equation (4.4) is the objective function that is minimized after each epoch; it is calculated as the Mean Squared Error (MSE) of the target and predicted values.

To train the model, back-propagation through time (BPTT) is used. The entire network is

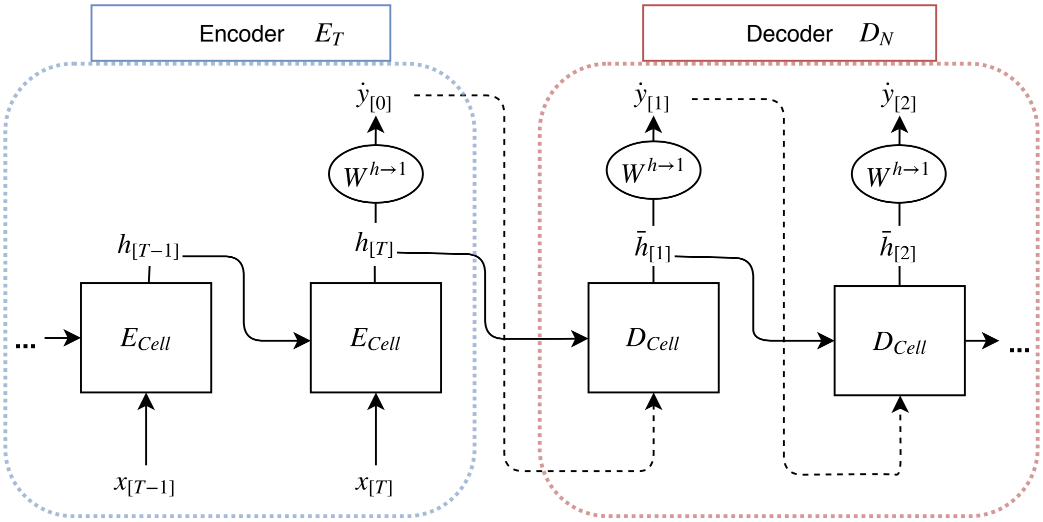


Figure 4.4: Detailed process of the S2S architecture. The left box shows the last two steps of E_T, while the right box shows the first two steps of D_N.

unrolled by a fixed number of time steps $T + N$; hence, it can be seen as a deep standard feed-forward neural network with shared parameters. Thus, standard BPTT can be applied to train the network using a gradient based method. Here, the ADAM [62] algorithm is used as the gradient based optimizer since it outperformed other methods in terms of faster convergence and better accuracy measures. The complete process is given in Algorithm 1.

In line 10, the hidden state is initialized to be $\vec{0}$ of size $B \times h$, where B is denoted as the batch size, which is the number of training examples the model processes for one update of weight parameters [63]. Batching is done to speed up convergence. Since the weight parameters are

Algorithm 1 Train-S2S($G = (E_T, D_N, W^{h\to1}, P_0)$)

```
 1: Input : Model $G$ consisting of: $E_T$ of length $T$, $D_N$ of length $N$, fully-connected layer
 2:   $W^{h\to1}$ and initial weight parameters denoted $P_0$.
 3: Output : Model $G$ trained to convergence, with updated weight parameters $P$.
 4:
 5: Generate randomized input samples $X \in \mathbb{R}^{T\times f}$ and corresponding target vectors $y \in \mathbb{R}^{N\times1}$
 6:
 7: for each epoch do
 8:   for each batch do
 9:
10:     initialize $h_{[j-1]} \leftarrow \vec{0} \in \mathbb{R}^{B\times h}$, where $B$ = batch size
11:
12:     for time step $j$ from 1 to $T$ do
13:       $h_{[j]} \leftarrow E_{cell}(x_{[j]}, h_{[j-1]})$
14:
15:     $\dot{y}_{[0]} \leftarrow W^{h\to1}(h_{[T]})$
16:     $\bar{h}_{[0]} \leftarrow h_{[T]}$
17:
18:     for time step $i$ from 1 to $N$ do
19:       $\bar{h}_{[i]} \leftarrow D_{cell}(\dot{y}_{[i-1]}, \bar{h}_{[i-1]})$
20:       $\dot{y}_{[i]} \leftarrow W^{h\to1}(\bar{h}_{[i]})$
21:       append $\dot{y}_{[i]}$ to predicted usage vector $\dot{y}$
22:
23:     compute $L$ as in equation (4.4)
24:     BPTT($L$)
25:     $P \leftarrow$ Adam($P_0, r$), where $r$ = learning rate
26:     $P_0 \leftarrow P$
27:
28: Return : Trained model $G$, with updated parameter weights $P$.
```

updated **per batch**, a smaller batch size provides more accurate updates. Once all the batches have been processed by the model, one full **epoch** has occurred. Lines 12-13 show the input example being passed through the E_T. Lines 15-16 transform the output of E_T, the context, to create the initial input and hidden state of the D_N. Lines 19-21 describe the process of predicting the usage value, appending that value, and preparing the next input and hidden state. In lines 23-26, the total loss is computed, BPTT is applied, and the model parameters are updated. See that Algorithm 1, Train-S2S, is used to train the model, and will differ slightly when applied to the test set. Namely, for the test set, we build the inputs and targets differently (as outlined in the section 4.2) and do not apply BPTT on the loss, hence do not update the parameters in any way.

4.4 S2S-prediction with Bahdanau Attention

Section 2.4 gives a generic breakdown of the BA mechanism, whereas here the specific process of applying BA to a S2S model to forecast building energy consumption is described. As in section 4.3, the S2S model with BA (S2S-BA) takes an identical encoder approach, except as can be seen in Fig. 4.5, the outputs of E_T are stored and used in the decoder attention process. Let us denote all of these encoder outputs as $\mathbf{H} = [h_{[1]}, ..., h_{[T]}] \in \mathbb{R}^{T \times h}$ where each output can be defined as $\mathbf{H}_{[j]} = h_{[j]} \in \mathbb{R}^{h}$. Here, the dimensions T and h are the input sequence length and hidden state size, respectively. Hence, at the end of the encoder process, we have obtained the context vector $h_{[T]}$, the initial input value $\dot{y}_{[0]}$ for D_N, and the encoder outputs $\mathbf{H}$. Even though the context vector $h_{[T]}$ is an element of $\mathbf{H}$, it can be seen as the initial hidden state of D_N; hence it is denoted as $h^b_{[0]}$. The BA process that occurs at decoder time step i to produce the output $\dot{y}_{[i]}$ is given in Fig. 4.5, while the entire training process is defined in Algorithm 2. Let us denote BA variables with the b superscript notation. See that lines 12-20 of Algorithm 2 are very similar to lines 10-16 of Algorithm 1, since an identical encoder approach is taken, except in Algorithm 2 the encoder outputs are initialized and stored in lines 13 and 17 respectively.

The first step of BA is to compute the attention energies $e^b_{[ij]}$. In order to do so, T copies of $h^b_{[i-1]}$ are made, and this matrix is denoted as $H^b_{[i-1]} \in \mathbb{R}^{T \times h}$. As in lines 23-25 of Algorithm 2, the energies are computed by:

$$\lambda_1 = [H^b_{[i-1]}; \mathbf{H}] \in \mathbb{R}^{T \times 2h} \tag{4.5a}$$

$$\lambda_2 = \tanh(W^{2h \to h}(\lambda_1)) \in \mathbb{R}^{T \times h} \tag{4.5b}$$

$$\lambda_3 = \langle \lambda_2, v \rangle, \text{where } v \in \mathbb{R}^h \tag{4.5c}$$

$$e^b_{[ij]} = \lambda_3 \in \mathbb{R}^T \tag{4.5d}$$

Here, $W^{2h \to h}$ and v are parameters to be learned during training. $W^{2h \to h}$ is a fully-connected layer and v is a vector randomly initialized from $\mathcal{N}(0, \frac{1}{\sqrt{h}})$. From Fig. 4.5, the inner product has been taken with the result from the $\tanh(W^{2h \to h})$ block. Note that the $[\cdot]$ notation is used to indicate concatenation, while the λ notation in equation (4.5), and in subsequent equations, is simply used as a placeholder to denote the steps taken to produce that respective output. See that the inner product is taken in equation (4.5c), where λ_2 from equation (4.5b) is right multiplied with v. In equation (4.5c), λ_2 has dimensions $T \times h$ which is right multiplied by a vector of dimension h (or $h \times 1$), which results in a vector of dimension T. Hence, at decoder time step i, $e^b_{[ij]}$ is a vector of length T, where each element $j \in 1, ..., T$ represents the attention energy dedicated to that encoder output. The next step, line 26 of Algorithm 2, is to compute the attention weights $\alpha^b_{[ij]} \in \mathbb{R}^T$:

$$\alpha^b_{[ij]} = \frac{\exp(e^b_{[ij]})}{\sum_{k=1}^T \exp(e^b_{[ik]})} \tag{4.6}$$

This is the Softmax of the attention energies computed in equation (4.5). Observe that equation (4.6) is identical to equation (2.13). In Fig. 4.5, at this point of the process we have passed the “Softmax” block. Continuing, line 27 of Algorithm 2 shows that these attention weights are then used in an inner product with $\mathbf{H}$, to compute the context vector $c^b_{[i]} \in \mathbb{R}^h$, given

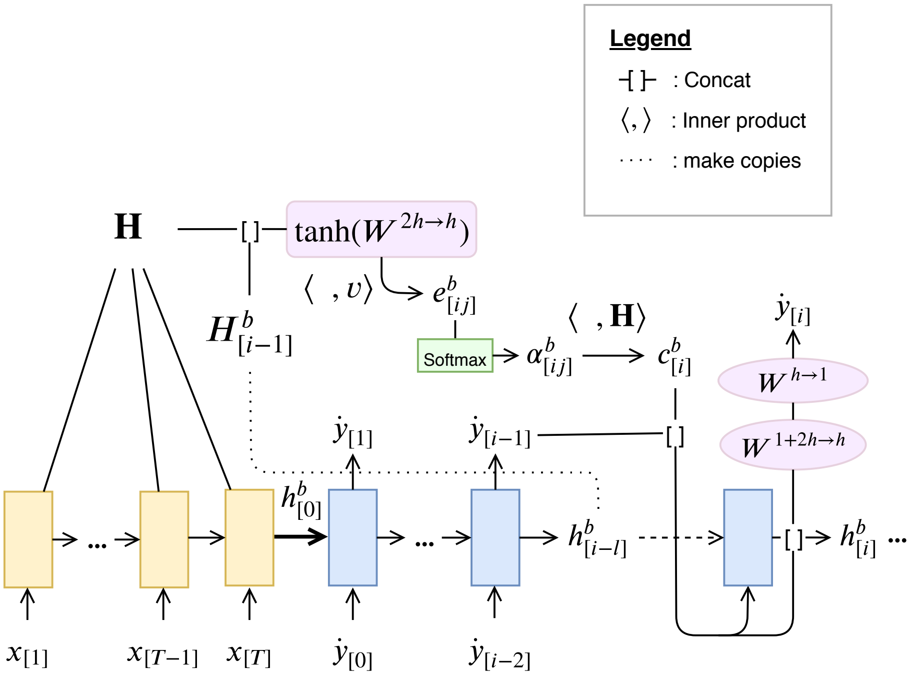


Figure 4.5: Detailed process of the BA mechanism at time step i.

as:

$$c^b_{[i]} = \langle \alpha^b_{[ij]}, \mathbf{H} \rangle \tag{4.7a}$$

$$= \sum_{j=1}^{T} \alpha^b_{[ij]} h_{[j]} \tag{4.7b}$$

The next step is to concatenate the context vector with the previously predicted output $\dot{y}_{[i-1]}$, and pass this newly formed vector through the respective RNN cell, to obtain the current hidden state $h^b_{[i]}$. In Algorithm 2, line 28 is given by the following steps:

$$\lambda_4 = [\dot{y}_{[i-1]}; c^b_{[i]}] \in \mathbb{R}^{1+h} \tag{4.8a}$$

$$\lambda_5 = \text{GRU}_{Cell}(\lambda_4, h^b_{[i-1]}) \tag{4.8b}$$

$$h^b_{[i]} = \lambda_5 \in \mathbb{R}^h \tag{4.8c}$$

Algorithm 2 S2S-BA$_{Train}$($G = (E_T, D_N, \tanh, \text{Softmax}, W^{2h \to h}, W^{1+2h \to h}, W^{h \to 1}, v, P_0)$)

```
Input : Model $G$ consisting of: $E_T$ of length $T$, $D_N$ of length $N$, activation functions
 tanh and Softmax, fully-connected layers $W^{2h \to h}$, $W^{1+2h \to h}$, $W^{h \to 1}$, vector parameter $v$,
 and initial weight parameters $P_0$.
Output : Model $G$ trained to convergence, with updated weight parameters $P$.

 Generate input samples $X \in \mathbb{R}^{T \times f}$ and corresponding target vectors $y \in \mathbb{R}^{N \times 1}$
 Initialize $v \sim \mathcal{N}(0, \frac{1}{\sqrt{h}}) \in \mathbb{R}^h$

 for each epoch do
   for each batch do

     initialize $h_{[j-1]} \leftarrow \vec{0} \in \mathbb{R}^{B \times h}$, where $B$ = batch size
     initialize $\mathbf{H} \leftarrow \vec{0} \in \mathbb{R}^{T \times B \times h}$

     for time step $j$ from 1 to $T$ do      # Encoder part
       $h_{[j]} \leftarrow \text{GRU}_{cell}(x_{[j]}, h_{[j-1]})$
       $\mathbf{H}_{[j]} \leftarrow h_{[j]}$

     $\dot{y}_{[0]} \leftarrow W^{h \to 1}(h_{[T]})$      # Obtaining initial inputs for Decoder
     $h^b_{[0]} \leftarrow h_{[T]}$

     for time step $i$ from 1 to $N$ do      # Decoder part
       $H^b_{[i-1]} \leftarrow T$ copies of $h^b_{[i-1]}$
       $e^b_{[ij]} \leftarrow \tanh(W^{2h \to h}([H^b_{[i-1]}; \mathbf{H}]))$
       $e^b_{[ij]} \leftarrow \langle e^b_{[ij]}, v \rangle^\intercal$      # Transpose to correspond with $B$ dimension
       $\alpha^b_{[ij]} \leftarrow \text{Softmax}(e^b_{[ij]})$
       $c^b_{[i]} \leftarrow \langle \alpha^b_{[ij]}, \mathbf{H}^\intercal \rangle$      # Transpose to correspond with $B$ dimension
       $h^b_{[i]} \leftarrow \text{GRU}_{Cell}([\dot{y}_{[i-1]}; c^b_{[i]}], h^b_{[i-1]})$
       $\dot{y}_{[i]} \leftarrow W^{h \to 1}(W^{1+2h \to h}([\dot{y}_{[i-1]}; c^b_{[i]}; h^b_{[i]}]))$
       append $\dot{y}_{[i]}$ to predicted usage vector $\dot{y}$

     compute $L$ as in equation (4.4)
     BPTT($L$)
     $P \leftarrow \text{Adam}(P_0, r)$, where $r$ = learning rate
     $P_0 \leftarrow P$

Return : Trained model $G$, with updated parameter weights $P$.
```

