Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2019.DOI

Deep Learning for Load Forecasting: Sequence to Sequence Recurrent Neural Networks with Attention

LJUBISA SEHOVAC (Student Member, IEEE) AND KATARINA GROLINGER (Member, IEEE)
[1]Department of Electrical and Computer Engineering, Western University, London, ON N6A 3K7, Canada (e-mail: {lsehovac, kgroling}@uwo.ca)
Corresponding author: Katarina Grolinger (e-mail: kgroling@uwo.ca).

ABSTRACT The biggest contributor to global warming is energy production and use. Moreover, a push for electrical vehicle and other economic developments are expected to further increase energy use. To combat these challenges, electrical load forecasting is essential as it supports energy production planning and scheduling, assists with budgeting, and helps identify saving opportunities. Machine learning approaches commonly used for energy forecasting such as feedforward neural networks and support vector regression encounter challenges with capturing time dependencies. Consequently, this paper proposes Sequence to Sequence Recurrent Neural Network (S2S RNN) with Attention for electrical load forecasting. The S2S architecture from language translation is adapted for load forecasting and a corresponding sample generation approach is designed. RNN enables capturing time dependencies present in the load data and S2S model further improves time modeling by combining two RNNs: encoder and decoder. The attention mechanism alleviates the burden of connecting encoder and decoder. The experiments evaluated attention mechanisms with different RNN cells (vanilla, LSTM, and GRU) and with varied time horizons. Results show that S2S with Bahdanau attention outperforms other models. Accuracy decreases as forecasting horizon increases; however, longer input sequences do not always increase accuracy.

INDEX TERMS Attention Mechanism, Gated Recurrent Units, GRU, Load Forecasting, Long Short-Term memory, LSTM, Recurrent Neural Networks, Sequence-to-Sequence Networks

I. INTRODUCTION

ENERGY production and use is the single biggest contributor to global warming, accounting for roughly two-thirds of human-induced greenhouse gas emissions [1]. 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 [2]. Furthermore, EIA (U.S. Energy Information Administration) estimates that the industrial and commercial sectors consume 50% of the total energy production [3]. Hence, efficient energy management in buildings will prove crucial to combat negative environmental hazards, such as degradation and carbon dioxide emission [4].

In addition to environmental impact, energy efficiency measures in buildings provide economic benefits in terms of reduced overall operating costs. Large electricity consumers commonly pay premium prices for demand peaks. For example, Independent Electricity System Operator (IESO) charges their large consumers fees based on the consumers' contribution to the top five province-wide peaks and another category of consumers is charged premiums based on their monthly peak [5].

Load forecasting has been attracting research and industry attention because of its importance for energy production planning and scheduling. The rapid increase of smart meter use has created opportunities for load forecasting on individual building and household level, thus facilitating budget planning, identifying savings opportunities, and reducing energy footprint.

Energy consumption data from smart meters is used, often together with meteorological data, to build models capable of inferring future energy consumption. One way of building these systems is by training Machine Learning (ML) models using historical data and then using these trained models to predicting future loads. 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. Examples of ML models for load forecasting in-

clude Neural Networks (NN) [6], Support Vector Regression (SVR) [7], and deep learning [8].

Feedforward neural networks (FFNN) have been commonly used for load forecasting [9] and Deep Neural Network (DNN) [8], large NNs with many hidden layers and neurons, have also been applied. They have achieved favorable results [8], [9]; however, FFNN and many Deep Learning (DL) architectures are not designed to capture time dependencies since they only take the current input to calculate predictions. Recurrent Neural Networks (RNNs) are capable of capturing time dependencies as their nodes establish a directed graph along a sequence [10]. This allows RNNs to consider the current input along with the previously received inputs and makes them suitable for time-series data.

While RNNs have an advantage over DNNs in analyzing the temporal dynamic behaviour [11], in language translation a Sequence-to-Sequence (S2S) RNN which combines an encoder RNN and decoder RNN has shown a greater success [12]. The encoder RNN is tasked with encoding information into a fixed-length vector, which the decoder RNN uses to sequentially produce translation outputs [12]. However, in these models the encoder is burdened with compressing all necessary information into this fixed-length vector. In language translation, this was addressed using attention mechanisms [13], [14] which allow the decoder to look back at the encoder outputs to find the most relevant information.

This paper proposes S2S RNN with Attention for load forecasting and evaluates prediction accuracy of different attention mechanisms with varied forecasting horizons. S2S RNN from neural machine translation which is a classification task, is adopted for the regression task of load forecasting. To accommodate S2S RNN, a sample generation approach based on the sliding window is applied. Attention mechanism is added to ease the connection between encoder and decoder. Bahdanau [13] and three variants of Luong [14] attention are considered as well as three RNN cells: vanilla RNN, Gated Recurrent Units (GRU), and Long-Short-Term Memory (LSTM). The results show that S2S with Bahdanau attention outperform DNNs, S2S RNNs, and S2S with other attention mechanisms. As expected, the accuracy decreases as the forecasting horizon expands; however, the increased input sequence length does not always lead to improved accuracy.

The rest of the paper is organized as follows: Section II discusses the background, Section III presents 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, S2S RNNs, and attention mechanisms.

A. RECURRENT NEURAL NETWORKS

Recurrent Neural Networks have an architecture similar to FFNN, but with the addition of recurrent connections to the same neurons in the previous time step. The output at each time step is based on both the current input and the input at the previous time steps, making RNNs good at modeling temporal behaviours found in time series data.

As illustrated in Fig. 1, RNNs take a sequence of inputs $x_{[1]}, ..., x_{[T]}$, and previous hidden states, to compute a sequence of outputs $y_{[1]}, ..., y_{[T]}$. Output $y_{[t]}$ at time step t is:

$$y_{[t]} = f^{\circ}(x_{[t]}, h_{[t-1]}) \tag{1a}$$

where $h_{[t-1]}$ denotes the previous hidden state and f° is a non-linear function, potentially combined of either a Vanilla RNN, LSTM, or GRU cell, and some multi-layered network.

Traditional or Vanilla RNNs are mainly trained using back-propagation through time (BPTT) [15]; however, this method can lead to the vanishing gradient problem for longer sequences [16]. Long-Short-Term Memory (LSTM) networks [17] were designed to overcome this problem; therefore, they are capable of storing information for longer periods of time and the model can make better predictions.

LSTMs are comprised of cells which contain gates responsible for learning which data in a given sequence should be kept and which data can be forgotten. The LSTM cell contains three gates (input i, forget f and output o), an update step g, a cell memory state c, and a hidden state h. LSTM cell computation at time t, for input x, is given as [17]:

$$i_{[t]} = \sigma(W_{xi}x_{[t]} + b_{xi} + W_{hi}h_{[t-1]} + b_{hi}) \tag{2a}$$
$$f_{[t]} = \sigma(W_{xf}x_{[t]} + b_{xf} + W_{hf}h_{[t-1]} + b_{hf}) \tag{2b}$$
$$g_{[t]} = \tanh(W_{xg}x_{[t]} + b_{xg} + W_{hg}h_{[t-1]} + b_{hg}) \tag{2c}$$
$$o_{[t]} = (W_{xo}x_{[t]} + b_{xo} + W_{ho}h_{[t-1]} + b_{ho}) \tag{2d}$$
$$c_{[t]} = f_t \odot c_{[t-1]} + i_t \odot g_t \tag{2e}$$
$$h_{[t]} = o_t \odot \tanh(c_{[t-1]}) \tag{2f}$$

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.

To simplify the LSTM model, the Gated Recurrent Unit (GRU) was recently introduced [18]. GRU merges the memory state and hidden state into a single hidden state and combines the input and forget gates into an update gate. As

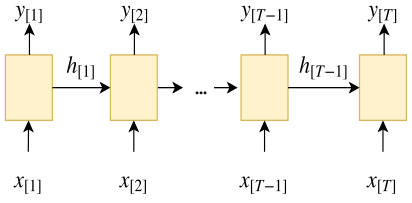

FIGURE 1. Recurrent neural network.

GRUs have fewer parameters, convergence is achieved faster than with LSTMs; nevertheless, GRUs contain sufficient gates and states for long-term memory retention.

