Forecasting Building Energy Consumption with Deep Learning: A Sequence to Sequence Approach

Ljubisa Sehovac, Cornelius Nesen, Katarina Grolinger
Department of Electrical and Computer Engineering
Western University
London, Canada
{*lsehovac, cnesen, kgroling*}@*uwo.ca*

Abstract—**Energy Consumption has been continuously increasing due to the rapid expansion of high-density cities, and growth in the industrial and commercial sectors. To reduce the negative impact on the environment and improve sustainability, it is crucial to efficiently manage energy consumption. Internet of Things (IoT) devices, including widely used smart meters, have created possibilities for energy monitoring as well as 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 paper uses Recurrent Neural Networks (RNN) to capture time dependencies and proposes a novel energy load forecasting methodology based on sample generation and Sequence-to-Sequence (S2S) deep learning algorithm. The S2S architecture that is commonly used for language translation was adapted for energy load forecasting. Experiments focus on Gated Recurrent Unit (GRU) based S2S models and Long Short-Term Memory (LSTM) based S2S models. All models were trained and tested on one building-level electrical consumption dataset, with five-minute incremental data. Results showed that, on average, the GRU S2S models outperformed LSTM S2S, RNN S2S, and Deep Neural Network models, for short, medium, and long-term forecasting lengths.**

Keywords-Deep Learning; Energy Load Forecasting; Recurrent Neural Networks; Sequence-to-Sequence; Gated Recurrent Units; GRU; Long Short-Term memory; LSTM

I. Introduction

The rapid expansion of high-density cities [1], combined with growing 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 [2]. Thus, energy efficiency and management will maintain vital roles in building energy consumption, to combat aspects such as the negative environmental effects and carbon dioxide emission [3].

In addition, buildings with efficient energy systems provide beneficial economic opportunities, by way of overall lower operating costs. Commonly commercial, industrial, and other large energy consumers pay premium prices for high energy peaks. For example, the Independent Electricity System Operator (IESO) charges their large customers Global Adjustment (GA) fees based on the consumers' contribution to the top five province-wide peaks in a set period [4]. Another category of consumers is charged premiums based on their peak monthly consumption [4]. The higher is the surge during these hours, the greater are the endured costs. Therefore, it is important for respective parties to improve their energy consumption efficiency during these peak-hours.

The emergence of smart meters has provided valuable data and information with regards to building energy consumption patterns. The usage and analysis of this data can be a pillar for effective energy management strategies. One approach is to train machine learning (ML) algorithms using this data from smart meters in order to predict the energy consumption for a desired time frame ahead. If the algorithm is able to predict values that behave similarly to the actual consumption, the interested parties can make cost-effective decisions based on these values. For example, if the algorithm predicts that the consumption will be high during peak hours, then management can take energy saving measures, plan ahead and budget their consumption behaviour more appropriately. Thus, this work focuses on a novel ML algorithm to successfully forecast building energy consumption.

Machine learning is a form of data analysis that builds models capable of automatically learning from data. A subfield of ML is Deep Learning (DL) which involves multiple layers of nonlinear processing units for data transformations: the outputs from the previous layer are used as the inputs for the next layer. An example of a DL architecture is a Deep Neural Network (DNN), an Artificial Neural Network (ANN) with multiple layers between the input and output layers [5]. A DNN finds the mathematical transformation that occurs for the input data to produce the output data. Feedforward neural networks process information strictly in a single direction, from input to output.

A Recurrent Neural Network (RNN) is a class of DNN where the connections between nodes constitute a directed graph along a sequence [6]. While general feedforward DNNs consider the current input, RNNs consider the current input along with the previously received inputs. RNNs also differ by using an internal hidden state (memory) as a mechanism to remember information throughout a temporal sequence. For these reasons, RNNs have an advantage in analyzing the temporal dynamic behaviour of a sequence [7].

Thus, to accurately forecast energy consumption, this work utilizes Sequence-to-Sequence (S2S) Recurrent Neural

Networks. These are two RNNs side-by-side, with the first RNN encoding information and the second RNN sequentially predicting from it [8]. These S2S models were chosen over the conventional DNNs because of the long range temporal dependencies they provide [8]. The work focused on Gated Recurrent Unit (GRU) based S2S models and Long Short-Term Memory (LSTM) based S2S models. These models were compared to a standard RNN based S2S model, as well as a traditional DNN model, in order to determine which model provides best accuracy.