where the GRU$_{Cell}$ in equation (4.8b) takes λ_4 and the previous hidden state $h^b_{[i-1]}$ as inputs. See that the entire BA process was given using the GRU cell and differs slightly for the LSTM cell, since the LSTM cell outputs both a hidden state and a cell state. Nonetheless, only the hidden state that is output from the LSTM cell is used for the attention process. For the Vanilla RNN cell, the process would remain the same, since both GRU and Vanilla RNN cell only output a hidden state. The last step, in computing the predicted output $\dot{y}_{[i]}$, is to pass the current hidden state, the context vector, and the previous output through a function as given in equation (2.11). For this work, all three variables are concatenated, as seen to the right of the last decoder GRU cell in Fig. 4.5, and this vector is passed through two fully-connected layers. Hence:

$$\lambda_6 = [\dot{y}_{[i-1]}; c^b_{[i]}; h^b_{[i]}] \in \mathbb{R}^{1+2h} \tag{4.9a}$$

$$\lambda_7 = W^{1+2h \to h}(\lambda_6) \in \mathbb{R}^h \tag{4.9b}$$

$$\dot{y}_{[i]} = W^{h \to 1}(\lambda_7) \in \mathbb{R}^1 \tag{4.9c}$$

$$\tag{4.9d}$$

Here, $W^{1+2h \to h}$ and $W^{h \to 1}$ are parameters to be learned during training. As seen in Fig. 4.5 and line 29 in Algorithm 2, the predicted output is obtained following the steps in equation (4.9). Therefore, this concludes the decoder process for one time step. After N time steps, we have obtained our predicted usage vector $\dot{y} \in \mathbb{R}^N$, and can compute the loss as given in equation (4.4). The remaining steps, BPTT using the gradient based weights update, are identical as in section 4.3.

4.5 S2S-prediction with Luong Attention

This section describes the detailed process of applying LA to a S2S model to forecast building energy consumption. The process for time step i is shown in Fig. 4.6, while the entire training process is given in Algorithm 3. As in Section 4.4, the encoder takes an identical approach; lines 11-19 of Algorithm 3 are identical to lines 12-20 of Algorithm 2, respective to their own variables. Hence, at time step T, we will have obtained $h_{[T]}$, $\dot{y}_{[0]}$, and $\mathbf{H}$. Let us denote LA variables with the l superscript notation. The key difference between LA and BA; the **first step** in LA is to compute the current hidden state $h^l_{[i]} \in \mathbb{R}^h$, while the first step in BA is to make copies of the previous hidden state. Hence, the first step in LA, as given in line 22 of Algorithm 3:

$$h^l_{[i]} = \mathrm{GRU}_{Cell}(\dot{y}_{[i-1]}, h^l_{[i-1]}) \tag{4.10}$$

where equation (4.10) takes as inputs the previously predicted output and the previous hidden state. Note that the LA mechanism is applied using this current hidden state; counter to BA which used the previous hidden state. The next step is to compute the attention energies $e^l_{[ij]}$, which is done by equations (2.17) and (2.18). Note, from equation (2.18), that only one score function is chosen at a time, and this constitutes one model. The procedure to compute $e^l_{[ij]}$ using the *dot* score function is given as:

$$e^l_{[ij]} = \langle \mathbf{H}, h^l_{[i]} \rangle \in \mathbb{R}^T \tag{4.11}$$

Here, the dot product between $h_{[j]}$ and $h^l_{[i]}$ is computed for $j \in 1, ..., T$, and store the scalar value in the j-th element of $e^l_{[ij]}$. Instead, if the *general* score function is chosen from equation (2.18), as in line 23 of Algorithm 3, the attention energies are computed as:

$$\lambda_1 = W^{h \to h}(\mathbf{H}) \in \mathbb{R}^{T \times h} \tag{4.12a}$$

$$\lambda_2 = \langle \lambda_1, h^l_{[i]} \rangle \tag{4.12b}$$

$$e^l_{[ij]} = \lambda_2 \in \mathbb{R}^T \tag{4.12c}$$

Where $W^{h \to h}$ is a parameter to be learned during training. From Fig. 4.6, the inner product has been taken with the result from the $W^{h \to h}$ block. See that the *dot* and *general* score functions, as obtained in equations (4.11) and (4.12) respectively, are very similar. They differ only by a fully-connected layer, as used for the *general* score function, that first transforms **H** into a matrix with the exact same dimensions. Hence, the *general* score function can be seen as applying the *dot* score function to the matrix resulting from equation (4.12a). The remaining score function, *concat*, computes the energies similarly as in equation (4.5). However, the difference here is the use of matrix $H^l_{[i]} \in \mathbb{R}^{T \times h}$, T copies of $h^l_{[i]}$, instead of $H^b_{[i-1]}$.

Continuing, line 24 of Algorithm 3 shows that the attention weights $\alpha^l_{[ij]} \in \mathbb{R}^T$ are computed using equation (4.6), except here we use the attention energies $e^l_{[ij]}$ as obtained in equations (4.11) or (4.12). The attention context vector $c^l_{[i]} \in \mathbb{R}^h$ is then computed in line 25 of Algorithm 3 using the equations of (4.7) with the respective attention weights $\alpha^l_{[ij]}$. There is no difference between BA and LA in how the attention weights or attention context vectors are obtained *from* each mechanisms respective attention energies. This can be seen from figures 4.5 and 4.6, as the the attention energies take the same path to obtain the respective attention

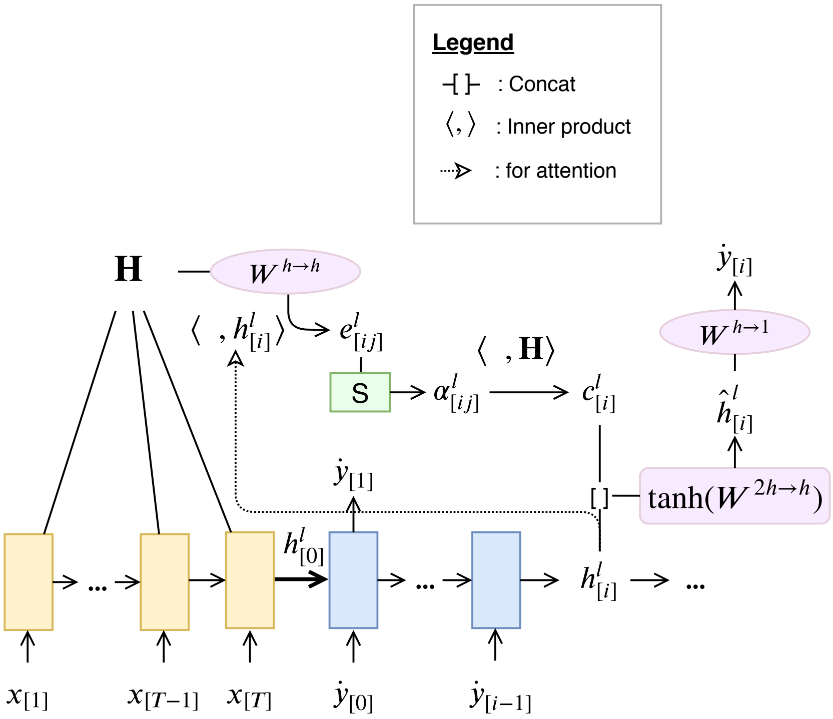


Figure 4.6: Detailed process of the LA mechanism at time step i, using the *general* score function.

context vectors. The next step, as in line 26 of Algorithm 3, is to compute the attentional hidden state $\hat{h}^l_{[i]}$:

$$\lambda_3 = [h^l_{[i]}; c^l_{[i]}] \in \mathbb{R}^{2h} \tag{4.13a}$$

$$\lambda_4 = \tanh(W^{2h\rightarrow h}(\lambda_3)) \tag{4.13b}$$

$$\hat{h}^l_{[i]} = \lambda_4 \in \mathbb{R}^h \tag{4.13c}$$

From Fig. 4.6, at this point of the process the result from the $\tanh(W^{2h\rightarrow h})$ block has been obtained. Lastly, the predicted output value is computed by passing $\hat{h}^l_{[i]}$ through a fully-connected layer:

$$\dot{y}_{[i]} = W^{h\rightarrow 1}(\hat{h}^l_{[i]}) \tag{4.14}$$

where $W^{2h\rightarrow h}$ and $W^{h\rightarrow 1}$ are parameters to be learned during training. Note that the attentional hidden state $\hat{h}^l_{[i]}$ and current hidden state $h^l_{[i]}$ have different functions in LA; $\hat{h}^l_{[i]}$ is only used in computing the output value, while $h^l_{[i]}$ is used in computing the attention context vector and is the hidden state to be used in the next time step. This concludes the LA mechanism for one time step. After N time steps, we have obtained the predicted usage vector, and the remaining steps are equivalent to that of Section 4.4.

Algorithm 3 S2S-LA-general$_{Train}(G = (E_T, D_N, \tanh, \text{Softmax}, W^{2h\to h}, W^{h\to h}, W^{h\to 1}, P_0))$

```
Input : Model $G$ consisting of: $E_T$ of length $T$, $D_N$ of length $N$, activation functions
 tanh and Softmax,fully-connected layers $W^{2h\to h}$, $W^{h\to h}$, $W^{h\to 1}$ and initial weight
 parameters denoted $P_0$.
Output : Model $G$ trained to convergence, with updated weight parameters $P$.

 Generate input samples $X \in \mathbb{R}^{T\times f}$ and corresponding target vectors $y \in \mathbb{R}^{N\times 1}$

 for each epoch do
   for each batch do

     initialize $h_{[j-1]} \leftarrow \vec{0} \in \mathbb{R}^{B\times h}$, $B$ = batch size
     initialize $\mathbf{H} \leftarrow \vec{0} \in \mathbb{R}^{T\times B\times h}$

     for time step $j$ from 1 to $T$ do      # Encoder part
       $h_{[j]} \leftarrow \text{GRU}_{Cell}(x_{[j]}, h_{[j-1]})$
       $\mathbf{H}_{[j]} \leftarrow h_{[j]}$

     $\dot{y}_{[0]} \leftarrow W^{h\to 1}(h_{[T]})$      # Obtaining initial inputs for Decoder
     $h^l_{[0]} \leftarrow h_{[T]}$

     for time step $i$ from 1 to $N$ do      # Decoder part
       $h^l_{[i]} \leftarrow \text{GRU}_{Cell}(\dot{y}_{[i-1]}, h^l_{[i-1]})$      # LA uses current hidden state, BA uses previous
       $e^l_{[ij]} \leftarrow \langle W^{h\to h}(\mathbf{H}), h^l_{[i]}\rangle^\intercal$      # Transpose to correspond with $B$ dimension
       $\alpha^l_{[ij]} \leftarrow \text{Softmax}(e^l_{[ij]})$
       $c^l_{[i]} \leftarrow \langle \alpha^l_{[ij]}, \mathbf{H}^\intercal\rangle$      # Transpose to correspond with $B$ dimension
       $\hat{h}^l_{[i]} \leftarrow \tanh(W^{2h\to h}(\text{cat}(h^l_{[i]}, c^l_{[i]})))$      # BA concatenates $\dot{y}_{[i-1]}$, LA does not
       $\dot{y}_{[i]} \leftarrow W^{h\to 1}(\hat{h}^l_{[i]})$  # LA computes $\dot{y}_{[i]}$ from attentional hidden state, BA does not
       append $\dot{y}_{[i]}$ to predicted usage vector $\dot{y}$

     compute $L$ as in equation (4.4)
     BPTT($L$)
     $P \leftarrow \text{Adam}(P_0, r)$, $r$ = learning rate
     $P_0 \leftarrow P$

Return : Trained model $G$, with updated parameter weights $P$.
```

Chapter 5

Evaluation

The utilized approach is evaluated on a real-world dataset that was provided by an industry partner. The dataset is based on approximately 1 year and 3 months of energy load data for one smart meter of one commercial building. The dataset, as obtained by the smart meter, contained readings with a timestamp (yyyy-mm-dd hh:mm:ss) and usage value in kW. The readings were in five minute increments, so the total number of readings was 12 readings in one hour × 24 hours in one day = 287 × 458 days = 131,446. As stated in Section 4.1, the smart meter data was processed into a dataset where each row consists of 9 features. The S2S and S2S attention-based models are evaluated for varying lengths of input and prediction sequences. In addition, the models are compared to a deep neural network, as well as a Non-S2S RNN. The rest of this section describes the conducted experiments, presents their results, and discusses the findings.

5.1 Experiments

All S2S models were tested for four different prediction lengths, which range from short to long. Each prediction length N is given as an element of the vector $\vec{N}$:

$$\vec{N} = [12,\ 48,\ 120,\ 288] \tag{5.1}$$

Hence $N_i \in \vec{N}$, where $i = 1,\ 2,\ 3,\ 4$. In 5 minute increments, this equates to predicting the next [1 hour, 4 hours, 10 hours, 24 hours] respectively. Since there are endless combinations of input length T to prediction length N, the following **four input cases** were chosen for experiments:

- **Case 1**: input $T = 12$, predict each $N_i \in \vec{N}$.
- **Case 2**: input $T = 48$, predict each $N_i \in \vec{N}$.
- **Case 3**: input $T = 120$, predict each $N_i \in \vec{N}$.
- **Case 4**: input $T = 288$, predict each $N_i \in \vec{N}$.