B. SEQUENCE TO SEQUENCE RNNS

Sequence to Sequence (S2S or Seq2Seq) RNNs [12] consist of an encoder and decoder RNN as illustrated in Fig. 2. A sequence $x_{[1]}, ..., x_{[T]}$ is passed into the encoder RNN, one time step at a time, to obtain a context vector $\vec{c}$. A common approach is to use an RNN such that:

$$h_{[j]} = f^*(x_{[j]}, h_{[j-1]}) \tag{3}$$

$$\vec{c} = q(\{h_{[1]}, ..., h_{[T]}\}) \tag{4}$$

where $h_{[j]}$ is the hidden state at time j, f^* is some non-linear functions, and $q = h[T]$ is usually the last hidden state produced by the encoder RNN [12].

The context vector is an encoded representation of the input sequence that is passed to the decoder RNN which extracts information at each unraveled time step to obtain the output sequence $\dot{y}_{[1]}, ..., \dot{y}_{[N]}$. The S2S output is obtained as:

$$\dot{y}_{[i]} = g^*(\dot{y}_{[i-1]}, {h^*}_{[i-1]}) \tag{5}$$

where $h^*_{[i]}$ is the decoder hidden state at time i, and g^* is some non-linear function. Note that $\dot{y}_{[0]}$ is the context value (derived from $\vec{c}$) used as the initial input for the decoder.

The use of two RNNs strengthens consecutive sequence prediction, while also allowing the time dimensionality of inputs and outputs to vary [12]. Although load forecasting does not require varying lengths, it can benefit from strong consecutive sequence prediction.

C. ATTENTION MECHANISM

In S2S models, the encoder is responsible for compressing all significant information of an input sequence into a single context vector $\vec{c}$. For longer input sequences, the decoder may have difficulties extracting valuable information from this single vector; thus, *attention* mechanism was added. Bahdanau *et al.* [13] and Luong *et al.* [14] both added attention-based architectures to S2S models for neural machine translation. Language translation is very different task than load forecasting; nevertheless, load forecasting could benefit from the attention mechanisms. Our work adopts Bahdanau and Luong attentions for load forecasting.

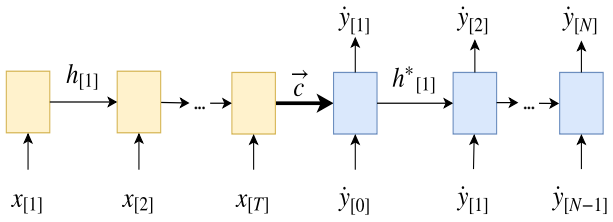

FIGURE 2. S2S RNN.

The Bahdanau [13] attention (BA) mechanism was the first form of attention for S2S models. With BA, encoder hidden states $h^b_{[i]}$ and outputs $\dot{y}_{[i]}$ at time step i are calculated as:

$$h^b_{[i]} = f^b([\dot{y}_{[i-1]}; c^b_{[i]}], h^b_{[i-1]}) \tag{6}$$

$$\dot{y}_{[i]} = g^b(\dot{y}_{[i-1]}, c^b_{[i]}, h^b_{[i]}) \tag{7}$$

where superscript ^{b} indicates BA variables, $[\cdot]$ represents concatenation, f^b denotes a Vanilla RNN, LSTM, or GRU cell, and g^b some non-linear function. The vector $c^b_{[i]}$ is computed as a weighted sum of the encoder hidden states:

$$c^b_{[i]} = \sum_{j=1}^{T} \alpha^b_{[ij]} h_{[j]} \tag{8}$$

where the attention weight $\alpha^b_{[ij]}$ between the output at time i and input at time j is computed as:

$$\alpha^b_{[ij]} = \frac{\exp(e^b_{[ij]})}{\sum_{m=1}^{T} \exp(e^b_{[im]})} \tag{9}$$

The attention energies $e^b_{[ij]}$ are:

$$e^b_{[ij]} = S(h^b_{[i-1]}, h_{[j]}) \tag{10}$$

Here, S is is an *alignment model* which scores how well the inputs around time j and the output at time i match.

The attention weight $\alpha^b_{[ij]}$, and its associated energy $e_{[ij]}$, reflect the importance of each encoder state $h_{[j]}$ 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.

Luong *et al.* [14] developed global and local attention-based models for machine translation, differing whether the attention is concentrated on a few input positions or on all. In remainder of our paper, Luong attention (LA) will refer to the global model. The main difference between BA and LA is that BA applies the attention mechanism before the variables are passed through the respective RNN cell, while LA applies the mechanism to the outputs of that respective cell. Luong *et al.* [14] presented attention variants differing in the score functions:

$$S(h^l_{[i]}, h_{[j]}) = \begin{cases} {h^l_{[i]}}^\intercal h_{[j]} & dot \\ {h^l_{[i]}}^\intercal W(h_{[j]}) & general \\ v^\intercal \tanh(W(cat(h^l_{[i]}, h_{[j]}))) & concat \end{cases} \tag{11}$$

where v is a parameter learned with the rest of the system. Ultimately, both LA and BA serve the same purpose: to relieve the encoder of having to compress all information from the input sequence into a single fixed-length vector.

III. RELATED WORK

This section discusses related load forecasting works as well as the S2S models in other domains.

A. LOAD FORECASTING

Load forecasting can be classified into three main categories: short, medium, and long-term [19] [20]; however, there is no clear distinction between those categories. In our work, *short-term* is considered the next hour, *medium-term* refers to next few hours to a day ahead, and *long-term* implies a day or more 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. Support Vector Regression (SVR) and NN have been very popular: several studies considered NN and SVM models for estimating energy loads [9], [21] and some compared their performance [9]. The accuracy and conclusions varied depending on data sets, features, system architectures, and similar. As the field of neural networks and deep learning has been evolving fast, several NN-based models have been proposed. Jetcheva *et al.* [22] proposed a NN model for day-ahead building-level load forecasting with an ensemble-based approach for parameters selection whereas Chae *et al.* [23] considered a NN model with Bayesian regularization algorithm. Araya *et al.* [24] proposed an ensemble framework for anomaly detection in building energy consumption; they included prediction-based classifiers (SVR and random forest) as their base forecasting models. Convolutional Neural Networks (CNN) have also been used for load forecasting [25]; they outperform SVM models while achieving comparable results to NN and other deep learning methods [26].

Recently, RNNs have been gaining popularity for load forecasting because of their ability to capture time dependencies in data. Kong *et al.* [27] proposed an LSTM based RNN model for short-term residential load forecasting. Likewise, Shi *et al.* [28] also focused on short-term forecasting; they proposed a novel pooling based deep recurrent neural network (PDRNN) for residential consumers. Short to medium term aggregate load forecasting was considered by Bouktif *et al*. [29]; they coupled a standard LSTM model with a genetic algorithm (GA). Yu *et al.* [30] combined GRU with dynamic time warping (DTW) for daily peak load forecasting. A time-dependency convolutional neural network (TD-CNN) and a cycle-based long short-term memory (C-LSTM) network have also been used to improve accuracy of short-term load forecasting [31]

As can be seen from recent works on load forecasting [27], [28], [30], RNNs have been outperforming other approaches. Our work differs by focusing on S2S RNN which have shown great success in modeling time-dependencies in language translation. Marino *et al*. [19] used standard LSTM and LSTM-based S2S models for residential load forecasting; our work differs by means of different sample generation, different connection of encoder and decoder, use of attention mechanisms, and a longer prediction sequence length. Zheng *et al.* [32] proposed a hybrid algorithm that combines similar days (SD) selection, empirical mode decomposition (EMD), and LSTM neural networks. Whereas Zheng *et al.* [32] work proposed a unique hybrid model, the S2S LSTM-based model is identical to the one used by Marino *et al.* [19]. Rahman *et al.* [33] developed two S2S LSTM-based models for medium to long-term forecasting. Our work differs from all three S2S works [19], [32], [33] by means of different sample generation, different connection of encoder and decoder, and use of attention mechanisms. In our previous work [34], we presented initial results on S2S RNN for energy forecasting; in contrast, this work focuses on adding attention mechanisms to the S2S models and evaluating their performance with different RNN cells and forecasting lengths.

B. SEQUENCE-TO-SEQUENCE MODELS

Sequence to Sequence models have been used not only for load forecasting, but also for a number of other tasks. Also known as encoder-decoder RNNs, these models have become increasingly popular in tackling classification problems. They were originally developed by Cho *et al.* [18] to improve performance of statistical machine translation (SMT). This same work not only proposed a novel model architecture, but also a novel RNN cell structure, later to become known as the GRU unit. The work by Sutskever *et al.* [12] introduced a slight variation to the RNN S2S framework, for translation from English to French. In contrast to Cho *et al.* [18], Sutskever *et al.* [12] used LSTM in place of GRU; moreover they also differ in how they connect encoder and decoder.