The rest of the paper is organized as follows: Section II discusses the background, Section III covers the related work, Section IV describes the methodology, Section V explains the experiments and corresponding results, and finally Section VI concludes the paper.

II. Background

This section introduces RNNs and describes Long Short-Term Memory (LSTM) and gated Recurrent Unit (GRU) cells.

A. Recurrent Neural Networks

Deep Neural Networks are ML models that have achieved success in solving many problems, such as speech recognition [9] and object recognition in images [10]. Nonetheless, commonly used DNN architectures, such as feedforward neural networks, are not well-suited at predicting a sequence consecutively, since DNNs directly predict the sequence. Such networks generate predictions based solely on the current input, irrelevant of any prior inputs, thus neglecting temporal dependencies present in time series problems. Hence, RNNs provide a solution by using an internal state to remember information in sequential time steps.

Conventional RNNs take a sequence of inputs $(x_{[1]}, ..., x_{[T]})$ to compute a sequence of outputs $(y_{[1]}, ..., y_{[T]})$, where t is a time step, $t \in 1, ..., T$. Hence, $y_{[t]}$ is computed using the inputs $x_{[t]}, x_{[t-1]} ..., x_{[1]}$. This is 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 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", conventional RNNs will struggle at predicting the consecutive sequence "so, I, went, outside" and will be unable to predict a sequence of different length such as "and, sunny."

In comparison, S2S-RNNs contain two RNNs, an Encoder and Decoder RNN. The general idea, as shown in Fig. 1, is to pass the input sequence $(x_{[1]}, ..., x_{[T]})$ into the Encoder RNN, one time step at a time, to obtain a context vector $(\vec{c})$. This context vector is an encoded representation of the processed input sequence, which is then passed through the Decoder RNN, extracting information at each unraveled time step to obtain the output sequence $(\dot{y}_{[1]}, ..., \dot{y}_{[N]})$, where $n \in 1, ..., N$. For the Decoder RNN, the predicted output at n is used as the next input, hence $\dot{y}_{[n]}$ is computed using the inputs $\dot{y}_{[n-1]}, ..., \dot{y}_{[0]}, x_{[T]}, ..., x_{[1]}$. Note that $\dot{y}_{[0]}$ is the context value, derived from $\vec{c}$, and is used as the initial input for the Decoder RNN. 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 ("the, sky, was, blue" = 4 steps), the output can be an arbitrary N time steps ("and, sunny" = 2 steps).

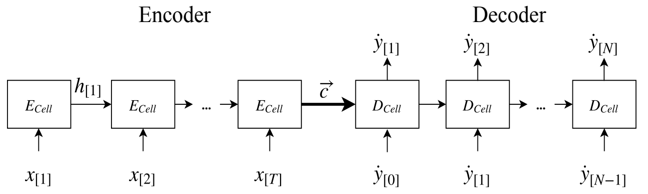

Figure 1. Sample passed through the RNN S2S network. Here, $\vec{c}$ and $\dot{y}_{[0]}$ denote the context vector and context value respectively.

B. LSTM and GRU Cells

This subsection provides a brief overview of the algorithms in Long Short-Term Memory (LSTM) and Gate Recurrent Unit (GRU) networks. While traditional RNN models are mainly trained using the back-propagation through time algorithm [11], this method will lead to the vanishing gradient problem for longer sequences [12]. Thus, LSTM networks [13] 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 contains three gates (input i, forget f and output o), an update step g, a cell memory state c and a hidden state h. Formally, the computations in a single LSTM cell at time t, for input x, can be given as [13]:

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

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

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

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

$$c_{[t]} = f_t \odot c_{[t-1]} + i_t \odot g_t \tag{1e}$$

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

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 (1b) 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 (1e). A value closer to 1 means ”keep this information”, while a value closer to 0 means ”forget”. The input and output gates work similarly.