These cases equate to fixed input lengths of 1 hour, 4 hours, 10 hours, and 24 hours, respectively, to predict the next $N_i \in \vec{N}$ time steps. See that each of the four input lengths is used in computing four different prediction lengths, hence a total of 16 different combinations/subcases are evaluated. Note that the longest prediction length is 24 hours, or 288 time steps, for five-minute data intervals. With datasets consisting of one-hour increments, 288 time steps would translate to predicting 288 hours (12 days) consecutively. All models were trained for no longer than 10 epochs, since this was sufficient to reach an acceptable level of convergence. To support this claim, figures throughout each case are provided to show that the total loss, for both the training and test set, did not exhibit signs of improvement as the 10th epoch approached. The hyperparameters used to compute the results were:

- Hidden dimension size $h = [64,\ 128]$
- Cell state dimension size $c = [64,\ 128]$ (LSTM)
- Batch size $B = 256$
- Learning rate = 0.001

Note that only one value of h, and equivalent dimension c for LSTM, is used for each experiment. Two sizes of h were considered to see if increasing the hidden state, hence adding more parameters, would improve the accuracy. A large grid search holds potential to optimize hyperparameters and further improve results. Each model was compared against the other models; overall a total of 21 different models were used for each of the experiments:

- S2S-o model with GRU/LSTM/RNN cell
  - Three models in total corresponding to three cell types
- S2S-BA model with GRU/LSTM/RNN cell
  - Three models in total corresponding to three cell types
- S2S-LA model with GRU/LSTM/RNN cell
  - Accompanied with each of three cells is one of three attention score functions: *dot, general, concat*
  - Nine models in total
- Non-S2S plain RNN model with GRU/LSTM/RNN cell
  - Three models in total corresponding to three cell types
- DNN model with varying sizes: small, medium, large
  - Three models in total

It is important to note that we define a **model** as a network with one of the 21 general architectures as listed above. However, the specific architecture does change for one of 16 subcases of T and N: four input cases that each predict four lengths. A trained model within a specific subcase will be unique to that subcase; it is technically different than the same model for a different subcase, although the algorithms are equivalent. For example, the GRU based S2S-BA model with $T = 48$ and $N = 12$ is ultimately different than the GRU based S2S-BA model with $T = 120$ and $N = 288$. Hence, these subcase specific models branch from the parent **model** architecture since they undergo the same algorithm, but are technically different as each specific subcase model uses their respective number of input and prediction steps.

The Non-S2S model is a Non-S2S RNN model, such as described in Section 2.1. This model does not use an encoder-decoder framework, and thus can not have a prediction length longer than input sequence length. Hence, this model is only used when $N <= T$. For example, for case 3 where input length is $T = 120$, the Non-S2S model can only predict $N_i \in \vec{N}$ for $i = 1, 2, 3$, and is omitted from results where $T = 120$ and $N = 288$.

The DNN model took the same input matrix $X \in \mathbb{R}^{T \times f}$ as was used with all other models, but this matrix was flattened it into a single vector of dimension size $= T \times f$. Hence, each input vector $x_{[j]}$ of X, as in 4.2, was now side-by-side constituting one long vector. The first f elements of this long vector are the first row of X, the second f elements is the second row, and so-forth. This vector was then passed through three different DNN models: DNN-small, DNN-medium, and DNN-large. The output was a layer consisting of $N_i \in \vec{N}$ nodes, which is equivalent to length N_i of the predicted usage sequence used in the S2S models. The three sizes of DNN models are:

- DNN-small: Input layer (size $T \times f$) $\bowtie$ 512 $\bowtie$ 256 $\bowtie$ 128 $\bowtie$ Output layer (size N_i)
- DNN-medium: Input layer $\bowtie$ $512^{\bowtie 3}$ $\bowtie$ 256 $\bowtie$ 128 $\bowtie$ Output layer
- DNN-large: Input layer $\bowtie$ $1024^{\bowtie 2}$ $\bowtie$ $512^{\bowtie 3}$ $\bowtie$ $256^{\bowtie 2}$ $\bowtie$ Output layer

Where the notation $\bowtie$ is used to indicate the joining of the previous layer to the next, and $A^{\bowtie B}$ notation used to indicate joining B layers of size A. Hence, $512^{\bowtie 3}$ indicates three fully-connected layers of size 512. Multiple layers of the same size were used to add more parameters, to see whether increasing parameters improved the accuracy. The DNN models are used to directly predict the sequence of values, all at once, while the S2S models sequentially predict the values one at a time. Thus, the DNN results give a good comparison between direct prediction accuracy versus sequential prediction accuracy. The activation function used for the DNN output layer was linear, since this is a regression problem.

See that the S2S models were compared against two baseline models, the Non-S2S RNN and the DNN. In addition, the moving average was also used as a baseline during experiments but the results were left out since these models did not achieve good accuracy and would impair visualization of figures and tables. For S2S models, as stated in the sample generation subsection, there was no overlap in target test samples to enable concatenation of predicted usage vectors into one vector. Hence, the accuracy measures were obtained by comparing this one overall predicted usage vector against the overall actual usage vector. The accuracy measures used throughout this work were the Mean Absolute Error (MAE) and Mean Absolute Percentage Error (MAPE), given by the following equations:

$$\mathrm{MAE} = \frac{1}{n}\sum_{i=0}^{n} |y_i - \hat{y}_i| \tag{5.2}$$

$$\mathrm{MAPE} = \frac{100\%}{n}\sum_{i=0}^{n} \left|\frac{y_i - \hat{y}_i}{y_i}\right| \tag{5.3}$$

Where y represents the actual value, $\hat{y}$ the predicted value, and number of observations or samples is given by n. Note that the MAE and MAPE were calculated with the unnormalized predicted values, meaning once the overall predicted usage vector was obtained in the normalized space, it was converted to the original domain using:

$$\hat{y} = (\hat{y} \times \sigma_{usage}) + \mu_{usage} \tag{5.4}$$

Where σ_{usage} and μ_{usage} was calculated as in equation (4.1) for the usage feature vector, and $\hat{y}$ on the right-hand side of (5.4) is the predicted vector in the normalized space.

Since this work randomized the training samples, it was important to run a several simulations where each model sees a different randomized order of training samples each time. Hence, 5 random seeds were used for each case. For example, if random seed equals 1, then the training samples are in one randomized order, and that order is used by each model, for that case. If random seed equals 2, the training samples are in a different randomized order. A value (such as 1 or 2) is assigned to the seed to keep track of the random order, to later reproduce identical results.

All models had their matrix weights initialized with a random (semi) orthogonal matrix, and their bias vector weights initialized to zero. The work by Saxe *et al*. [64] showed that these random orthogonal initial weights are favourable for depth independent learning times, and also lead to faithful propagation of gradients even in deep nonlinear networks.

5.2 Results and Discussion

The applied approach was implemented using Python and the PyTorch optimized tensor library [65], which is an open-source machine learning library. Experiments were conducted using two different workstations: the first contains two NVIDIA GeForce RTX 2080 Ti GPU cards, and the second contains one NVIDIA GeForce GTX 1060 GPU card. The following four subsections present results for the four input length cases as outlined in Section 5.1. Subsection 5.2.1 analyzes case 1 with three figures: cell performance comparison, best models achieved, and total loss computed. Subsection 5.2.2 analyzes case 2 with the same three figures, only the results are respective to case 2, while subsections 5.2.3 and 5.2.4 considers the results for cases 3 and 4 respectively. The aforementioned subsections contain discussions that accompany each figure. The last subsection, 5.2.5, provides results on how each model performed while the prediction length N was fixed, and input length T varied instead.

The first figure, in each of the four sections corresponding to four cases, is given to analyze the performance of cells used in the models. Each S2S and Non-S2S RNN model used one of: Vanilla RNN, GRU, or LSTM cell. This result is shown to compare cell performance as input length T is kept stationary while prediction length N varies. Note that the Non-S2S RNN models will only show results for when $N <= T$, hence for the first figure of each subsection, if $T > N$, no results will appear for the Non-S2S RNN model. In addition, the DNN model results were not used in this first figure, since it does not deal with RNN cell variation and performance comparison; it would show the same results for each cell (different MAPE and MAE).

The second figure in each case subsection is used to show the best achieved results by each model. This result is shown to compare the performance of each model, regardless of which cell or hidden dimension size h was used. Note for these second figures, the best obtained MAPE among all models, for each N, is bolded. This is done to identify the best performing models and to expand on their results further - as provided in the third figure, where each row of graphs represents the best achieved MAPE for that prediction length N. In this way, the second and third figure of each case subsection are interconnected. This third figure is given to analyze the performance of the best models, in terms of how well it fits to the actual data and whether or not the model is overfitting/underfitting.

The discussions in each subsection center around the MAPE accuracy measure. For the most part, the MAPE and MAE graphs show identical results, hence commenting on the MAE graphs is omitted, since it would repeat the statements given for the MAPE graphs. Nevertheless, figures include results for both MAPE and MAE.

5.2.1 Case 1: Input One Hour

This subsection analyzes the results obtained for case 1, where input length is fixed at $T = 12$ and the models predict each $N_i \in \vec{N}$. From Fig. 5.1, it can be seen that the accuracy decreases across all cells as prediction length N increases. While accuracy does decrease, it is interesting to see that the Vanilla RNN cell achieves comparable results with two attention models: S2S-LA-general and S2S-LA-concat. In addition, for the Vanilla RNN cell, every attention model outperformed the S2S-o model for $N = 120$ and $N = 288$. Note that the Non-S2S RNN model produced results only for $N = 12$, and for each cell, this model produced the worst results among all models.

Furthermore, the MAPE and MAE for the GRU cell is relatively similar among all models across $N_i \in \vec{N}$, with the majority of attention models performing better than the S2S-o model

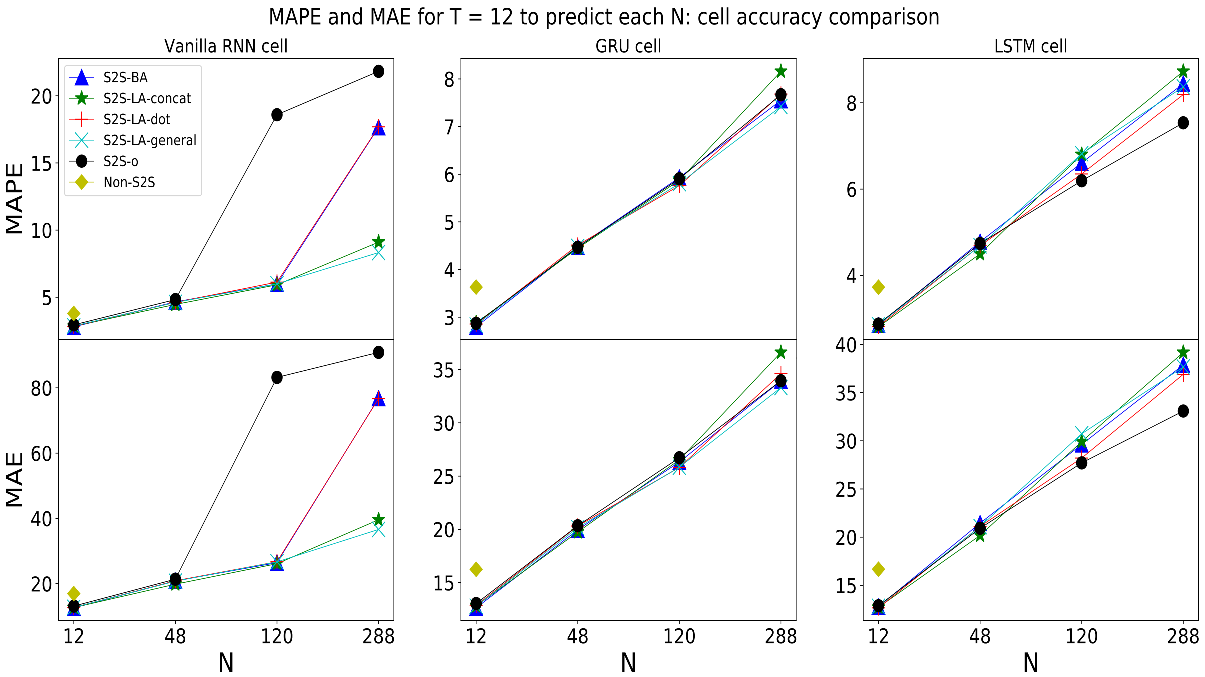


Figure 5.1: Comparing cell performance of each non-DNN model: input $T = 12$ to predict $N_i \in \vec{N}$. First row graphs show MAPE, while bottom row shows MAE.

for $N = 288$. However, this is not the case for the LSTM cell, as the S2S-o model outperforms the attention models for $N = 120$ and $N = 288$. See that the DNN model results were not used in Fig. 5.1, since it does not deal with RNN cell variation and performance comparison; it would show the same results for each cell (different MAPE and MAE).

The following figure, Fig. 5.2, shows the best achieved MAPE and MAE by each model, for each prediction length $N_i \in \vec{N}$. Again, the Non-S2S model only has a result for $N = 12$, and is the worst performing model from all the models, even the DNN model achieved better results. Furthermore, it can be seen that as N increases, the accuracy of each model decreases as well. The accuracy of the DNN model decreases at a much greater rate than the other models.

It is interesting to see that the S2S-o models perform comparable to the attention models. We assumed that the attention mechanism would be an advantage for the S2S models, however, a short input length such as $T = 12$ produces only 12 encoder outputs, and this might not contribute enough to these attention models. Nonetheless, the attention models obtained the

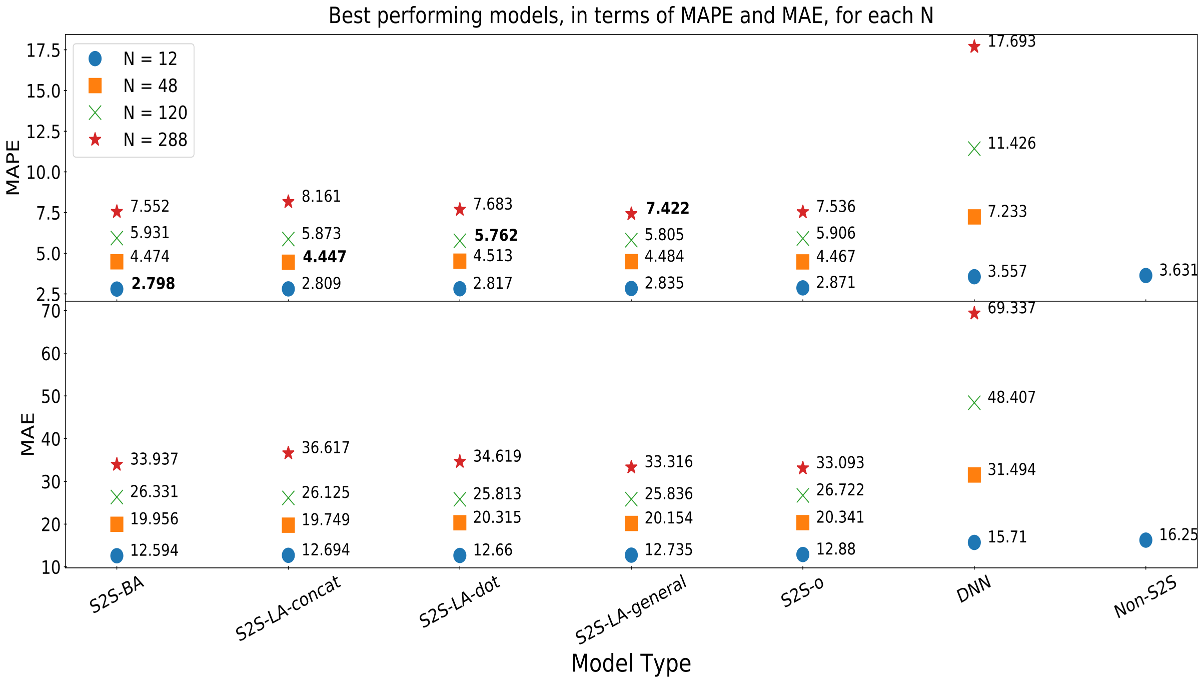


Figure 5.2: Best achieved results, by each model: input $T = 12$ to predict $N_i \in \vec{N}$. First row graph shows MAPE, bottom row shows MAE.

best MAPE for each $N_i \in \vec{N}$, as can be seen by the bolded numbers in the top row of Fig. 5.2.