Examples of S2S use in other domains include the work of Venugopalan *et al.* [35] on LSTM-based S2S models for generating descriptions of real-world videos and the work of Kawano *et al.* [36] on predicting changes in protein stability.

While S2S RNN models have found success in several domains, the encoder in S2S is burdened with the need to represent all information in a fixed-length vector. Thus, an attention mechanism, otherwise known as an “alignment model", was added to these models by Bahdanau *et al.* [13] and also by Luong *et al.* [14]: these mechanisms have been described in Section II. Both were designed for machine translations, whereas our work adapts them for load forecasting. Moreover, we evaluate performance of different attention models with different cell types and different time horizons.

IV. METHODOLOGY

This section first introduces the features and evaluation process. Next, the sample generation and the proposed BA and LA S2S RNNs for load forecasting are described.

A. FEATURES AND EVALUATION PROCESS

Data sets obtained from smart meters typically contain the reading date and time with corresponding energy consumption. From those attributes, additional features are extracted and the resulting data set contains nine features: month, day of year, day of month, weekday, weekend, holiday, hour, season, and energy usage.

The data set was divided into a training and test set: the first 80% of data was used for training and the last 20% for testing. This validation process was chosen to ensure that the model is built using older data and tested on newer data.

Standardization was applied to bring all variables into similar ranges. The values of each feature in the data were transformed to have zero-mean and unit-variance:

$$\tilde{x} = \frac{x - \mu}{\sigma} \tag{12}$$

where 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

Sample generation here refers to the process of transforming data into the input and target samples to be passed to the ML model. The same approach is used as in Sehovac *et al.* [34]. An input sample is 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. Each input sample is defined as:

$$\begin{aligned} x_{[j]} = [&\text{Month}_{[j]}\ \text{DayOfYear}_{[j]}\ \text{DayOfMonth}_{[j]} \\ &\text{Weekday}_{[j]}\ \text{Weekend}_{[j]}\ \text{Holiday}_{[j]}\ \text{Hour}_{[j]} \\ &\text{Season}_{[j]}\ \text{Usage}_{[j]}] \end{aligned} \tag{13}$$

For each input sample, one target sample $y \in \mathbb{R}^{N \times 1}$ is generated, where N is the number of predicted time steps. The target vector is given by:

$$y = [\text{Usage}_{[1]},\ \ldots,\ \text{Usage}_{[i]},\ \ldots,\ \text{Usage}_{[N]}] \tag{14}$$

where $y_{[i]}$ is the actual usage value at time step i, and $i \in 1, \ldots, N$.

Fig. 3 illustrates the sample generation process for training set. T denotes the length of the input window and N denotes the length of the target vector. For each time step i of the training set, consecutive data from $i+1$ to $i+T$ forms an input sample and data from $i+T+1$ to $i+T+N$ makes the corresponding target vector. After the samples are generated, their order is randomized.

For the test set, samples are generated somewhat differently. As illustrated in Fig. 4, the sliding windows for the test set shifts sequentially with the overlap equivalent to the target length N. Similar to the training set, if the input sample

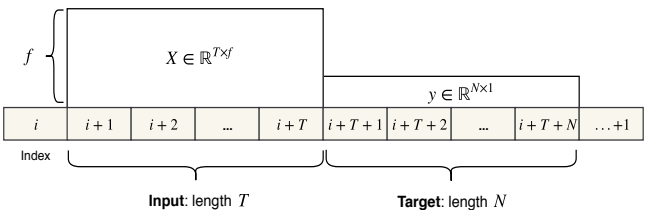

FIGURE 3. Training set sample generation.

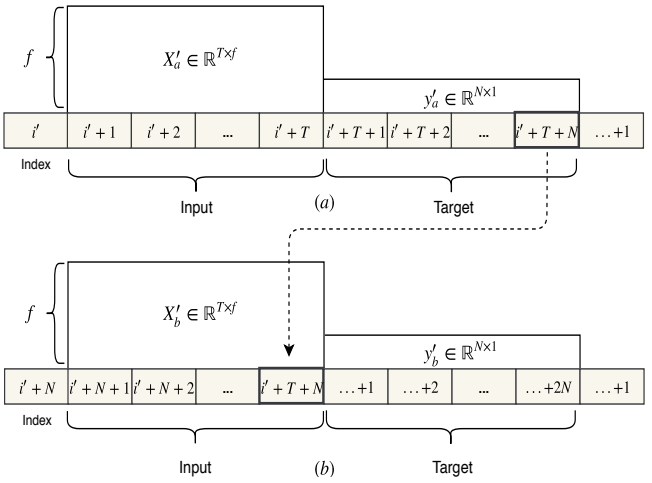

FIGURE 4. Test set sample generation. (a) Samples generated at index $i' + 1$. (b) Samples generated from sliding window with overlap length N.

starts at $i' + 1$, the target sample still ends at $i' + T + N$; however, the difference is that the next sample will start at $i'+N+1$. This way prediction vectors from different samples do not overlap. Sliding each time by N allows for an overlap in input test samples, but there is no overlap in target test samples. Consequently, the predicted usage vectors can be concatenated into one vector with the length equivalent to the number of time steps in the test set.

C. S2S PREDICTION WITH BA

S2S load forecasting proposed by Sehovac *et al.* [34] is augmented by adding Bahdanau Attention (BA). Whereas Section II-C gives a generic breakdown of the BA mechanism, here we give the process of adapting BA to S2S models for load forecasting. The overall process is illustrated in Fig. 5 and details are provided in Algorithm 1.

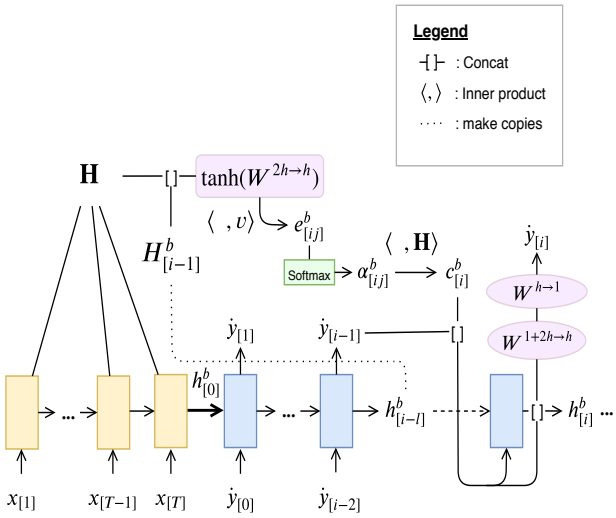

FIGURE 5. The BA mechanism at decoder time step i.

The encoder process, lines 14-22 of Algorithm 1, are the same as in Sehovac *et al.* [34] since an identical encoder (E_T) approach is taken, except in Algorithm 1 the encoder output weights are initialized and stored in lines 15 and 19 respectively. Let us denote these encoder outputs as $H = \left[h_{[1]}, ..., h_{[T]}\right] \in \mathbb{R}^{T \times h}$ where each output can be defined as $H_{[j]} = h_{[j]} \in \mathbb{R}^h$. Dimensions T and h are the length of the input sequence and the hidden state dimension, 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 decoder D_N, and the encoder outputs H.

The context vector is used as the initial hidden state of decoder D_N, so we denote it as $h^b_{[0]}$. Therefore, at decoder time step i, 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 created; this matrix is denoted as $H^b_{[i-1]} \in \mathbb{R}^{T \times h}$. As in lines 25-27 of Algorithm 1, the energies are computed by:

$$\lambda_1 = [H^b_{[i-1]}; H] \in \mathbb{R}^{T \times 2h} \tag{15a}$$

$$\lambda_2 = \tanh(W^{2h \to h}(\lambda_1)) \in \mathbb{R}^{T \times h} \tag{15b}$$

$$e^b_{[ij]} = \langle \lambda_2, v \rangle \in \mathbb{R}^T, \text{where } v \in \mathbb{R}^h \tag{15c}$$

Here, $W^{2h \to h}$ and v are parameters to be learned during training, $\langle \cdot \rangle$ denotes inner product, $W^{2h \to h}$ is a fully-connected layer and v is a vector randomly initialized from $\mathcal{N}(0, \frac{1}{\sqrt{h}})$. Note that the $[\cdot]$ notation is used to indicate concatenation, while the λs in equations (15), and in subsequent equations are used as placeholders to denote the steps taken to produce respective outputs and to separate steps illustrated in Fig. 5.