The GRU model [14] was recently introduced to simplify the LSTM model, while maintaining similar functionality. 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 (2c). Formally, the computations in a single GRU cell are given as [14]:

$$r_t = \sigma(W_{xr}x_{[t]} + b_{xr} + W_{hr}h_{[t-1]} + b_{hr}) \tag{2a}$$

$$z_t = \sigma(W_{xz}x_{[t]} + b_{xz} + W_{hz}h_{[t-1]} + b_{hz}) \tag{2b}$$

$$k_t = \tanh(W_{xk}x_{[t]} + b_{xk} + r_t \odot (W_{hk}h_{[t-1]} + b_{hk})) \tag{2c}$$

$$h_{[t]} = (1 - z_t) \odot k_t + z_t \odot h_{[t-1]} \tag{2d}$$

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.

III. Related Work

Forecasting energy can be classified into three main categories: short, medium, and long-term [15] [16]. Unfortunately, long term forecasting still poses problems to researchers, especially those working with one-minute [15] [16] or even five-minute incremental data. Nonetheless, the presented work focuses on prediction lengths comparable to short, medium, and long-term energy forecasting.

There are two main methods for carrying out energy load forecasting: physics based models and statistical/machine learning based models. Physics based models incorporate engineering principles which relay on a complex mix of material, structural, and geometric properties. The statistical based models analyze historical energy consumption data, by implementing ML algorithms to mathematically represent the relationship between the historical data and variables affecting energy consumption. This section concentrates on the statistical methods as our work belongs to that category.

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.* [17] proposed an ANN model to forecast day-ahead building-level energy, with an ensemble approach to select model parameters. The use of ANNs for general load forecasting has been explored in several studies, for all three forecasting horizons: short, medium and long [18] [19]. In comparison, the work by Naji *et al.* [20] predicted building energy consumption by applying an Extreme Learning Machine method with the data regarding building material thickness and their thermal insulation capability. Several studies [21] [22] have proposed ANN and SVM models for estimating energy consumption and compared performance. Convolutional Neural Networks (CNN) have also been used for load forecasting [23], which showed to outperform SVM models while achieving comparable results to ANN and other deep learning methods [24]. The work by Mocanu *et al.* [16] showed that newly developed stochastic models, Factored Conditional Restricted Boltzmann Machine, outperformed ANN, SVM, and classic 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 S2S GRU and LSTM based models. These S2S models offer a stronger analysis in time series problems, since their internal hidden state is passed through a directed graph along a sequence. This allows S2S models to retain information in sequential data better than traditional ANNs, SVMs and CNNs.

LSTM and GRU based RNNs are becoming increasingly popular for time series regression problems. The work by Malhotra *et al.* [25] used standard LSTM networks to detect anomalies in power demand. Similarly, Bouktif *et al.* [26] used a standard LSTM model, coupled with a genetic algorithm, for short to medium term aggregate load forecasting. Although these two studies used LSTM based models for load forecasting, their focus was on standard LSTM prediction, rather than S2S prediction as in our work. The main difference, as described in section II, is that standard 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.

The most comparable work to ours is the work by Marino *et al.* [15]. They used standard LSTM and LSTM-based S2S models to forecast residential-level energy consumption, on one hour and one-minute time step datasets. It was shown that the S2S model performed well on both datasets, and produced comparable results with other deep learnings methods [16]. While our work also used S2S models, it varies by means of: overall different algorithm, sample generation, and longer prediction sequence length. Moreover, the technique of using the previous Decoder output as next input is not utilized in the aforementioned work for their respective LSTM-S2S model.

Therefore, the combination of sample generation and S2S algorithm in the presented work provides a novel approach to building-level load forecasting.

IV. Methodology

This section discusses the detailed methodology process. The features and evaluation process are introduced first, followed by the sample generation and S2S algorithm.

A. *Features and Evaluation Process*

Smart meters are digital electricity meters that are able to measure how much electricity is used and when it is used. The original private dataset, provided by an industry partner, contained 5-minute incremental readings from a smart meter, with the time stamp and usage recorded at each reading. The weather data [27] for the meter's specific location was appended to the original dataset. Ultimately, the dataset consisted of readings with 11 features, as given in Table 1. Three readings from the dataset are randomly given to show how the data can vary. There was a "minute" feature, but it was omitted since the usage at minute 55 and minute 0, of the next hour, could be very similar, while the minute readings are the maximum and minimum respectively. For re-assurance, simulations were run with the extra "minute" feature with no improvement in accuracy.