These best performing models are analyzed further in Fig. 5.3. The predicted vs. actual (PvA) graph in each row shows that the models are capable of learning the trend even from a short input length, such as one hour. From these PvA graphs, it can be seen that the models do not fit perfectly to the actual data, but rather they are able to predict the general trend very well. This is an indicator that the model is not overfitting. This statement is further supported by the respective training and testing loss graphs of (a), (b), (c) in Fig. 5.3. These three models show that as the loss decreases for the training set, it also decreases for the test set, which is an indicator that a model is not overfitting.

However, the test loss graph of Fig. 5.3 (d) does raise some caution as the loss is not continuously decreasing. By the last output sample of the PvA graph of Fig. 5.3 (d), it can be seen that the predicted does not match the actual as it does for shorter lengths. It is known that the accuracy does decrease as prediction length N increases, hence, this may be a reason for the

S2S-LA-general model fitting worse for $N = 288$ than the other models for shorter prediction lengths.

5.2.2 Case 2: Input Four Hours

This subsection analyzes the results obtained for case 2, where input length is fixed at $T = 48$ and the models predict each $N_i \in \vec{N}$. From Fig. 5.4, it can be seen that, for all cells, the Non-S2S RNN model performs the worst when $N <= T$. Similarly as in Fig. 5.1, the Vanilla RNN cell shows comparable results for two of the attention models, here S2S-LA-dot and S2S-LA-concat, for $N = 120$ and $N = 288$. However, Fig. 5.1 shows the S2S-LA-general model as the dominant model for $N = 288$, for the Vanilla RNN cell, over other attention based models. This observation is switched in Fig. 5.4, as the S2S-LA-dot model shows better accuracy over

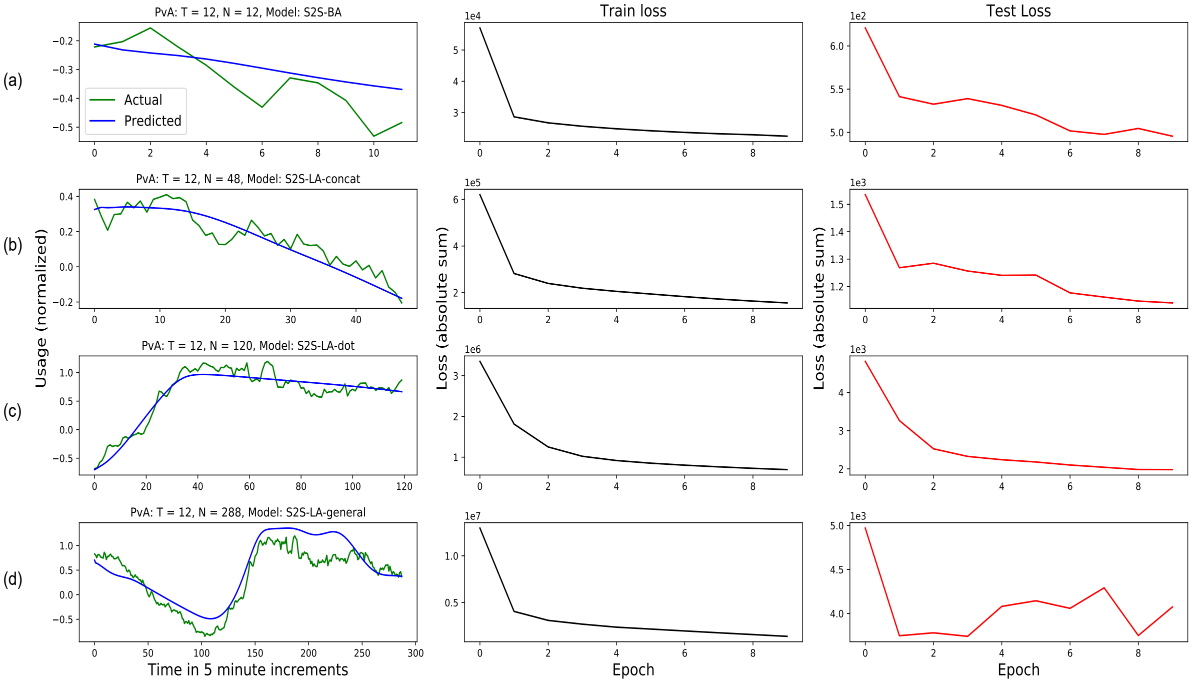


Figure 5.3: (a) Results for the S2S-BA model where $T = 12,\ N = 12$. From left to right: Predicted vs. Actual of the last output sample, total train loss and total test loss computed at each epoch, respectively. (b) Results for the S2S-LA-concat model where $T = 12,\ N = 48$. (c) Results for the S2S-LA-dot model where $T = 12,\ N = 120$. (d) Results for the S2S-LA-general model where $T = 12,\ N = 288$.

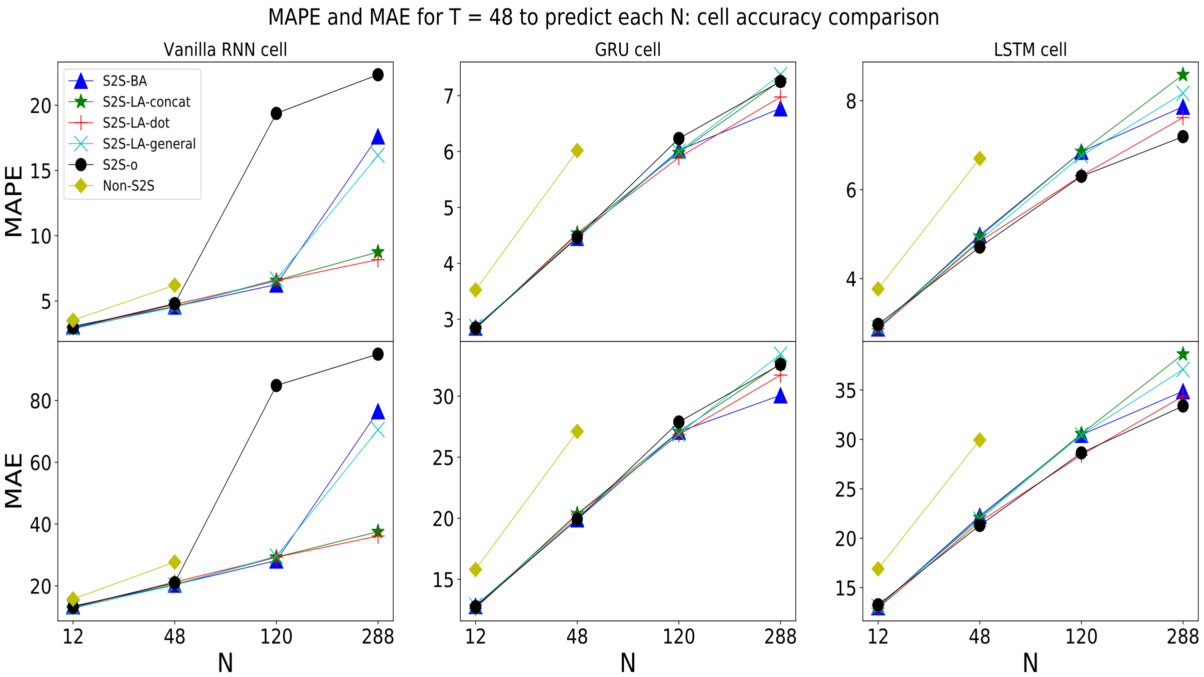


Figure 5.4: Comparing cell performance of each non-DNN model: input $T = 48$ to predict $N_i \in \vec{N}$. First row graphs show MAPE, while bottom row shows MAE.

the S2S-LA-general model, for the Vanilla RNN cell.

Furthermore, the GRU cell results in Fig. 5.4 show the majority of attention models outperforming the S2S-o model as N increases. As this result holds for both the Vanilla RNN and GRU cell, it is interesting to see that the opposite happens for the LSTM cell. Namely, Fig. 5.4 shows the S2S-o model outperforming the attention models as N increases. One possible hypothesis as to why this may be happening: the attention models “confuse themselves” since they use the encoder outputs in determining the next decoder output. Namely, the LSTM cell contains a hidden state, which are the encoder outputs after each cell operation, and a cell state, which is **not** used in the attention mechanism. Hence, the use of only one internal state, from the LSTM cell, in determining the decoder output for attention models could be the underlying reason as to why the S2S-o models outperformed the attention models, for increasing N.

Fig. 5.5 gives the best results achieved by each model, not taking into account the cell used or hidden dimension size, for each $N_i \in \vec{N}$. Similarly as in Fig. 5.2, the Non-S2S RNN model

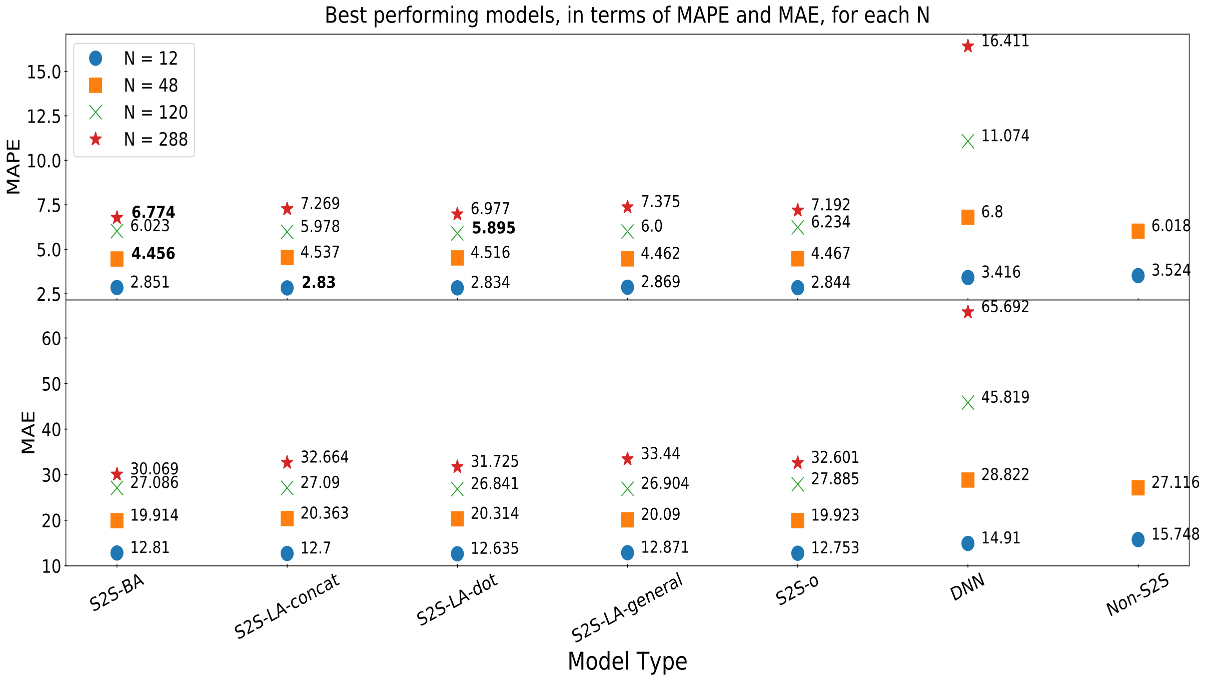


Figure 5.5: Best achieved results, by each model: input $T = 48$ to predict $N_i \in \vec{N}$. First row graph shows MAPE, bottom row shows MAE.

performs the worst for $N = 12$, but does perform better than the DNN model for $N = 48$. It is interesting to see that the S2S-o model shows comparable results to the attention models for $N = 12$ and $N = 48$. Ultimately, a similar outcome is obtained as in case 1, where the best achieved MAPE for each $N_i \in \vec{N}$ was obtained by one of the attention models, as can be seen by the bolded values in Fig. 5.5.

Comparing the DNN model results between Fig. 5.2 and Fig. 5.5, it can be seen that the accuracy, both MAPE and MAE, slightly increases as input length T increases. Nonetheless, as N increases in Fig. 5.5, the DNN models are significantly outperformed by all S2S models for $N = 120$ and $N = 288$.