Concatenation is performed first (15a), followed by a fully-connected layer and activation function $tanh$ (15b), and finally the inner product is calculated (15c). Equations (15) also provide dimensions for each step: for example, in equation (15c), λ_2 has dimensions $T \times h$. 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 (Algorithm 1, line 28) is to compute the attention weights $\alpha^b_{[ij]} \in \mathbb{R}^T$:

$$\alpha^b_{[ij]} = \frac{\exp(e^b_{[ij]})}{\sum_{k=1}^{T} \exp(e^b_{[ik]})} \tag{16}$$

Equation (16) is the Softmax of the attention energies computed in (15c). These attention weights are then used in an inner product with H (Algorithm 1, line 29), to compute the context vector $c^b_{[i]} \in \mathbb{R}^h$ (Algorithm 1, line 30) as:

$$c^b_{[i]} = \langle \alpha^b_{[ij]}, \mathbf{H} \rangle \tag{17a}$$

$$= \sum_{j=1}^{T} \alpha^b_{[ij]} h_{[j]} \tag{17b}$$

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 (Algorithm 1, line 30), to obtain the current hidden state $h^b_{[i]}$:

$$\lambda_3 = [\dot{y}_{[i-1]}; c^b_{[i]}] \in \mathbb{R}^{1+h} \tag{18a}$$

$$h^b_{[i]} = \text{GRU}_{Cell}(\lambda_3, h^b_{[i-1]}) \in \mathbb{R}^h \tag{18b}$$

where the GRU_{Cell} takes λ_3 and the previous hidden state $h^b_{[i-1]}$ as inputs.

The last step is to pass the current hidden state, the context vector, and the previous output through a function as given

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

1: *Input* : Model G consisting of: E_T, D_N, activation
2: functions tanh and Softmax, fully-connected layers
3: $W^{2h \to h}, W^{1+2h \to h}, W^{h \to 1}$, vector parameter v,
4: and initial weights P.
5: *Output* : Trained model G
6:
7: Generate input samples $X \in \mathbb{R}^{T \times f}$
8: and corresponding target vectors $y \in \mathbb{R}^{N \times 1}$
9: Initialize $v \sim \mathcal{N}(0, \frac{1}{\sqrt{h}}) \in \mathbb{R}^h$
10:
11: *for* each *epoch* **do**
12: *for* each *batch* **do**
13:
14: initialize $h_{[j-1]} \leftarrow \vec{0} \in \mathbb{R}^{B \times h}$, $B =$ batch size
15: initialize $H \leftarrow \vec{0} \in \mathbb{R}^{T \times B \times h}$
16:
17: *for* time step j **from** 1 **to** T **do** # Encoder
18: $h_{[j]} \leftarrow \text{GRU}_{cell}(x_{[j]}, h_{[j-1]})$
19: $H_{[j]} \leftarrow h_{[j]}$
20:
21: $\dot{y}_{[0]} \leftarrow W^{h \to 1}(h_{[T]})$
22: $h^b_{[0]} \leftarrow h_{[T]}$
23:
24: *for* time step i **from** 1 **to** N **do** # Decoder
25: $H^b_{[i-1]} \leftarrow T$ copies of $h^b_{[i-1]}$
26: $e^b_{[ij]} \leftarrow \tanh(W^{2h \to h}([H^b_{[i-1]}; H]))$
27: $e^b_{[ij]} \leftarrow \langle e^b_{[ij]}, v \rangle^\intercal$
28: $\alpha^b_{[ij]} \leftarrow \text{Softmax}(e^b_{[ij]})$
29: $c^b_{[i]} \leftarrow \langle \alpha^b_{[ij]}, H^\intercal \rangle$
30: $h^b_{[i]} \leftarrow \text{GRU}_{Cell}([\dot{y}_{[i-1]}; c^b_{[i]}], h^b_{[i-1]})$
31: $\dot{y}_{[i]} \leftarrow W^{h \to 1}(W^{1+2h \to h}([\dot{y}_{[i-1]}; c^b_{[i]}; h^b_{[i]}]))$
32: append $\dot{y}_{[i]}$ to predicted usage vector $\dot{y}$
33:
34: compute $loss$
35: BPTT($loss$)
36: $P \leftarrow \text{Adam}(P, r)$, $r =$ learning rate
37:
38: *Return* : Trained model G.

in equation (7). In this work, we concatenate all three variables and pass this vector through two fully-connected layers (Algorithm 1, line 31):

$$\lambda_4 = [\dot{y}_{[i-1]}; c^b_{[i]}; h^b_{[i]}] \in \mathbb{R}^{1+2h} \tag{19a}$$
$$\lambda_5 = W^{1+2h \to h}(\lambda_4) \tag{19b}$$
$$\dot{y}_{[i]} = W^{h \to 1}(\lambda_5) \in \mathbb{R}^1 \tag{19c}$$

where $W^{1+2h \to h}$ and $W^{h \to 1}$ are parameters to be learned during training.

This concludes the process for one time step and after N time steps, we have obtained the predicted usage vector $\dot{y} \in \mathbb{R}^N$, and we can now compute the loss. The remaining steps, BPTT using gradient based method to update the weights, are identical as in traditional S2S [34].

D. S2S PREDICTION WITH LA

This section explains the process of adapting LA to S2S RNN for load forecasting: Fig. 6 provides the overview whereas Algorithm 2 shows details. The encoder takes the identical approach as in BA; lines 12-20 in Algorithm 2 are identical to lines 14-22 of Algorithm 1, respective to their own variables. Hence, at time step T, encoder obtained $h_{[T]}$, $\dot{y}_{[0]}$, and H. Let us denote LA variables with the l superscript notation.

The key difference between BA and LA is that in BA energies are computed first whereas the first step in LA is to compute the current hidden state $h^l_{[i]} \in \mathbb{R}^h$. This hidden state is calculated taking as inputs the previous predicted output and previous hidden state; for the decoder time step i, this is calculated as (Algorithm 2, line 23):

$$h^l_{[i]} = \text{GRU}_{Cell}(\dot{y}_{[i-1]}, h^l_{[i-1]}) \tag{20}$$

This hidden state $h^l_{[i]}$ is then used in the LA mechanism. Hence, the LA mechanism is applied using the current hidden state counter to the previous hidden state used in BA (equation (15)).

The next step is to compute the attention energies $e^l_{[ij]}$ using one of three score functions in equation (11). Note that in equation (11), when used with LA, only one score function is chosen at a time, and this constitutes one model. If using *dot* score function, energies $e^l_{[ij]}$ are computed as:

$$e^l_{[ij]} = \langle H, h^l_{[i]} \rangle \in \mathbb{R}^T \tag{21}$$

Here, the dot product is computed between $h_{[j]}$ and $h^l_{[i]}$, and the scalar value is stored in the j-th element of $e^l_{[ij]}$. Instead, if the *general* score function is chosen, the attention energies are computed as in line 24 of Algorithm 2:

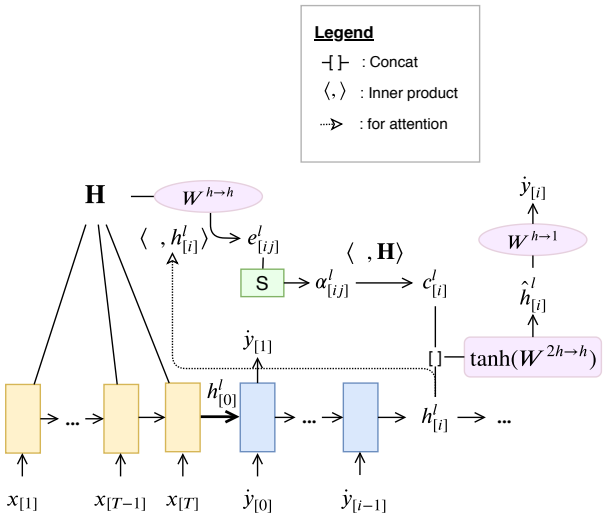

FIGURE 6. The LA mechanism at time step i, using the *general* score function.