The entire dataset was divided into a training and test set. The first 80% of all readings were assigned as the training set and the last 20% as the test set. Fig. 2 shows where the train set ends and test set begins for the usage feature. This validation process was chosen since the problem is time series specific, and thus, randomizing the entire dataset prior to subsetting (splitting into train and test set) is counter intuitive; it would potentially allow the model to see the most recent energy data during training, while being tested on the first most data. Note that the usage data, as seen in Fig. 2, 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.

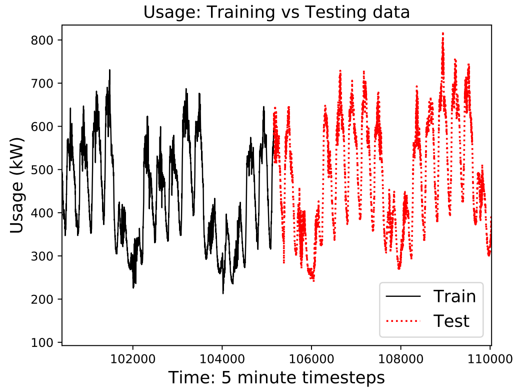

Figure 2. Usage data zoomed in to show breakdown between the train and test sets.

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

$$\tilde{x} = \frac{x - \mu}{\sigma} \tag{3}$$

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.

B. *Sample Generation*

An important step of the S2S approach is the sample generation process. More specifically, how the input and target samples were generated to be passed to the model. Let one input sample be represented as a matrix, $X \in \mathbb{R}^{T \times f}$, where T is the number of time steps and f is the number of features. As defined in the previous subsection, the number of features was $f = 11$, where each row of the matrix X is defined as:

$$\begin{aligned} x_{[t]} = [&\text{Month}_{[t]}\ \text{DayOfYear}_{[t]}\ \text{DayOfMonth}_{[t]} \\ &\text{Weekday}_{[t]}\ \text{Weekend}_{[t]}\ \text{Holiday}_{[t]}\ \text{Hour}_{[t]} \\ &\text{Season}_{[t]}\ \text{Temp}_{[t]}\ \text{Humidity}_{[t]}\ \text{Usage}_{[t]}] \end{aligned} \tag{4}$$

Where $t \in 1, ..., T$. 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:

$$y = [\text{Usage}_{[1]},\ \dots\ ,\ \text{Usage}_{[n]},\ \dots\ ,\ \text{Usage}_{[N]}] \tag{5}$$

Where $y_{[n]}$ is the actual usage value at time step n, and $n \in 1, ..., N$. 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. 3. 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. 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 uniformly 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 formal process for sample generation, of the training set, can be given as:

1) Uniformly randomize indices i. Here, $i \in 1, ..., i_{last}$. Where i_{last} denotes the total length of the training set minus the last $T + N$ indices. If i was chosen to be

Table I
FEATURE DATA (BEFORE STANDARDIZATION)

Index	Month	Day of Year	Day of Month	Weekday	Weekend	Holiday	Hour	Season	Temp. (°C)	Humidity	Usage (kW)
0	7	187	5	1	0	0	0	3	21.0	83.0	405.8743
23470	9	268	24	5	1	0	18	4	19.0	60.0	332.7853
104275	7	184	3	0	0	1	7	3	18.0	100.0	297.4848

$i_{last}+1$, 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, a sliding window technique was used; the window shifted sequentially with 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. 3 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 formal process for sample generation, of the test set, can be 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 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 **concatenate** 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 and plotting.

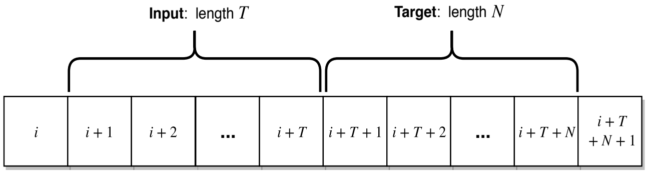

Figure 3. Training set sample generation. Input and target samples generated from random index $i+1$ of the dataset.