Continuing, Fig. 5.6 analyzes results from the best models as obtained in Fig. 5.5. All PvA graphs of Fig. 5.6 show that the models are able to learn and predict the general trend of usage data. As in Fig. 5.3, this is an indicator that the models are not overfitting. Supporting this statement, the train and test loss graphs, in Fig. 5.6 (a), (b), and (d), show continuously

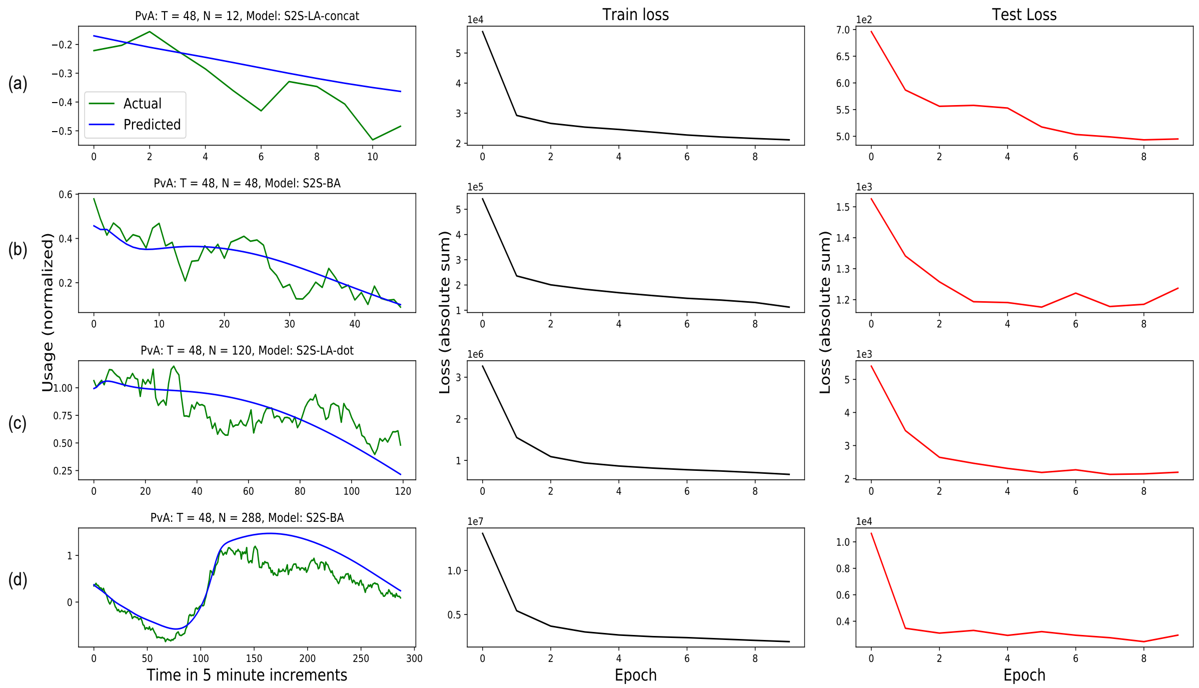


Figure 5.6: (a) Results for the S2S-LA-concat model where $T = 48,\ N = 12$. (b) Results for the S2S-BA model where $T = 48,\ N = 48$. (c) Results for the S2S-LA-dot model where $T = 48,\ N = 120$. (d) Results for the S2S-BA model where $T = 48,\ N = 288$.

decreasing absolute loss for both the training and testing sets. Only the S2S-BA model for $N = 48$, Fig. 5.6 (b), shows a slight sign of overfitting, as the very last epoch in the test loss graph shows as increase in loss rather than a decrease.

5.2.3 Case 3: Input Ten Hours

This subsection provides the results obtained for case 3, where input length is fixed at $T = 120$ and the models predict each $N_i \in \vec{N}$. From Fig. 5.7, it can be observed that the Non-S2S RNN model with the Vanilla RNN cell obtains better results than the S2S-o model for $N = 120$. One possible reason for this outcome is that the S2S-o model with Vanilla RNN cell loses more information as N increases, than the Non-S2S RNN model. Hence, the use of a Vanilla RNN cell in the decoder part of the S2S-o model does not retain information better than the one encoder in the Non-S2S RNN model.

For the GRU and LSTM-based models, the Non-S2S RNN model performs the worst for

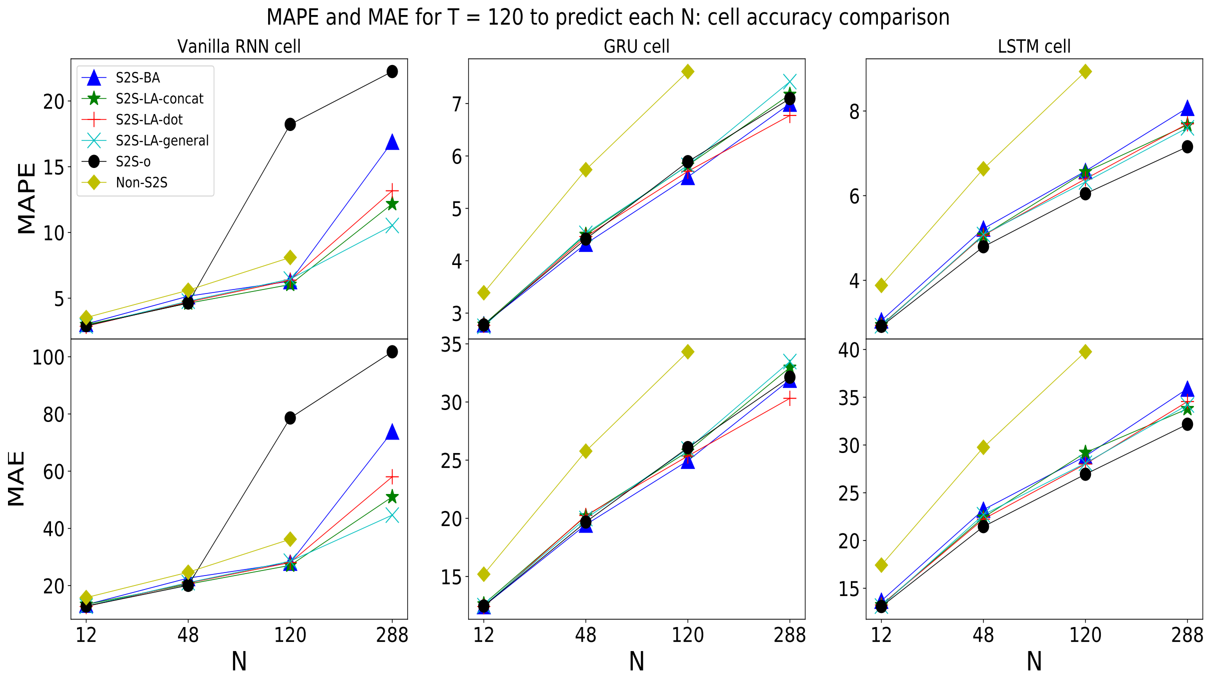


Figure 5.7: Comparing cell performance of each non-DNN model: input $T = 120$ to predict $N_i \in \vec{N}$. First row graphs show MAPE, while bottom row shows MAE.

all N except for $N = 288$, as there is no result for it in case 3, since $N = 288 > T = 120$. From Fig. 5.7, the Vanilla RNN-based S2S attention models outperformed the S2S-o model for $N = 120$ and $N = 288$, as were similar findings from figures 5.1 and 5.4. In addition, the GRU and LSTM cell results from Fig. 5.7 are similar to that of figures 5.1 and 5.4. Hence, the majority of GRU-based attention models outperform the S2S-o model as N increases, while the LSTM-based S2S-o model still outperforms the attention models for $N_i \in \vec{N}$. A possible reason as to why this may be the case is given in subsection 5.2.2.

The next figure, Fig. 5.8, shows the best achieved MAPE and MAE by each model for $N_i \in \vec{N}$, regardless of which cell or hidden dimension size was used. A re-occurring pattern is seen from figures 5.8, 5.5, and 5.2, where the attention models obtain the best MAPE across all prediction lengths, as given by the bolded values. From Fig. 5.8, it can be seen that the DNN model obtains the worst MAPE for $N > 12$. Nonetheless, from figures 5.8, 5.5, and 5.2, the DNN model improves as input length T increases, hence as we move from case 1 to case 3.

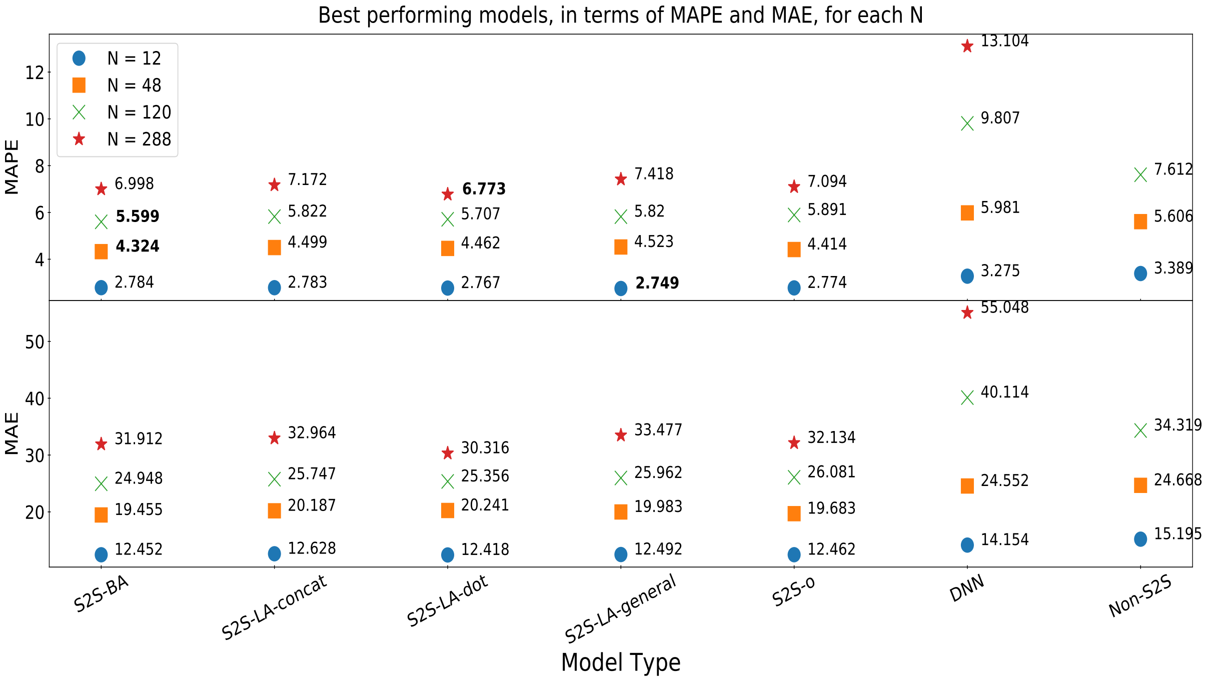


Figure 5.8: Best achieved results, by each model: input $T = 120$ to predict $N_i \in \vec{N}$. First row graph shows MAPE, bottom row shows MAE.

Fig. 5.9 provides more insight into the best achieved models as obtained in Fig. 5.8. Similarly as in figures 5.3 and 5.6, the PvA graphs of Fig. 5.9 show that the models are able to capture the general trend of data, and thus predict this future general trend rather than perfectly fitting to the data. From the test loss graphs of Fig. 5.9 (a) and (c), these models do not show signs of overfitting, while the models of Fig. 5.9 (b) and (d) do raise some caution. One possible way to address this: reduce the learning rate or have it decay after a certain number of epochs. The learning rate was fixed at 0.001 for all models, hence there is potential to improve results by additional tuning.

5.2.4 Case 4: Input Twenty-Four Hours

This subsection presents the results obtained for case 4, where input length is fixed at $T = 288$ and the models predict each $N_i \in \vec{N}$. Fig. 5.10, shows the GRU and LSTM-based Non-S2S RNN models performed the worst for all prediction lengths. This was not the case for the

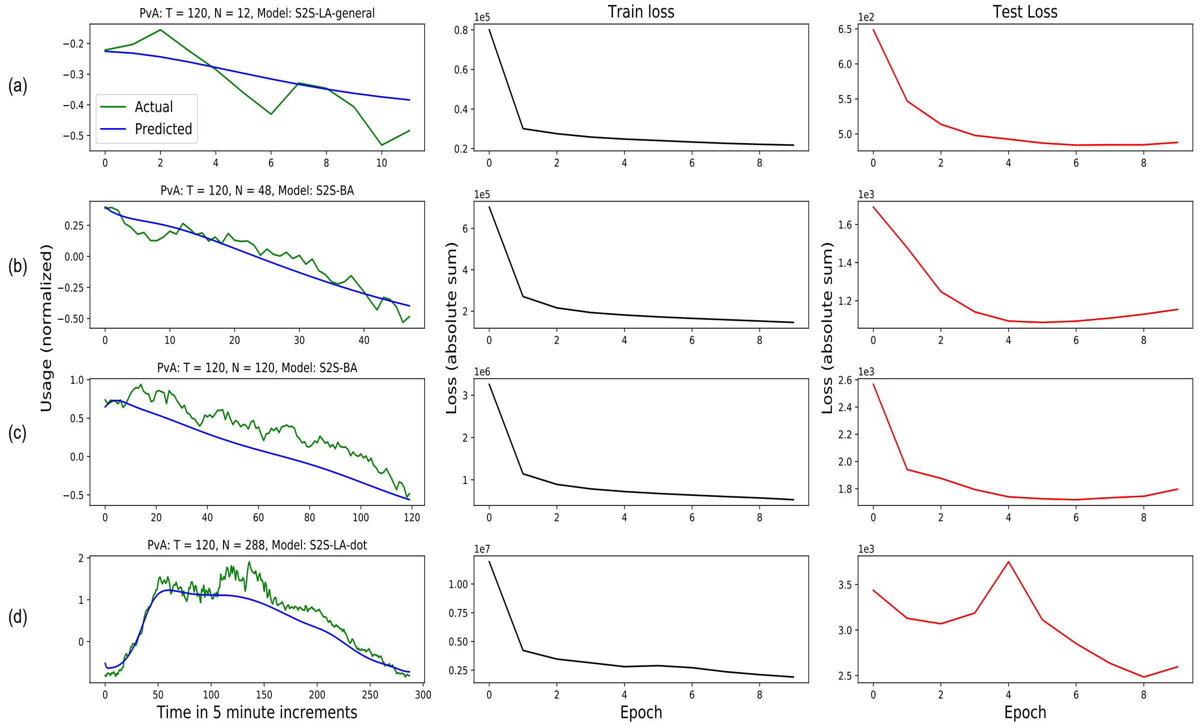


Figure 5.9: (a) Results for the S2S-LA-general model where $T = 120,\ N = 12$. (b) Results for the S2S-BA model where $T = 120,\ N = 48$. (c) Results for the S2S-BA model where $T = 120,\ N = 120$. (d) Results for the S2S-LA-dot model where $T = 120,\ N = 288$.

Vanilla RNN cell-based models, as the Non-S2S RNN model performed the best for $N = 288$. A similar hypothesis as stated in subsection 5.2.3 can be give here as well. Namely, the combination of an Encoder-Decoder + Vanilla RNN cell does not retain information better than a model consisting of a sole Encoder + Vanilla RNN cell. Like figures 5.1, 5.4, and 5.7, the LSTM-based S2S-o model in Fig. 5.10 outperformed all other models for $N = 288$. It is interesting to see that, unlike figures 5.1, 5.4, and 5.7, Fig. 5.10 shows that the GRU-based S2S-o models outperformed all other models for $N = 288$. One hypothesis for this outcome: the attention models are “confusing themselves” by trying to “pay attention” to too much of the input sequence. Hence, for case 4, all 288 encoder outputs contribute to each decoder output, which may be excessive and causes a loss in accuracy rather than improvement.