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

1: *Input* : Model G consisting of: E_T, D_N, activation functions
2: tanh and Softmax, fully-connected layers
3: $W^{2h\to h}, W^{1+2h\to h}, W^{h\to 1}$, and initial weights P.
4: *Output* : Trained Model G.
5:
6: Generate input samples $X \in \mathbb{R}^{T\times f}$ and
7: and corresponding target vectors $y \in \mathbb{R}^{N\times 1}$
8:
9: *for* each *epoch* **do**
10: *for* each *batch* **do**
11:
12: initialize $h_{[j-1]} \leftarrow \vec{0} \in \mathbb{R}^{B\times h}$, B = batch size
13: initialize $H \leftarrow \vec{0} \in \mathbb{R}^{T\times B\times h}$
14:
15: *for* time step j **from** 1 **to** T **do** # Encoder
16: $h_{[j]} \leftarrow \text{GRU}_{Cell}(x_{[j]}, h_{[j-1]})$
17: $H_{[j]} \leftarrow h_{[j]}$
18:
19: $\dot{y}_{[0]} \leftarrow W^{h\to 1}(h_{[T]})$
20: $h^l_{[0]} \leftarrow h_{[T]}$
21:
22: *for* time step i **from** 1 **to** N **do** # Decoder
23: $h^l_{[i]} \leftarrow \text{GRU}_{Cell}(\dot{y}_{[i-1]}, h^l_{[i-1]})$
24: $e^l_{[ij]} \leftarrow \langle W^{h\to h}(H), h^l_{[i]}\rangle^\intercal$
25: $\alpha^l_{[ij]} \leftarrow \text{Softmax}(e^l_{[ij]})$
26: $c^l_{[i]} \leftarrow \langle \alpha^l_{[ij]}, H^\intercal\rangle$
27: $\hat{h}^l_{[i]} \leftarrow \tanh(W^{2h\to h}(\text{cat}(h^l_{[i]}, c^l_{[i]})))$
28: $\dot{y}_{[i]} \leftarrow W^{h\to 1}(\hat{h}^l_{[i]})$
29: append $\dot{y}_{[i]}$ to predicted usage vector $\dot{y}$
30:
31: compute $loss$
32: BPTT($loss$)
33: $P \leftarrow \text{Adam}(P, r)$, r = learning rate
34:
35: *Return* : Trained model G.

$$\lambda_6 = W^{h\to h}(H) \in \mathbb{R}^{T\times h} \tag{22a}$$
$$e^l_{[ij]} = \langle \lambda_6, h^l_{[i]} \rangle \in \mathbb{R}^T \tag{22b}$$

Note that the *dot* and *general* score functions are very similar, differing only by the *general* score adding a fully-connected layer 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 (22a).

The remaining score function, *concat*, computes the energies similarly as in the equations (15). However, the difference here is that LA use the matrix $H^l_{[i]} \in \mathbb{R}^{T\times h}$, T copies of $h^l_{[i]}$, instead of $H^b_{[i-1]}$ used in BA.

Continuing (line 25, Algorithm 2), shows that the attention weights $\alpha^l_{[ij]} \in \mathbb{R}^T$ are computed using equation (16), except with attention energies $e^l_{[ij]}$ in place of $e^b_{[ij]}$. The attention context vector $c^l_{[i]} \in \mathbb{R}^h$ is then computed in line 26 using the equations (17) with the respective attention weights $\alpha^l_{[ij]}$.

The next step (Algorithm 2, line 27) is to compute the attentional hidden states $\hat{h}^l_{[i]}$:

$$\lambda_7 = [h^l_{[i]}; c^l_{[i]}] \in \mathbb{R}^{2h} \tag{23a}$$
$$\hat{h}^l_{[i]} = \tanh(W^{2h\to h}(\lambda_7)) \in \mathbb{R}^h \tag{23b}$$

Lastly, the predicted output value is obtained by passing $\hat{h}^l_{[i]}$ through a fully-connected layer (line 28):

$$\dot{y}_{[i]} = W^{h\to 1}(\hat{h}^l_{[i]}) \tag{24}$$

where $W^{2h\to h}$ and $W^{h\to 1}$ are parameters to be learned during training.

Note that the attentional hidden state and current hidden state have different functions in LA. The attentional hidden state $\hat{h}^l_{[i]}$ is only used in computing the output value, equation (24), while the current hidden state $h^l_{[i]}$ is used to compute the attention context vector (equation (17)), and as 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 in LA algorithm are equivalent to those in the BA algorithm.

V. EVALUATION

The proposed approach was evaluated on a real-world dataset from a commercial building provided by an industry partner. The dataset contained one year and three months of energy load data in five minute intervals for a total number of readings: 12 readings in one hour $\times$ 24 hours in one day $=$ 287 $\times$ 458 days $=$ 131,446. As stated in section IV, the data were pre-processed and the resulting set contained nine features. The rest of this section describes the conducted experiments, presents their results, and discusses the findings.

A. EXPERIMENTS

All S2S models were tested for four different prediction lengths N given as elements of vector $\vec{N}$:

$$\vec{N} = [12,\ 48,\ 120,\ 288] \tag{25}$$

For five-minute reading intervals, this equates to predicting the next [1 hour, 4 hours, 10 hours, 24 hours]. There are endless combinations of input length T to prediction length N; the following **four input cases** were chosen for experiments:

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

The four input cases with four prediction lengths, makes for a total of 16 cases considered. All models were trained for 10 epochs, since this was sufficient to reach an acceptable level of convergence. 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

Only one h, and equivalent dimension c for LSTM, was 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. With each of the 16 cases, 21 different models were evaluated:

- S2S-o model from the work of Sehovac *et al.* [34] with GRU/LSTM/RNN cells (3 models)
- S2S-BA model with GRU/LSTM/RNN cells (3 models)
- S2S-LA model with GRU/LSTM/RNN cells. Accompanied with each cell is one of three attention score functions: *dot, general, concat* (9 models)
- Non-S2S RNN model with GRU/LSTM/RNN cell (3 models)
- DNN model with sizes: small, medium, and large (3 models)

We define a **model type** as one of the seven listed above (three LA score functions are considered separately) and a **model** as a combination of model type and cell type. Each model is different for each of the 16 cases in terms of input length T and prediction length N.

The Non-S2S RNN model is a conventional RNN model, and thus cannot have a prediction length longer than input sequence length. Hence, this model is only used when $N \leq T$. For example, for input case 3, input length $T = 120$, the Non-S2S RNN can only predict for 12, 48, and 120 steps ahead.

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 into a single vector of dimension size $= T \times f$. Hence, each input vector $x_{[j]}$ of X was now side-by-side constituting one long vector. The three considered DNN sizes are:

- DNN-small: Input layer (size $T \times f$) $\bowtie$ 512 $\bowtie$ 256 $\bowtie$ 128 $\bowtie$ Output layer (size N)
- 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

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. Multiple layers of the same size were used to add more parameters, to see whether increasing parameters improved the accuracy.

The accuracy measures used throughout this work are the Mean Absolute Error (MAE) and Mean Absolute Percentage Error (MAPE):

$$\text{MAE} = \frac{1}{n}\sum_{i=0}^{n}|y_i - \hat{y}_i| \tag{26}$$

$$\text{MAPE} = \frac{100\%}{n}\sum_{i=0}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right| \tag{27}$$

where y represents the actual value, $\hat{y}$ the predicted value, and n the number of samples. Note that the MAE and MAPE were calculated with the unnormalized values; the predicted usage vector obtained in the normalized space was converted to the original domain space for the accuracy calculations.

Since this work randomizes the training samples and uses randomly initialized initial weights, five random seeds were used for each case so that each model sees a different randomized order of training samples and different initial weights.

B. RESULTS AND DISCUSSION

The proposed approach was implemented in Python with the PyTorch tensor library [37]. Experiments were conducted using GPUs on two different machines: the first contained two NVIDIA GeForce RTX 2080 Ti GPU cards, and the second contained one NVIDIA GeForce GTX 1060 GPU card.

The following four subsections present results obtained for the four input cases. The first figure in each of the four subsections shows the best achieved results by each model type and compares the performance of each model type, regardless of cell or hidden dimension size h. This figure also identifies the overall best performing model.

The second figure in each input case subsections analyzes the performance of cells used in the models and compares cell performance as input length T is kept stationary while prediction length N varies. The DNN models are not included in this figure as DNNs do not deal with RNN cell variation.