C. S2S Algorithm

This section gives a detailed breakdown of the S2S algorithm used for this work. As briefly described in section II, each RNN (LSTM and GRU) has an Encoder RNN (E_T) of length T and Decoder RNN (D_N) of length N respectively. The inputs, as obtained in the previous subsection, are passed through E_T, one time step at a time. Hence, Fig. 1 (a) shows E_T unrolled, where a vector, as in (4), is passed to it at each time step. Each cell takes the previous hidden state and the current input at time t, performs the actions discussed in section II (equations 1a-f for an LSTM cell or 2a-d for a GRU cell), and outputs a hidden state. Note that GRU cells output one hidden state, while LSTM cells output both a cell state and a hidden state.

Fig. 5 shows how E_T and D_N connect for the proposed 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 [8], since it encodes context from the entire input sequence. This context vector, shown as $h_{[T]}$ in Fig. 5, 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, otherwise known as the number of features in the hidden state of each cell. Note that the E_T cell took $f = 11$ features as input, while the D_N cell only takes $f = 1$ feature, hence the previously predicted usage value $\dot{y}_{[n-1]}$. 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. Hence, this approach is novel to building-level load forecasting, and separates this work from others in literature. Thus, let us denote these proposed models with a "-o" notation, representing the distinguishing characteristic of the proposed architecture for energy load forecasting; one predicted value computed at time step n is passed as the next input. Hence, GRU S2S will be denoted as GRU S2S-o.

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

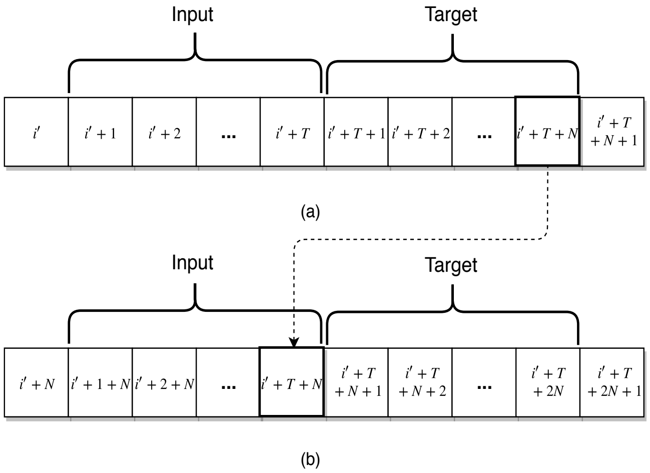

Figure 4. Test set sample generation. (a) Input and target samples generated from dataset at index $i' + 1$. (b) Sequential input and target samples generated from sliding window with overlap length N.

the loss from the target usage vector, formally given as:

$$L = \frac{1}{n}\sum_{n=1}^{N}(y_{[n]} - \dot{y}_{[n]})^2 \qquad (6)$$

Where (6) 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 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 [28] 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 formally given in Algorithm 1. In step 13, we initialize the hidden state to be $\vec{0}$ of size $B \times h$, where B is denoted as the batch size - the number of training examples the model processes for one update of weight parameters [29]. Batching is done to speed up convergence. Since the weight parameters are updated **per batch**, a smaller batch size provides a more accurate update. Once all the batches have been processed by the model, one full epoch has occurred. Steps 15-17 show the input being passed through the E_T. Steps 19-20 transform the output of E_T, the context, to create the initial input and hidden state of the D_N. Steps 22-27 describe the process of predicting the usage value, appending that value, and preparing the next input and hidden state. In steps 29-32, 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 subsection IV-B) and do not apply BPTT on the loss, hence do not update the parameters in anyway.

V. Experiments and Results

The used dataset is based on approximately 1 year and 3 months of energy consumption data, in kW, for one commercial building/smart meter. The readings were in five minute increments, so the total number of readings was 12 readings in one hour $\times$ 24 hours in one day $=$ 287 $\times$ 458 days $=$ 131,446. As stated in section IV, each reading, or row of data, consists 11 features. The rest of this section covers the conducted experiments and their results.

A. Experiments

The model was tested for three different prediction length scenarios, which we consider here as short, medium, and long prediction lengths. Since input length T to prediction length N combinations are endless, the following **three cases** were chosen for experiments:

- **Case 1** (Short): input $T = 12$, predict $N = 6$
- **Case 2** (Medium): input $T = 144$, predict $N = 48$
- **Case 3** (Long): input $T = 288$, predict $N = 120$

In 5 minute increments, those cases equate to: 1 hour $\rightarrow$ predicts next 30 minutes, 12 hours $\rightarrow$ predicts next 4 hours, and 24 hours $\rightarrow$ predicts next 10 hours, respectively. Although the longest length of applicable prediction is 10 hours, this length correlates to predicting 120 time steps. With datasets consisting of one-hour increments, 120 time steps would translate to predicting 120 hours (5 days) consecutively.