The following figure, Fig. 5.11, gives the best MAPE and MAE achieved by each model, regardless of which cell was used or hidden dimension size. See that the DNN model achieved

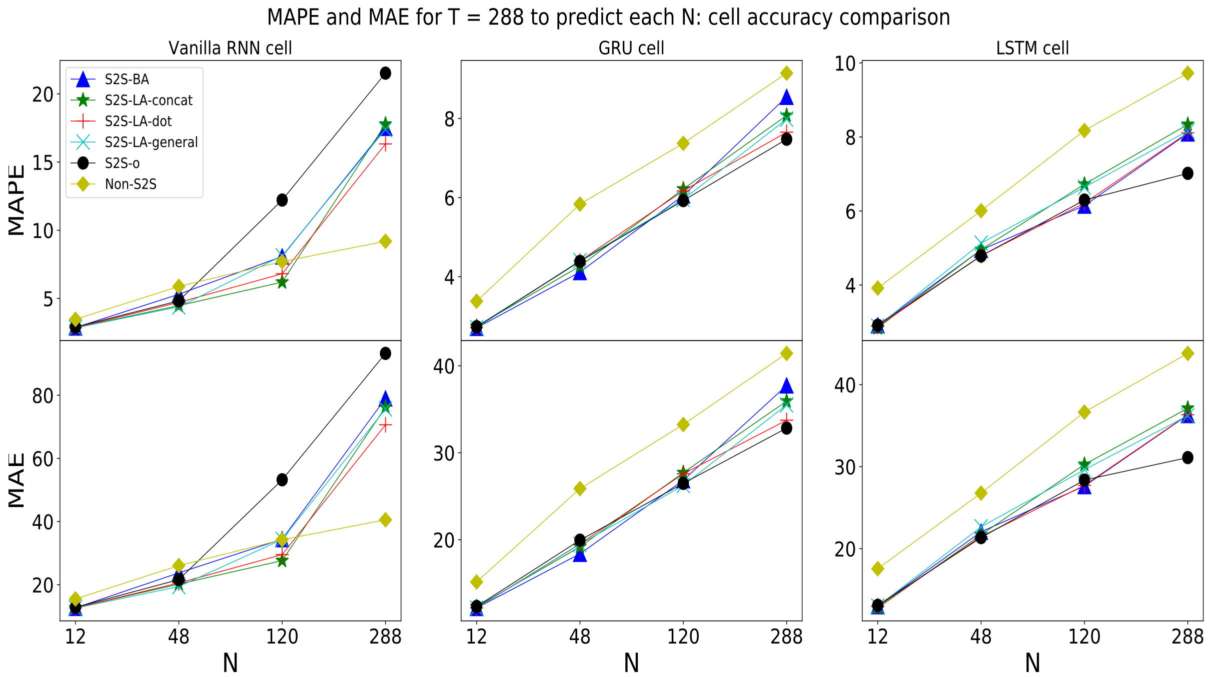


Figure 5.10: Comparing cell performance of each non-DNN model: input $T = 288$ to predict $N_i \in \vec{N}$. First row graphs show MAPE, while bottom row shows MAE.

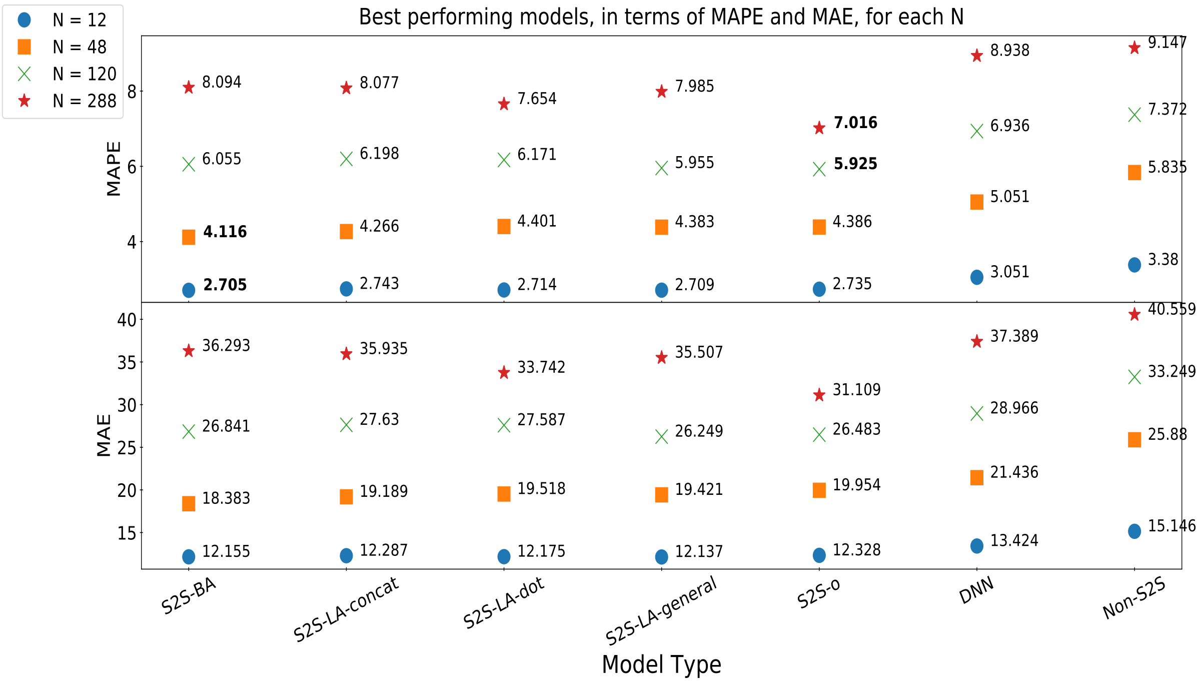


Figure 5.11: Best achieved results, by each model: input $T = 288$ to predict $N_i \in \vec{N}$. First row graph shows MAPE, bottom row shows MAE.

worse results than the S2S models, but did outperform the Non-S2S RNN model for each $N_i \in \vec{N}$. Nonetheless, the DNN model results in Fig. 5.11 are a significant improvement from the figures 5.2, 5.5, and 5.8. However, unlike the previous second figures of each subsection, the S2S-o model achieved the best MAPE for $N = 120$ and $N = 288$. The reasoning for this can be given as stated in the previous paragraph, the attention models lose accuracy for using too much information from the input sequence.

Lastly, Fig. 5.12 provides an analysis of the best models as obtained in Fig. 5.11. The PvA graphs in Fig. 5.12 are similar to the PvA graphs in figures 5.3, 5.6, and 5.9. Hence, we can conclude that all S2S models, with and without attention, are able to predict the future general trend of the usage data. This statement is an indication that these S2S models do not experience overfitting. Furthermore, the test loss graphs of Fig. 5.12 confirm the best models, as obtained from Fig. 5.11, do not overfit.

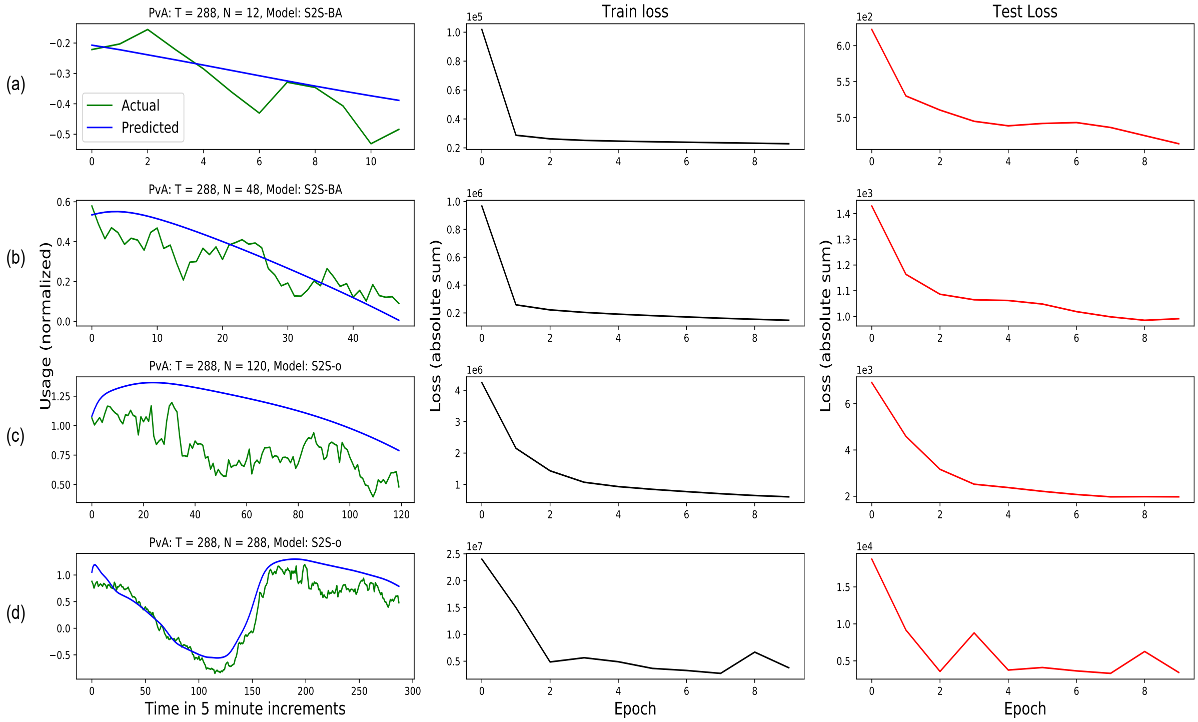


Figure 5.12: (a) Results for the S2S-BA model where $T = 288,\ N = 12$. (b) Results for the S2S-BA model where $T = 288,\ N = 48$. (c) Results for the S2S-o model where $T = 288,\ N = 120$. (d) Results for the S2S-o model where $T = 288,\ N = 288$.

5.2.5 Discussion

The previous subsections of this chapter analyzed model behaviour for a fixed input length T to predict each $N_i \in \vec{N}$. This subsection analyzes model results for varying input length T, hence across input time, as the prediction length increases. This result is shown to determine how each model performed with different input lengths. Meaning, one specific model produced four different results at each $N_i \in \vec{N}$, since it saw four different input lengths for that respective N. For example, the S2S-BA model produced four different results for $N = 12$: S2S-BA for $T = 12$, $T = 48$, $T = 120$, and $T = 288$. Note that for these results, the Non-S2S RNN model produced four models for $N = 12$, three models for $N = 48$, two for $N = 120$, and only one model for $N = 288$. Hence, the Non-S2S RNN model was not included in this analysis, since analysis could not be performed over all prediction lengths.

From Fig. 5.13, it can be seen that, for the longest prediction length $N = 288$, the best model versions are obtained for input lengths either $T = 48$ or $T = 120$. Looking at the S2S-BA model in Fig. 5.13 (a), when input length is $T = 288$, this version achieved the best results for $N = 12$ and $N = 48$; a result that is shared with every other model. However, the S2S-BA model, when input length is $T = 288$, achieved the worst results for $N = 120$ and $N = 288$; a result that is not shared with any other model. The S2S-BA$_{T=48}$ model achieved the best MAPE for $N = 288$; similar to the S2S-LA-general$_{T=48}$ model. Note the subscript notation indicates the respective model version for that input length. The S2S-LA-dot$_{T=120}$ model achieved the best results for $N = 120$, $N = 288$; similar to the S2S-LA-concat$_{T=120}$ model.

Mostly, the S2S attention models share a similar pattern: versions with $T = 288$ achieve the best results for $N <= 48$, while versions for $T < 288$ achieve the best results for $N >= 120$. One possible reason for this, as given in the first paragraph of subsection 5.2.4, the attention models "confuse themselves" trying to "pay attention" to an entire previous day in the latter end of the decoder part. Hence, as the decoder approaches the end of a long prediction sequence, looking at the entire previous day does not seem necessary in determining the next predicted value. For example, the process to predict the last output value at $n = 288$ incorporates the very first encoder output at $t = 1$, which is 576 (288×2) time steps in the past, nearly 48 hours previous. Therefore, the aforementioned reason supports plausibility as to why the attention models with $T = 288$ performed worse for $N >= 120$, than models with input length $T < 288$.

Furthermore, the above reasoning is weakly confirmed by analyzing the non-attention S2S-o model results in Fig. 5.13 (a). As can be seen, the best S2S-o models for $N = 288$ are, in order: S2S-$o_{T=288}$, S2S-$o_{T=120}$, S2S-$o_{T=48}$, S2S-$o_{T=12}$. This result is what was expected of all models. Nonetheless, the S2S-o models do not use an attention mechanism; each predicted output is computed using the previously predicted output and previous decoder hidden state. This allows no "confusion" between a long encoder and decoder, thus weakly supporting the above reasoning.

Lastly, The DNN model in Fig. 5.13 (a) shows the clearest signs of improvement over input length versions T. Hence, the longer the input sequence, the better the results obtained for each $N_i \in \vec{N}$. The $\text{DNN}_{T=288}$ model achieved the best results across all prediction lengths, with significant improvement from the $\text{DNN}_{T=12}$ model as N increased.