1) Input Case 1: One Hour

This subsection analyzes the results obtained for input length $T = 12$. Fig. 7 shows the best achieved MAPE and MAE by each model type, for each prediction length. The best obtained MAPE and MAE for each prediction N are indicated in bold. The lowest accuracy was observed with the Non-S2S RNN model, even the DNN model achieved better results. Furthermore, it can be seen that as N increases, the accuracy of each model decreases with the accuracy of the DNN model decreasing at a much greater rate than the accuracy of other models.

It is important to note that the S2S-o models perform comparable to the attention models. 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 best MAPE for each input length, as can be seen by the bold numbers in Fig. 7.

Fig. 8 shows how accuracy changes with increase of prediction length N for each model type and each cell: Vanilla RNN, GRU, and LSTM. For all cells, and all model types, the accuracy decreases as the prediction length N increases. The Non-S2S RNN model produced results only for $N = 12$, and for each cell, this model has the lowest accuracy.

The Vanilla RNN cell achieved comparable results with the 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 $N = 288$.

For the GRU cell, results are similar among all models across all prediction lengths, with the majority of attention models performing slightly better than the S2S-o model 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 $N = 288$.

MAE results show the same patterns as MAPE for all four input cases, therefore, MAE graphs are not included with the remaining three cases.

2) Input Case 2: Four Hours

This subsection analyzes the results obtained for input length $T = 48$. Fig. 9 shows the best results achieved by each model type, for each prediction length. Similarly, as shown in Fig. 7, the Non-S2S RNN model performs the worst for $N = 12$, but does perform better than the DNN model for $N = 48$. 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 input case 1: the best achieved MAPE for each prediction length is obtained by one of the attention models.

The comparison of the DNN model results between Fig. 7 and Fig. 9 shows that the accuracy slightly increases as input length T increases. Nonetheless, as N increases in Fig. 9, the DNN models are significantly outperformed by all S2S models.

Fig. 10 analyzes performance of different cells. It can be seen that for all cells the Non-S2S RNN model performs the worst when $N \leq T$. The Vanilla RNN cell, similarly as in Fig. 8, 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. 8 shows the S2S-LA-general

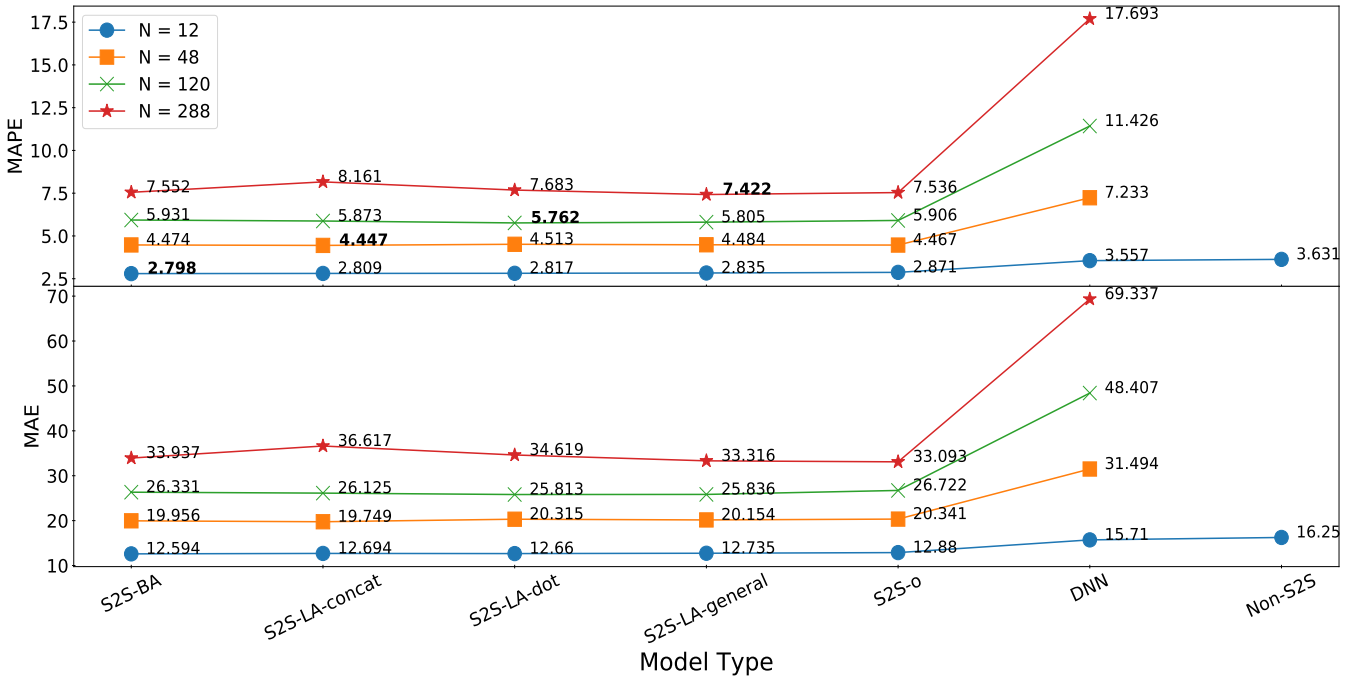

FIGURE 7. Best achieved results by each model type: input $T = 12$, all prediction lengths N.

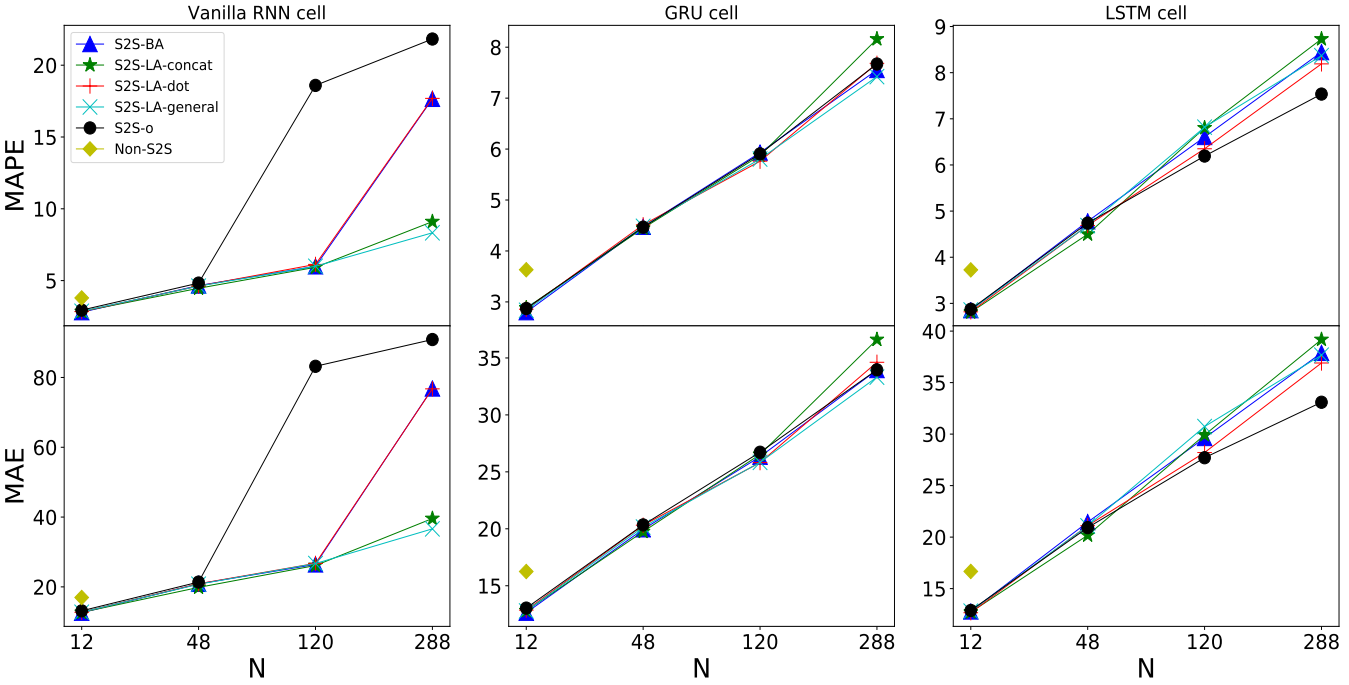

FIGURE 8. Comparing cell performance of each non-DNN model: input $T = 12$, all prediction lengths N.

model as the best Vanilla RNN model for $N = 288$ whereas in Fig. 10, the S2S-LA-dot model shows better accuracy.