No model was trained for longer than 10 epochs, and the hyperparameters were kept the same for each respective model, and for each case. In all experiments, 10 epochs was sufficient to reach an acceptable level of convergence. The total loss computed, for both the training and test set, showed minimal signs of improvement as the 10th epoch approached. These precautions were taken to observe and compare each model's performance fairly and reasonably in

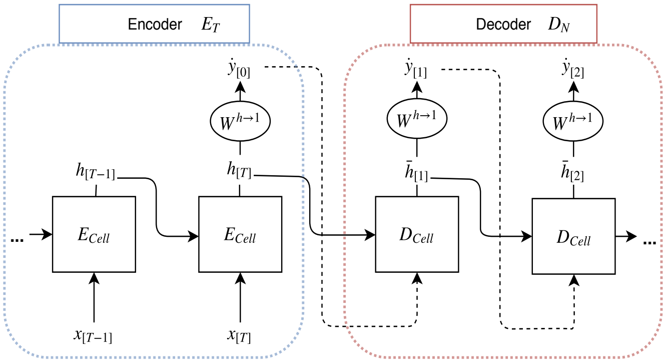

Figure 5. Detailed process of the S2S RNN. The left box shows the last two steps of E_T, while the right box shows the first two steps of D_N.

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

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

Generate randomized 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 $h_{[t-1]} \leftarrow \vec{0} \in \mathbb{R}^{B\times h}$, $B =$ batch size

    for time step $t$ from $1$ to $T$ do
      $h_{[t]} \leftarrow E_{cell}(x_{[t]}, h_{[t-1]})$
      $h_{[t-1]} \leftarrow h_{[t]}$

    $\dot{y}_{[0]} \leftarrow W^{h\rightarrow 1}(h_{[T]})$
    $\bar{h}_{[0]} \leftarrow h_{[T]}$

    for time step $n$ from $1$ to $N$ do
      $\bar{h}_{[n]} \leftarrow D_{cell}(\dot{y}_{[n-1]}, \bar{h}_{[n-1]})$
      $\dot{y}_{[n]} \leftarrow W^{h\rightarrow 1}(\bar{h}_{[n]})$
      append $\dot{y}_{[n]}$ to predicted usage vector $\dot{y}$
      $\bar{h}_{[n-1]} \leftarrow \bar{h}_{[n]}$
      $\dot{y}_{[n-1]} \leftarrow \dot{y}_{[n]}$

    compute $L$ as in (6)
    BPTT($L$)
    $P \leftarrow$ Adam($P_0, r$), $r =$ learning rate
    $P_0 \leftarrow P$

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

terms of accuracy. The hyperparameters used to compute the results were:

- Hidden dimension size $h = 64$
- Cell state dimension size $c = 64$ (only for LSTM)
- Batch size = 256
- Epochs = 10
- Learning rate = 0.001

Note that choosing these external parameters has yet to be optimized, and doing so holds potential for improved results.

The S2S-o models with GRU/LSTM cells were compared against a S2S-o model with the basic RNN cell, as well as a Deep Neural Network, for all three cases. The DNN took the same input matrix $X \in \mathbb{R}^{T\times f}$, but instead flattened it into a single vector of dimension size $= T \times f$. This vector was then passed through three deep layers consisting of 512 nodes per layer. The output was a layer consisting of N nodes, which is equivalent to length N of the predicted usage sequence used in the S2S-o models. The DNN is used to directly predict the sequence of values, all at once, while the S2S-o 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 output layer was linear, since this is a regression problem.

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:

$$\text{MAE} = \frac{1}{n}\sum_{i=0}^{n} |y_i - \hat{y_i}| \tag{7}$$

$$\text{MAPE} = \frac{100\%}{n}\sum_{i=0}^{n} \left|\frac{y_i - \hat{y_i}}{y_i}\right| \tag{8}$$

Where y represents the actual value, $\hat{y}$ the predicted value, and number of observations or samples is given by n.