Overall, it can be concluded that as the prediction length N increases, the accuracy in both MAPE and MAE decreases. This can be observed from Table 5.1, which is given to summarize all results and identify the overall best performing model. The Table provides the best achieved MAPE and MAE for each model, as well as the standard deviation of the Absolute Percentage Error (APE) and Absolute Error (AE) respectively. The standard deviation-APE (SD-APE) was calculated as:

$$\vec{\text{APE}} = 100\% * \left| (\vec{actual} - \vec{predicted}) / \vec{predicted} \right| \tag{5.5a}$$

$$\text{SD-APE} = \sqrt{\frac{1}{n} \sum_{i=0}^{n} (\vec{\text{APE}}_i - \mu)} \tag{5.5b}$$

where equation (5.5a) computes the APE of the actual and predicted vectors, then equation (5.5b) computes the standard deviation of this APE vector. In equation (5.5b), n and μ are the number of elements and the mean of $\vec{\text{APE}}$ respectively, hence μ is simply the MAPE. Similarly, the standard deviation-AE (SD-AE) was calculated as:

$$\vec{\text{AE}} = \left| \vec{actual} - \vec{predicted} \right| \tag{5.6a}$$

$$\text{SD-AE} = \sqrt{\frac{1}{n}\sum_{i=0}^{n}(\vec{\text{AE}}_i - \mu)} \tag{5.6b}$$

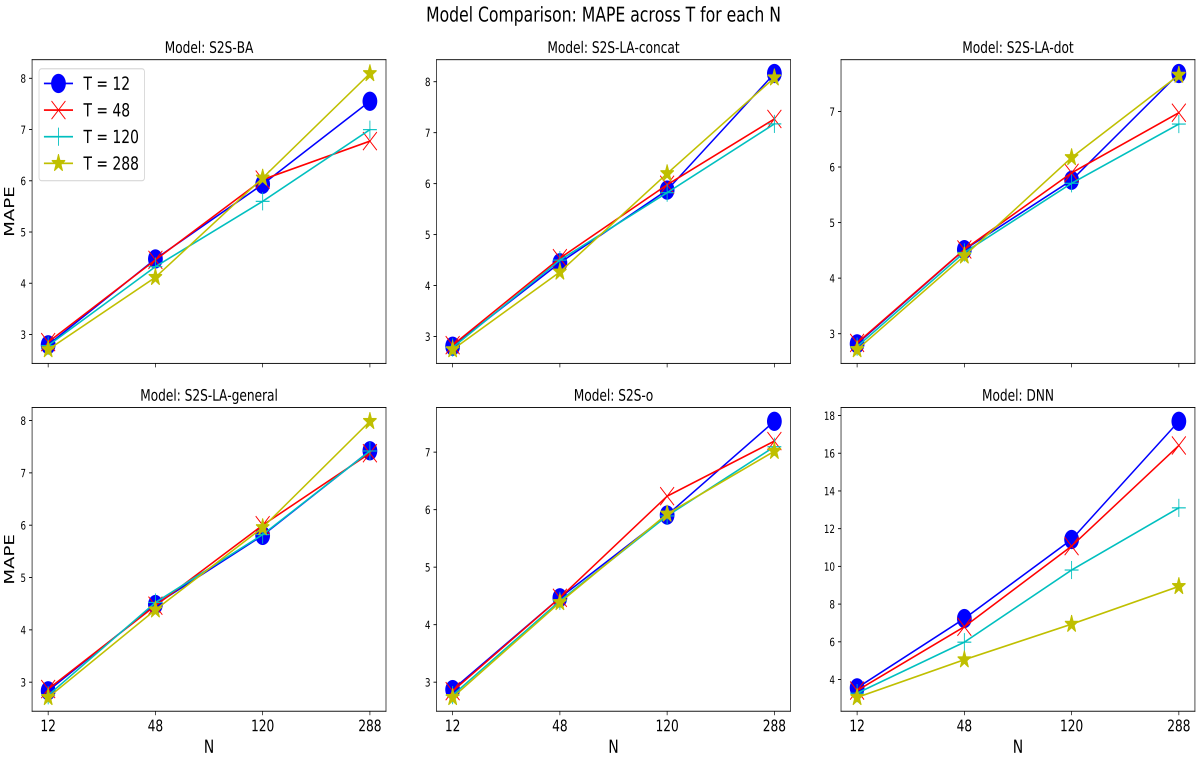
Table 5.1 shows that the S2S-BA model was dominant among all models. It obtained the best MAPE for $N = 12, 48, 120$ and almost matches the best MAPE for $N = 288$. The S2S-BA model also obtained the best MAE for $T = 48, 120, 288$ and obtained the second best MAE for $N = 12$. However, for each MAPE achieved by the S2S-BA model, it did not achieve the lowest SD-APE. A lower standard deviation means the model obtained results that are closer to the mean (more reliable). Hence, Table 5.1 shows that the S2S-LA-dot model provides results closest to the achieved MAPE ($N = 12, 120$) and MAE ($N = 12, 120, 288$), making it slightly more reliable than the other models. Therefore, if a slight loss in accuracy is an

Table 5.1: Best achieved MAPE and MAE for each model to predict $N_i \in \vec{N}$. Input length, cell used, and hidden state size not taken into consideration. The respective standard deviation of the APE and AE for those specific test set samples is given in the brackets below.

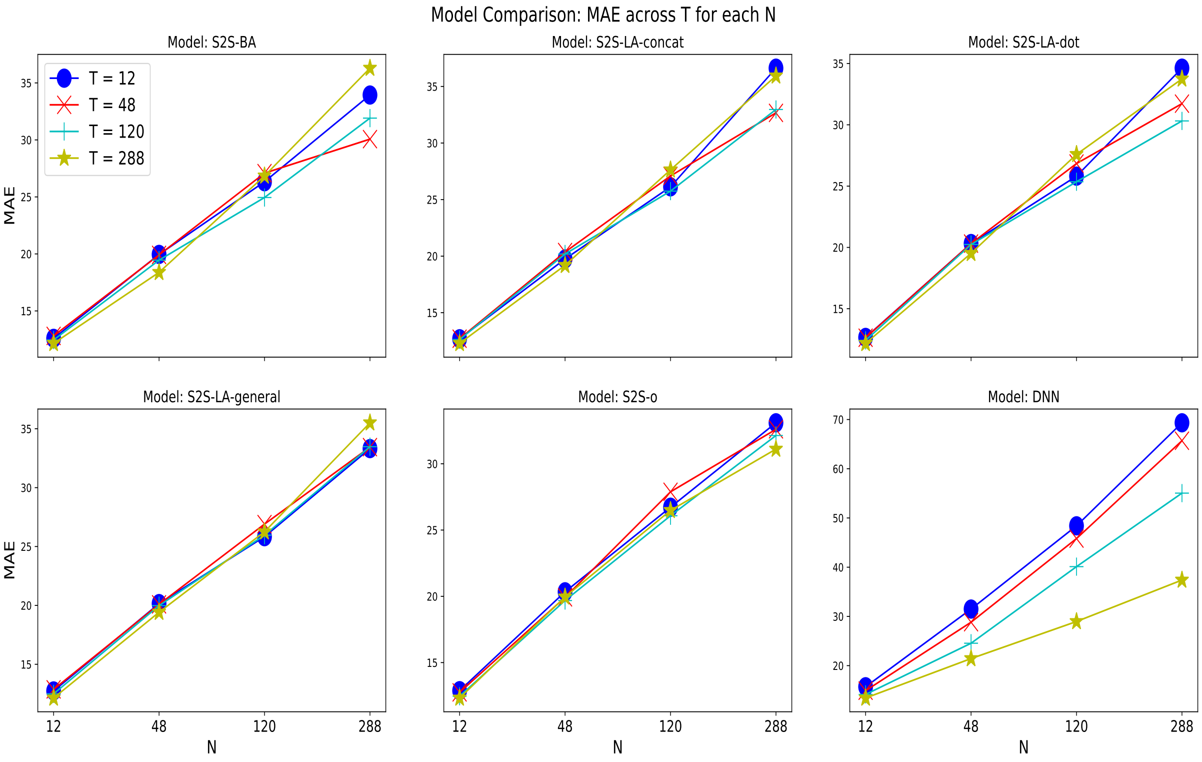
Model	MAPE (%) (SD-APE)				MAE (SD-AE)			
N	12	48	120	288	12	48	120	288
S2S-o	2.735	4.386	5.891	7.016	12.328	19.683	26.081	31.109
	(2.803)	(3.994)	(5.401)	(6.923)	(12.313)	(18.891)	(24.214)	(30.712)
S2S-LA-dot	2.714	4.401	5.707	**6.773**	12.175	19.518	25.356	30.316
	(2.738)	(4.101)	**(5.125)**	(6.345)	**(11.970)**	(18.500)	**(23.108)**	**(28.027)**
S2S-LA-general	2.709	4.383	5.805	7.375	**12.137**	19.421	25.836	33.316
	(2.750)	(4.073)	(5.398)	(7.674)	(11.984)	(18.068)	(24.806)	(37.264)
S2S-LA-concat	2.743	4.266	5.822	7.172	12.287	19.189	25.747	32.664
	(2.783)	**(3.826)**	(5.477)	**(6.055)**	(12.266)	(18.007)	(23.993)	(30.296)
S2S-BA	**2.705**	**4.116**	**5.599**	6.774	12.155	**18.383**	**24.948**	**30.069**
	(2.801)	(3.863)	(5.219)	(6.859)	(12.482)	**(17.650)**	(23.372)	(31.783)
Non-S2S	3.380	5.606	7.372	9.147	15.146	24.668	33.249	40.559
	(3.234)	(5.037)	(6.704)	(8.471)	(14.117)	(23.244)	(33.808)	(39.808)
DNN	3.051	5.051	6.936	8.938	13.424	21.436	28.966	37.389
	(3.005)	(4.823)	(7.100)	(10.523)	(12.688)	(19.542)	(25.817)	(36.819)

acceptable exchange for a slight increase in reliability, the S2S-LA-dot model is preferred. In addition, the preferred model of choice, for interested parties, may not necessarily be the S2S-BA model, the S2S-LA-dot model, or any attention model for that matter. The attention models contain more parameters than all other models, with the S2S-BA model containing the most parameters. Nonetheless, the S2S-o model achieves comparable results to all attention based models, for each prediction length, while needing fewer parameters and therefore is faster to train. Hence, if the interested parties main objective is high accuracy irrelevant of the training speed and reliability, the S2S-BA model is preferred. If a slight decrease in accuracy and reliability is acceptable, a reduction in training speed can be achieved by using the S2S-o model.

Furthermore, figures 5.1, 5.4, 5.7, and 5.10 showed that the Vanilla RNN based attention models outperformed the regular S2S model as N increased. These figures also showed that the GRU and LSTM-based Non-S2S RNN model performed the worst among all models. Lastly, it can be seen from Fig. 5.13 that all S2S attention models performed better, for longer prediction lengths N, when an input length less than $T = 288$ was used. This was not the case for S2S-o and DNN models, as they both performed better when longer input lengths were used.



(a) Analyzing the best MAPE achieved by models, as input length T is varied for each prediction length $N_i \in \vec{N}$.



(b) Analyzing the best MAE achieved by models, as input length T is varied for each prediction length $N_i \in \vec{N}$.

Figure 5.13: Analyzing model performance for varying input lengths T, by (a) MAPE and (b) MAE comparison.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

Continuously rising electricity consumption and its impact on the environment is increasing importance of efficient energy management and conservation strategies. Load forecasting is contributing to energy management efforts through improved maintenance scheduling, budget planning, and by identifying energy savings opportunities. Feedforward neural networks and Support Vector Regression have had a great success in load forecasting; however, Recurrent Neural Networks have an advantage because of their ability to model time dependencies.

This thesis adapted an attention-based Sequence-to-Sequence RNN algorithm with a novel sample generation approach for the task of building-level energy forecasting. RNNs provide the ability to model time dependencies while the S2S approach strengthens this ability by using an encoder-decoder framework. For the sample generation, the training and testing samples were generated in a way that is unique to energy forecasting problems. For the training set, an index was chosen at random, and from this index to the end of the desired input length, a training sample was obtained. The actual usage vector, that accompanied this input sample, was obtained from the end of the input length to the end of a desired prediction length ahead. This usage vector, or output sample, was compared with the predicted usage vector that our models generated. For the testing set, an index was chosen without randomization, the input sample and output sample were generated as before, then the index was shifted the desired prediction length, and the process was repeated.

For the machine learning algorithm, this work adapts a S2S approach from language translation for energy load forecasting: the encoder RNN compresses information from the input sequence into a fixed length context vector, which the decoder RNN then uses to sequentially output a predicted value. The previously predicted output and previous hidden state are passed as next inputs in the decoder.

This work also adapts the Bahdanau *et al.* and Luong *et al.* S2S attention mechanisms, used in NMT, for energy load forecasting: the decoder considers the encoder outputs at each decoder step before making a prediction. These attention models were introduced to alleviate the encoder task of solely compressing all input information into a fixed-length vector. Also known as alignment models in the task of NMT, each decoder step computes attention weights: the probabilities that the target output is aligned to, or most derived from, the respective input vector. While BA only considered one attention score function, LA considered three; meaning a total of four different S2S attention-based models were evaluated.

Overall, seven different models were evaluated: the regular S2S and four attention based models were compared with two baseline models, a Non-S2S RNN and a DNN with sizes small, medium, and large. Furthermore, each Non-DNN model was evaluated with three different RNN cells: Vanilla RNN, GRU, and LSTM. All models were trained and tested on five minute incremental data, with equivalent hyperparameters, and compared for four different cases. Hence, input length was held constant for four separate values, while prediction lengths were varied. Lastly, the models were compared with fixed prediction length while the input length was varied.

The Vanilla RNN based attention models outperformed the regular S2S model as prediction length increased, while the GRU and LSTM-based Non-S2S RNN models performed the worst among all Non-DNN models. The S2S-BA model proved to be dominant as it outperformed all other models, in terms of MAPE and MAE for most of the four prediction lengths. However, the S2S-LA-dot model proved to be the most reliable model, in terms of lowest SD-APE and SD-AE for most of the prediction lengths. While the DNN model did show the most improvement as input length increased, it was still outperformed by all S2S models, for each prediction length. However, it was shown that the S2S-o model achieved comparable results to all attention-based models, for each prediction length. The preferred model choice rests on the needs of the interested parties; the S2S-LA-dot model is suggested if reliability is the number one priority, while the S2S-BA model is suggested if accuracy is valued over training speed and reliability. The S2S-o model is suggested if training speed is vital and a slight decrease in accuracy and reliability is not a concern.

6.2 Future Work

Future work holds the potential to improve presented models, fully understand minor uncertainties, and add insight to areas that peaked our curiosity. Thus, future work will analyze:

- Changing the data preparation process for the test set. Hence, changing the input and output sample generation as done for the training set, then averaging the overlapping target values. This could potentially improve accuracy as the test samples would be generated and passed to the models for each of the time indices, not just for certain times as was done in the presented thesis.

- Changing the dataset entirely. Hence, evaluate the models on different datasets, such as residential, schools, offices, etc., to observe if the models will behave similarly. Also, evaluate the models on datasets with different increments, such as 15-minute or hourly increments.

- Using a local attention mechanism instead of a global one, for all attention models. This will allow attention models to not use all encoder outputs for longer input lengths, hence improved training time.