Like with the Vanilla cell, the GRU cell results in Fig. 10 show the majority of attention models outperforming the S2S-o model as N increases. The opposite happens for the LSTM cell: the S2S-o model outperforms the attention models as N increases. The LSTM cell contains two states, the hidden state and the cell state, and only one state, the hidden state, is used in the attention mechanisms. Hence, the use of only one internal LSTM state in attention models could be the underlying reason as to why the S2S-o models outperformed the attention models, for increasing N.

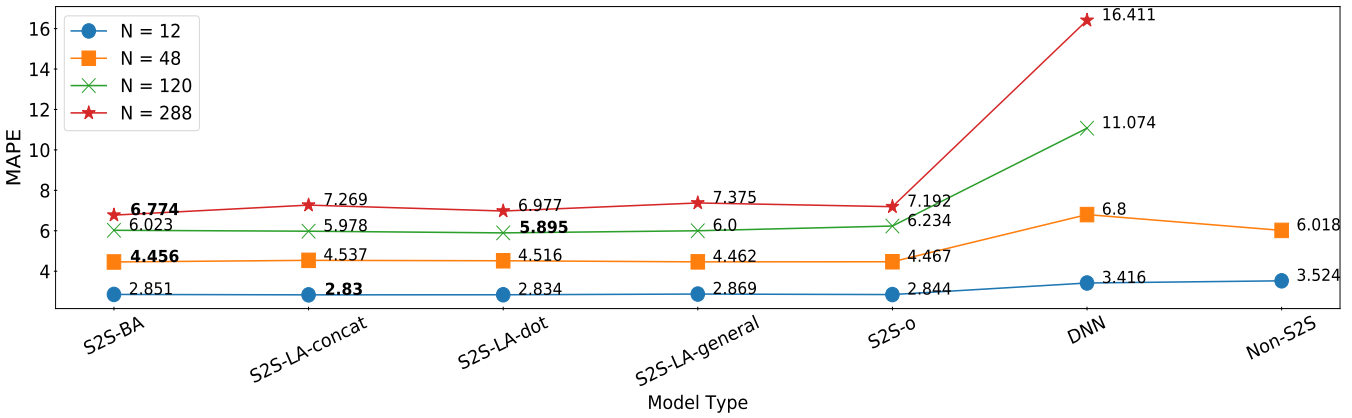

FIGURE 9. Best achieved results by each model type: input $T = 48$, all prediction lengths N.

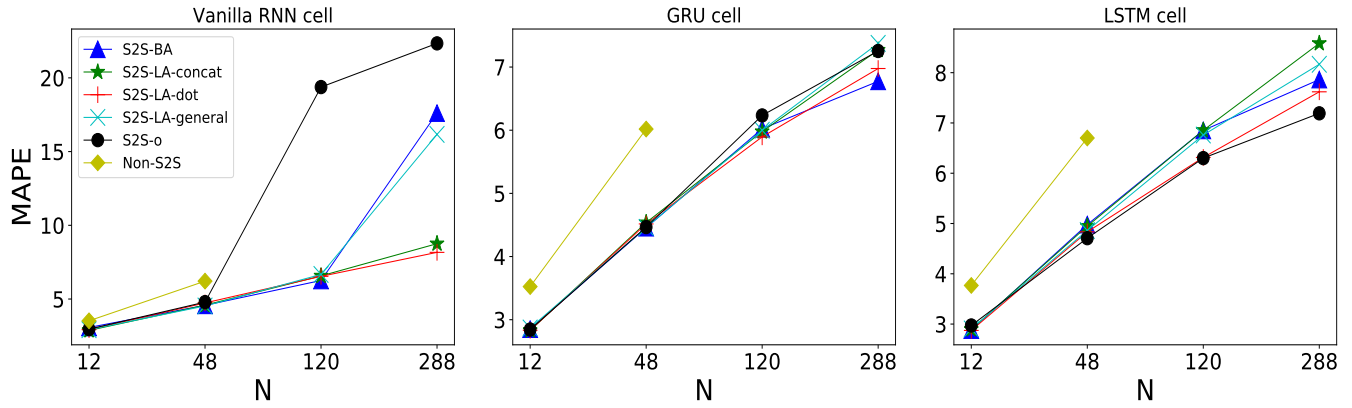

FIGURE 10. Comparing cell performance of each non-DNN model: input $T = 48$, all prediction lengths N.

3) Input Case 3: Ten Hours

This subsection provides the results obtained for input length $T = 120$. Fig. 11, shows a re-occurring pattern seen in cases 1 and 2, figures 7 and 9: the attention models obtain the best MAPE across all prediction lengths

In Case 3, as can be observed from Fig. 12, for the Vanilla RNN cell, the Non-S2S RNN model obtains better results than the S2S-o model for $N = 120$. 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. The Vanilla RNN-based S2S attention models outperformed the S2S-o model for $N = 120$ and $N = 288$, similar to findings for input cases 1 and 2 (figures 8 and 10).

For the GRU and LSTM models, the Non-S2S RNN model performs the worst for all prediction lengths. Similar to the GRU and LSTM cell results for input cases 1 and 2, 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 all input lengths.

4) Input Case 4: Twenty-Four Hours

For this case, the input length is $T = 288$. The best results by each model type are shown in Fig. 13. The DNN model achieved worse results than the S2S models, but did outperform the Non-S2S RNN model for each prediction length. Nonetheless, the DNN model results in case 4 (Fig. 13) are a significant improvement from those in cases 1 to 3 (figures 7, 9, and 11). Unlike in the previous input cases, the S2S-o model achieved the best MAPE for $N = 120$ and $N = 288$ indicating that using 288 values in the attention mechanisms may not be necessary.

With the Vanilla RNN cells, the Non-S2S RNN model performed the best for $N = 288$ what is the same observation as in input case 3: an Encoder-Decoder + Vanilla RNN cell model does not retain information better than a model consisting of a sole Encoder + Vanilla RNN cell.

As in input case 3, the GRU and LSTM Non-S2S RNN models performed the worst for all prediction lengths. As in other input cases, the LSTM-based S2S-o model outperformed all other LSTM models for $N = 288$. It is important to note that, unlike in cases 1, 2, and 3 (figures 8, 10, and 12), in case 4, the GRU-based S2S-o models outperformed all other GRU models for $N = 288$. Hence, for case 4, all

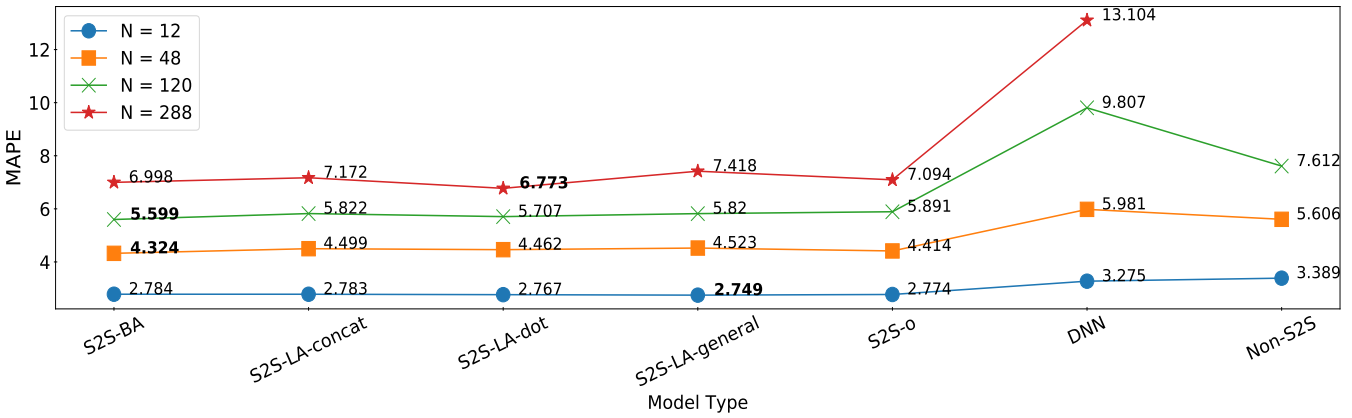

FIGURE 11. Best achieved results by each model type: input $T = 120$, all prediction lengths N.