Lastly, since this work randomized the training samples, it was important to run a few simulations where the model saw a different randomized order of training samples each time. Hence, 7 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.

B. Results and Discussion

The results were computed using Python and the PyTorch optimized tensor library [30], which is an open-source machine learning library.

Fig. 6 shows the predicted usage values obtained from the GRU S2S-o model compared to the actual, for all three cases. Each plot shows only the last 7 days of the test set, and it was produced with the same external parameters for each case. From Fig. 6, we can observe that the accuracy decreases as we move from Case 1 → 3. Hence, Fig. 6 (a) shows the predicted values best fitting to the actual values, with 6 (b) showing lower accuracy and 6 (c) showing the worst. Hence, as the prediction length N increases, the accuracy is expected to decrease [15] [16]. This can also be seen in Table II, which gives the accuracy that each model obtained in terms of MAE and MAPE for all three cases. A visualization of the MAPE increasing from Case 1 → 3 is demonstrated in Fig. 7.

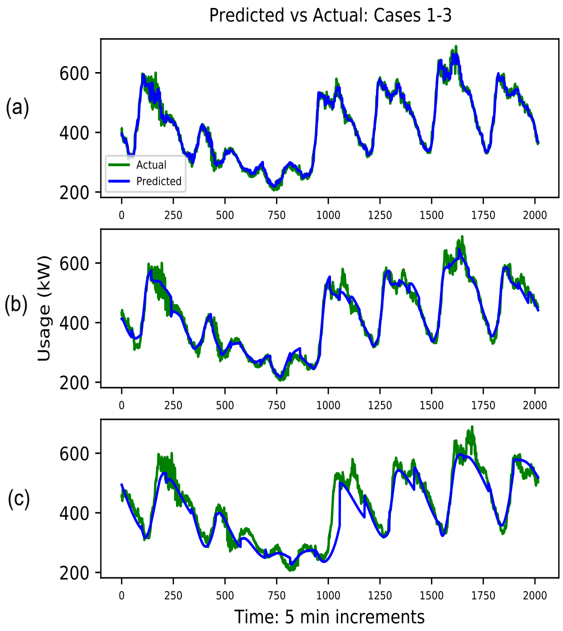

Figure 6. Predicted usage compared to actual, for the three cases: Hence, (a) represents case 1, (b) and (c) represent cases 2 and 3 respectively.

Therefore, as seen in Table II and Fig. 7, even though accuracy does decrease for the GRU/LSTM S2S-o models, it is clear they still outperform the RNN S2S-o and DNN model, for all three forecasting cases. As expected, the GRU/LSTM S2S-o models significantly outperformed the basic RNN S2S-o model for longer prediction horizon represented with case 3. This was expected as LSTM and GRU cells were introduced to overcome the long-term drawbacks that RNN cells pose.

It is interesting to compare the results between the GRU S2S-o and LSTM S2S-o models. For cases 1 and 2, the results are very similar. For case 3, the GRU S2S-o model outperformed the LSTM S2S-o model, with 12.56% decrease in MAE and 14.05% decrease in MAPE. Section II provides a detailed analysis of calculations occurring in an LSTM and GRU cell respectively. The LSTM cell contains an extra secondary state, called the cell state, as well as an extra gate. Even though the GRU cells have fewer states, they outperformed the LSTM cells. This result is similarly confirmed by Chung *et al.* [31] and Jozefowicz *et al.* [32].

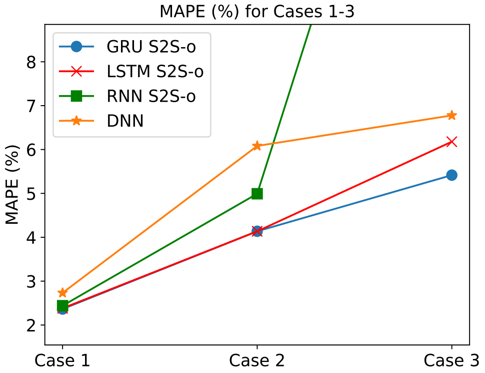

Figure 7. MAPE (%) for the three cases and for each model.