- Evaluating more models such as one-dimensional Convolutional Neural Networks, and Support Vector Regression.
- Evaluating the GRU and LSTM cells to see whether a different initialization of the weights improves accuracy measures.
- Further analyzing the Vanilla RNN-based S2S attention models. These results proved comparable to the GRU and LSTM-based models, for shorter prediction lengths.
- Testing the models for extreme input and prediction lengths. Hence, see how well the models perform for very short input lengths and for predictions lengths greater than one thousand steps. Then compare to the DNN, as it may be the case that the DNN will outperform the S2S models for extreme prediction lengths.
- Performing a grid search for parameter optimization. Hence, a smaller batch size could lead to improved accuracy measures, while a smaller or decaying learning rate could improve performance. Furthermore, a larger hidden size would give each model more parameters. However, this would further slow down the training speed of the S2S attention models. It could also see the S2S-o models perform better, and holds potential to see the Vanilla RNN S2S models outperform the GRU and LSTM-based models for shorter prediction lengths.
- Modifying all models for an online learning scenario: hence, see how well the models perform for a constant influx of data. For these tests, we hypothesize that training speed will be more valuable than obtaining the highest accuracy.

The adapted S2S attention-based models performed well on the considered dataset, achieving better accuracy results than the regular S2S model and two baseline models. However, there is still space for further improvements, such as in testing set sample generation, using a different dataset, evaluating more models, and completing a much larger grid search. Further analysis would fine-tune these dominant time series models.

Bibliography

[1] U.S. Energy Information Administration. Use of energy in the United States explained. `https://www.eia.gov/energyexplained`, 2017. [Accessed July 2019].

[2] U.S. Energy Information Administration. Total energy. `https://www.eia.gov/totalenergy/data/browser`, 2018. [Accessed July 2019].

[3] U. N. E. Programme. Energy. `https://www.unenvironment.org/explore-topics/energy`, 2018. [Accessed July 2019].

[4] I. E. Agency. Energy. `https://www.iea.org/weo2018/`, 2018. [Accessed July 2019].

[5] H. C. Akdag and T. Beldek. Waste management in green building operations using GSCM. *International Journal of Supply Chain Management*, Vol. 6 (3): pp. 174–180, 2017.

[6] Independent Electricity System Operator. Global adjustment and peak demand factor. `http://www.ieso.ca/Sector-Participants/Settlements/Global-Adjustment-and-Peak-Demand-Factor`, 2019. [Accessed: July 2019].

[7] Y. Wang, D. Gan, M. Sun, N. Zhang, Z. Lu, and C. Kang. Probabilistic individual load forecasting using pinball loss guided LSTM. *Applied Energy*, Vol. 235: pp. 10–20, 2019.

[8] S. Mujeeb, N. Javaid, M. Ilahi, Z. Wadud, F. Ishmanov, and M. K. Afzal. Deep long short-term memory: A new price and load forecasting scheme for big data in smart cities. *Sustainability*, Vol. 11 (4): pp. 987–1016, 2019.

[9] M. Sun, T. Zhang, Y. Wang, G. Strbac, and C. Kang. Using bayesian deep learning to capture uncertainty for residential net load forecasting. *IEEE Transactions on Power Systems*, 2019.

[10] K. Grolinger, A. L'Heureux, M. A. Capretz, and K. Seewald. Energy forecasting for event venues: big data and prediction accuracy. *Energy and Buildings*, Vol. 112: pp. 222–233, 2016.

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

[12] Z. Yu, Z. Niu, W. Tang, and Q. Wu. Deep learning for daily peak load forecasting–a novel gated recurrent neural network combining dynamic time warping. *IEEE Access*, Vol. 7: pp. 17184–17194, 2019.

[13] S. Bouktif, A. Fiaz, A. Ouni, and M. Serhani. Single and multi-sequence deep learning models for short and medium term electric load forecasting. *Energies*, Vol. 12 (1): pp. 149–170, 2019.

[14] X. Gao, X. Li, B. Zhao, W. Ji, X. Jing, and Y. He. Short-term electricity load forecasting model based on EMD-GRU with feature selection. *Energies*, Vol. 12 (6): pp. 1140–1158, 2019.

[15] A. Graves, S. Fernandez, F. Gomez and J. Schmidhuber. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. *Proceedings of the International Conference on Machine Learning (ICML)*, pages 369–376, 2006.

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

[17] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *Proceedings of the International Conference on Learning Representations (ICLR)*, 2015.

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

[19] D. Alberg and M. Last. Short-term load forecasting in smart meters with sliding window-based arima algorithms. *Vietnam Journal of Computer Science*, Vol. 5: pp. 241–249, 2018.

[20] G. E. Dahl, D. Yu, L. Deng, and A. Acero. Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. *IEEE Transaction on Audio, Speech, and Language Processing*, Vol. 20 (1): pp. 30–42, 2012.

[21] G. Krishna, C. Tran, J. Yu, Jianguo and A. H. Tewfik. Speech recognition with no speech or with noisy speech. *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1090–1094, 2019.

[22] H. Zen, A. Senior, and M. Schuster. Statistical parametric speech synthesis using deep neural networks. *Proceedings of the International Conference on Acoustics, Speech and Signal Processing*, pages 7962–7966, 2013.

[23] J. Lee, K. Song, K. Noh, T. Park, and J. Chang. DNN based multi-speaker speech synthesis with temporal auxiliary speaker id embedding. *Proceedings of the International Conference on Electronics, Information, and Communication (ICEIC)*, 2019.

[24] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Proceedings of Advances in Neural Information Processing Systems (NIPS)*, pages 1097–1105, 2012.

[25] Z. Zhao, P. Zheng, S. Xu, and X. Wu. Object detection with deep learning: a review. *IEEE transactions on neural networks and learning systems*, 2019.

[26] D. C. Cireşan, A. Giusti, L. M. Gambardella, and J. Schmidhuber. Mitosis detection in breast cancer histology images with deep neural networks. *Proceedings of the International Conference on Medical Image Computing and Computer-assisted Intervention*, pages 411–418, 2013.

[27] M. Havaei, A. Davy, D. Warde-Farley, A. Biard, A. Courville, Y. Bengio, C. Pal, P. Jodoin, and H. Larochelle. Brain tumor segmentation with deep neural networks. *Medical image analysis*, Vol. 35: pp. 18–31, 2017.

[28] J. Elman. Finding structure in time. *Cognitive Science*, Vol. 14 (2): pp. 179–209, 1990.

[29] Y. Zhao, R. Yang, G. Chevalier, X. Xu, and Z. Zhang. Deep residual bidir-LSTM for human activity recognition using wearable sensors. *Mathematical Problems in Engineering*, Vol. 2018: pp 1–13, 2018.

[30] P. J. Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, Vol. 78 (10): pp. 1550–1560, 1990.

[31] R. Pascanu, T. Mikolov, and Y. Bengio. On the difficulty of training recurrent neural networks. *Proceedings of the International Conference on Machine Learning (ICML)*, pages 1310–1318, 2012.

[32] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, Vol. 9 (8): pp. 1735–1780, 1997.

[33] S. Varsamopoulos, K. Bertels, and C. G. Almudever. Designing neural network based decoders for surface codes. *arXiv preprint arXiv:1811.12456*, 2018.

[34] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk and Y. Bengio. Learning phrase representations using RNN encoder–decoder for statistical ma-

chine translation. *Proceedings of the Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, 2014.

[35] J. Che, L. Tang, S. Deng, and X. Su. Chinese word segmentation based on bidirectional GRU-CRF model. *International Journal of Performability Engineering*, Vol. 14 (12): pp. 3066–3075, 2018.

[36] D. L. Marino, K. Amarasinghe and M. Manic. Building energy load forecasting using deep neural networks. *Proceedings of the IEEE Industrial Electronics Society (IECON)*, pages 7046–7051, 2016.

[37] E. Mocanu, P. H. Nguyen, M. Gibescu and W. L. Kling. Deep learning for estimating building energy consumption. *Sustainable Energy, Grids and Networks*, Vol. 6 (1): pp. 91–99, 2016.

[38] T. Hong. Energy forecasting. `http://blog.drhongtao.com/2014/10/very-short-short-medium-long-term-load-forecasting.html`, 2014. [Accessed: July 2019].

[39] J. G. Jetcheva, M. Majidpour and W. Chen. Neural network model ensembles for building-level electricity load forecasts. *Energy and Buildings*, Vol. 84 (1): pp. 214–223, 2014.

[40] S. Ferlito, M. Atrigna, G. Graditi, S. De Vito, M. Salvato, A. Buonanno, and G. Di Francia. Predictive models for building's energy consumption: an artificial neural network (ANN) approach. *Proceedings of the Italian Association of Sensors and Microsystems (AISEM)*, pages 1–4, 2015.

[41] Y. T. Chae, R. Horesh, Y. Hwang, and Y. M. Lee. Artificial neural network model for forecasting sub-hourly electricity usage in commercial buildings. *Energy and Buildings*, Vol. 111 (1): pp. 184–194, 2016.

[42] D. B. Araya, K. Grolinger, H.F. ElYamany, M. Capretz and G. Bitsuamlak. An ensemble learning framework for anomaly detection in building energy consumption. *Energy and Buildings*, Vol. 144 (1): pp. 191–206, 2017.

[43] S. Seyedzadeh, F. P. Rahimian, I. Glesk and M. Roper. Machine learning for estimation of building energy consumption and performance: a review. *Visualization in Engineering*, Vol. 6 (1): pp. 5–25, 2018.

[44] S. Naji, A. Keivani, S. Shamshirband, U. J. Alengaram, M. Z. Jumaat, Z. Mansor and M. Lee. Estimating building energy consumption using extreme learning machine method. *Energy*, Vol. 97 (1): pp. 506–516, 2016.

[45] N. L. Tasfi, W. A. Higashino, K. Grolinger and M. A. Capretz. Deep neural networks with confidence sampling for electrical anomaly detection. *Proceedings of the IEEE International Conference on Smart Data*, pages 1038–1045, 2017.

[46] K. Amarasinghe, D. L. Marino and M. Manic. Deep neural networks for energy load forecasting. *Proceedings of the IEEE International Symposium on Industrial Electronics (ISIE)*, pages 1483–1488, 2017.

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

[48] H. Shi, M. Xu, and R. Li. Deep learning for household load forecasting—a novel pooling deep RNN. *IEEE Transactions on Smart Grid*, Vol. 9 (5): pp. 5271–5280, 2017.

[49] P. Malhotra, L. Vig, G. Shroff and P. Agarwal. Long short term memory networks for anomaly detection in time series. *Proceedings of the European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN)*, pages 89–95, 2015.

[50] S. Bouktif, A. Fiaz, A. Ouni and M. A. Serhani. Optimal deep learning LSTM model for electric load forecasting using feature selection and genetic algorithm: comparison with machine learning approaches. *Energies*, Vol. 11 (7): pp. 1636–1656, 2018.

[51] H. Zheng, J. Yuan, and L. Chen. Short-term load forecasting using EMD-LSTM neural networks with a Xgboost algorithm for feature importance evaluation. *Energies*, Vol. 10 (8): pp. 1168–1188, 2017.

[52] A. Rahman, V. Srikumar, and A. D. Smith. Predicting electricity consumption for commercial and residential buildings using deep recurrent neural networks. *Applied energy*, Vol. 212: pp. 372–385, 2018.

[53] S. Venugopalan, M. Rohrbach, J. Donahue, R. Mooney, T. Darrell, and K. Saenko. Sequence to sequence-video to text. *Proceedings of the IEEE international conference on computer vision*, pages 4534–4542, 2015.

[54] Y. Chung, and C. Wu, and C. Shen, and H. Lee, and L. Lee. Audio word2vec: unsupervised learning of audio segment representations using sequence-to-sequence autoencoder. *arXiv preprint arXiv:1603.00982*, 2016.

[55] P. Malhotra, V. TV, A. Ramakrishnan, G. Anand, L. Vig, P. Agarwal, and G. Shroff. Multi-sensor prognostics using an unsupervised health index based on LSTM encoder-decoder. *arXiv preprint arXiv:1608.06154*, 2016.

[56] S. H. Park, B. Kim, C. M. Kang, C. C. Chung, and J. W. Choi. Sequence-to-sequence prediction of vehicle trajectory via LSTM encoder-decoder architecture. *IEEE Intelligent Vehicles Symposium (IV)*, pages 1672–1678, 2018.

[57] Y. Qin, D. Song, H. Chen, W. Cheng, G. Jiang, and G. Cottrell. A dual-stage attention-based recurrent neural network for time series prediction. *arXiv preprint arXiv:1704.02971*, 2017.

[58] R. Prabhavalkar, K. Rao, T. N. Sainath, B. Li, L. Johnson, and N. Jaitly. A comparison of sequence-to-sequence models for speech recognition. *Proceedings of the International Speech Communication Association (INTERSPEECH)*, pages 939–943, 2017.

[59] K. Yao, and G. Zweig. Sequence-to-sequence neural net models for grapheme-to-phoneme conversion. *arXiv preprint arXiv:1506.00196*, 2015.

[60] H. Zhang, J. Li, Y. Ji, and H. Yue. Understanding subtitles by character-level sequence-to-sequence learning. *IEEE Transactions on Industrial Informatics*, Vol. 13 (2): pp. 616–624, 2016.

[61] L. Sehovac, C. Nesen, and K. Grolinger. Forecasting building energy consumption with deep learning: a sequence to sequence approach. *Proceedings of International Conference on Internet of Things (ICIOT)*, 2019.

[62] D. P. Kingma and J. Ba. Adam: a method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[63] E. Hoffer, I. Hubara and D. Soudry. Train longer, generalize better: closing the generalization gap in large batch training of neural networks. *Proceedings of Advances in Neural Information Processing Systems (NIPS)*, pages 1731–1741, 2017.

[64] A. M. Saxe, J. L. McClelland, and S. Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv preprint arXiv:1312.6120*, 2013.

[65] A. Paszke, S. Gross, S. Chintala, and G. Chanan. PyTorch: from research to production. `https://pytorch.org`, 2016. [Accessed: May, 2018].

Curriculum Vitae

Name:	Ljubisa Sehovac
Post-Secondary Education and Degrees:	Western University London, ON 2012 - 2017 BSc., Honours Specialization in Applied Mathematics
	Western University London, ON 2017 - 2019 MESc., Electrical and Computer Engineering Vector Institute: Collaborative Specialization in Artificial Intelligence
Honours and Awards:	
Related Work Experience:	Teaching Assistant and Research Assistant Western University 2017 - 2019

Publications:

L. Sehovac, C. Nesen, K. Grolinger. Forecasting building energy consumption with deep learning: a sequence to sequence approach. *Proceedings of International Conference on Internet of Things (ICIOT)*, pages 10, 2019.