VI. Conclusion

The rapid expansion of high-density cities along with accompanying industrial and commercial development result in increasing energy consumption. To reduce the energy footprint, it is crucial to enhance energy management strategies for buildings; energy forecasting is an important aspect of such initiatives. Sensor-based energy forecasting uses historical data from smart meters or other sensors to predict energy consumption. This study belongs to sensor-based forecasting category; it uses data from smart meters and deep learning algorithms to predict energy consumption. A novel methodology in both sample generation and S2S RNN algorithm for building-level energy forecasting is proposed. Hence, this work adapts a true S2S approach from language translation for energy load forecasting: the predicted output value is passed as the next input in the decoder. The methodology was applied to three S2S RNN cell-based models: basic RNN, GRU, and LSTM. All were trained and tested on five minute incremental data, with equivalent external parameters, and compared against a Deep Neural Network. The GRU S2S-o and LSTM S2S-o models outperformed the basic RNN S2S-o and DNN models, for short, medium, and long-term prediction lengths. On average, the GRU S2S-o model outperformed the LSTM S2S-o model, for all three prediction lengths.

Table II
Accuracy results for the all models and all cases. Top number is best achieved; number in brackets is the average of runs.

Model	MAE			MAPE (%)		
	Case 1	Case 2	Case 3	Case 1	Case 2	Case 3
GRU S2S-o	**10.673** (10.946)	**18.432** (19.369)	**24.378** (26.037)	**2.365** (2.441)	4.137 (4.363)	**5.416** (5.834)
LSTM S2S-o	10.736 (11.029)	18.491 (21.161)	27.441 (29.745)	2.381 (2.451)	**4.136** (4.745)	6.178 (6.695)
RNN S2S-o	10.908 (11.196)	21.584 (43.000)	84.729 (99.994)	2.440 (2.510)	4.992 (10.233)	18.950 (23.698)
DNN	12.258 (12.487)	25.681 (28.845)	28.40 (29.486)	2.733 (2.826)	6.082 (6.898)	6.777 (7.041)

Future work will analyze the GRU S2S-o and LSTM S2S-o model outcomes for longer-term prediction, and apply the S2S-o algorithm to relatable datasets.

Acknowledgment

The authors would like to thank Utilismart Corporation for supplying industry knowledge and data used in this study.

References

[1] H. Ritchie and M. Roser, "Urbanization," 2018, Our World in Data, Available: https://ourworldindata.org/urbanization. [Accessed: Jan. 2019].

[2] U.S. Energy Information Administration, "Use of Energy in the United States Explained," 2017, Available: https://www.eia.gov/energyexplained. [Accessed: Jan. 2019].

[3] 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.

[4] Independent Electricity System Operator, "Global Adjustment and Peak Demand Factor," 2019, Available: http://www.ieso.ca/Sector-Participants/Settlements/Global-Adjustment-and-Peak-Demand-Factor. [Accessed: Jan. 2019]

[5] J. Schmidhuber, "Deep learning in neural networks: an overview," *Neural Networks*, Vol. 61, pp. 85–117, 2015.

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

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

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

[9] 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.

[10] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," In *Proc. of Advances in Neural Information Processing Systems (NIPS)*, pp. 1097–1105, 2012.

[11] 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.

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

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

[14] K. Cho *et al.*, "Learning phrase representations using RNN encoder–decoder for statistical machine translation," In *Proc. of the Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1724–1734, 2014.

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

[16] 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.

[17] 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.

[18] S. Ferlito *et al.*, "Predictive models for building's energy consumption: an artificial neural network (ANN) approach," In *Proc. of the Italian Association of Sensors and Microsystems (AISEM)*, pp. 1–4, 2015.

[19] 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.

[20] S. Naji *et al.*, "Estimating building energy consumption using extreme learning machine method," *Energy*, Vol. 97 (1), pp. 506–516, 2016.

[21] 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.

[22] 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.

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

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

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

[26] 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.

[27] Weather Underground, "Historical Weather," 2018, Available: https://www.wunderground.com/history. [Accessed: May 2018]

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

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

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

[31] J. Chung, C. Gulcehre, K. Cho and Y. Bengio, "Emperical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.

[32] R. Jozefowicz, W. Zaremba, and I. Sutskever, "An emperical exploration of recurrent network architectures," In *Proc. of the International Conference on Machine Learning (ICML)*, pp. 2342–2350, 2015.