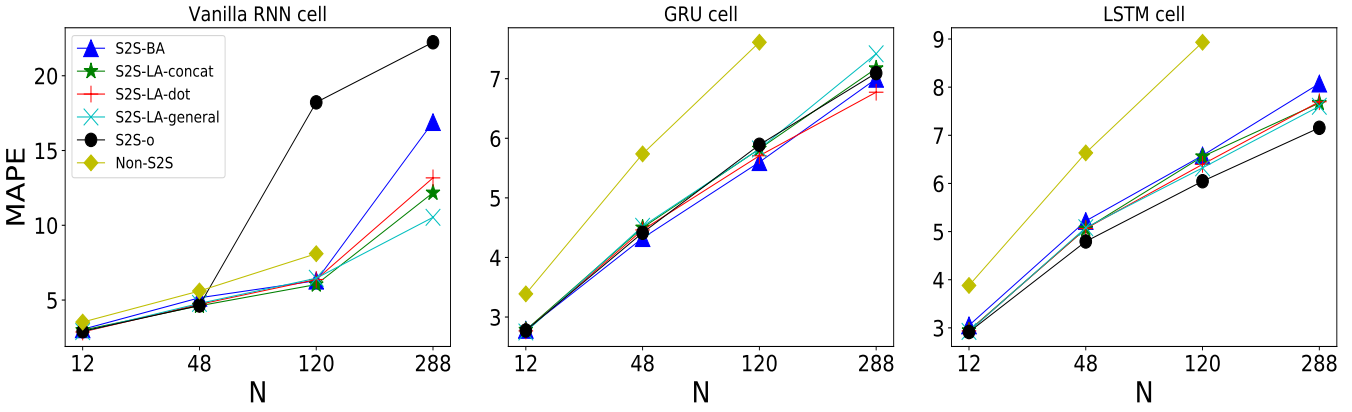

FIGURE 12. Comparing cell performance of each non-DNN model: input $T = 120$, all prediction lengths N.

288 encoder outputs contribute to each decoder output, which may be excessive and causes a loss in accuracy rather than improvement.

C. DISCUSSION

This subsection analyzes model results for varying input length T, hence across input time as the prediction length increases to determine how each model performs with different input lengths.

From Fig. 15, it can be seen that, for the longest prediction length $N = 288$, the best model versions were obtained for input lengths either $T = 48$ or $T = 120$. Looking at the S2S-BA model in Fig. 15, the model with input length $T = 288$ achieved better accuracy than the same model with other input lengths for prediction lengths $N = 12$ and $N = 48$; a result that is shared with all other model types. However, the S2S-BA model with input length $T = 288$ 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 $N = 288$. 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 $N = 120$ and $N = 288$; 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 $N \leq 48$, while versions for $T < 288$ achieve the best results for $N \geq 120$. With $T = 288$ and $N = 288$, attention models are trying to "pay attention" to an entire previous day in the latter end of the decoder part. 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, to predict the last output value at $i = 288$, attention calculation incorporates the very first encoder output at $j = 1$, which is 576 (288×2) time steps in the past, nearly 48 hours previous.

As expected, 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$_{T=12}$. The S2S-o models do not use an attention mechanism; the predicted output is computed using the previously predicted output and previous decoder hidden state, therefore there is no loss of accuracy with long input and output sequences.

Lastly, the DNN model in Fig. 15 shows the clearest signs of improvement over input length versions T; the DNN$_{T=288}$ model achieved the best results across all prediction lengths.

Overall, it can be concluded that as the prediction length N

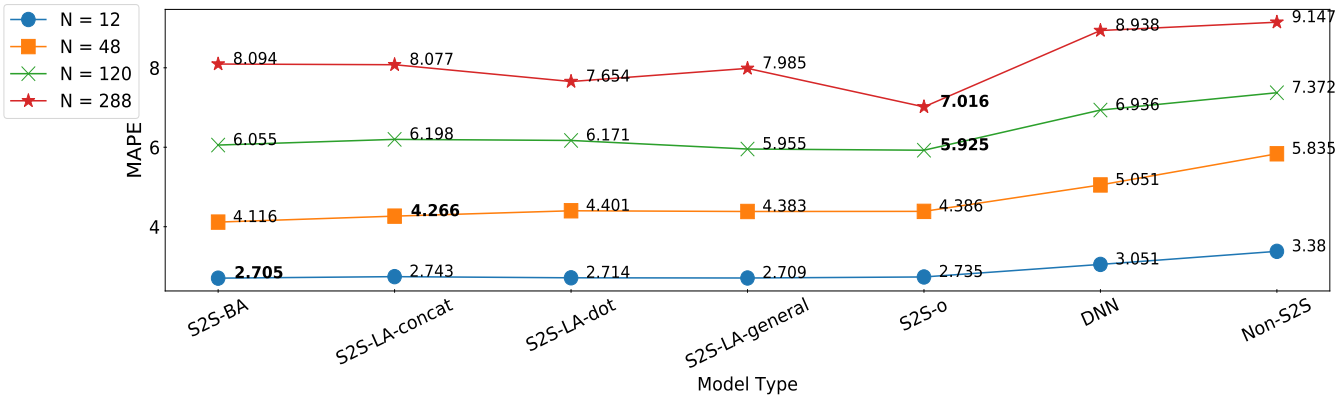

FIGURE 13. Best achieved results by each model type: input $T = 288$, all prediction lengths N.

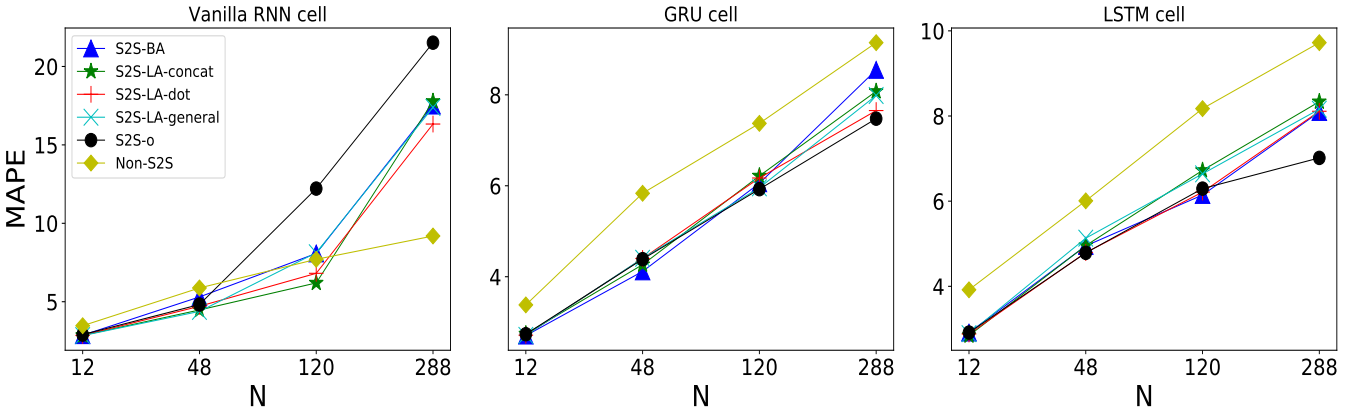

FIGURE 14. Comparing cell performance of each non-DNN model: input $T = 288$, all prediction lengths N.

increases, the accuracy decreases. This can also be observed from Table 1 which summarizes all results and identifies the best performing model. As seen from the table, the S2S-BA model was the dominant model; 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, the preferred model may not necessarily be the S2S-BA model or any attention model for that matter. The attention models contain more parameters than 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 being faster to train because of having fewer parameters. Hence, if the interested party main objective is high accuracy irrelevant of the training speed, the S2S-BA model is preferred. If a slight decrease in accuracy is acceptable, a reduction in training speed can be achieved by using the S2S-o model.

VI. CONCLUSION

Continuously increasing electricity consumption and its impact on global warming are escalating importance of energy efficiency and conservation. Load forecasting is contributing to energy management efforts through improved production planning and scheduling, budget planning, and by identifying 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 paper proposes Sequence to Sequence Recurrent Neural Network (S2S RNN) with Attention for electrical load forecasting. RNN provides ability to model time dependencies and the S2S approach strengthens this ability by using two RNNs: encoder and decoder. Moreover, an attention mechanism was added to ease the connection between encoder and decoder and further improve load forecasting. The proposed solution was evaluated with four attention mechanisms, three RNN cells (vanilla, LSTM, and GRU), with different forecasting horizons, and different input lengths. As expected, the forecasting accuracy decreases as prediction length increases; however, the decline is much steeper with DNN models then with S2S-o or S2S models with attention.

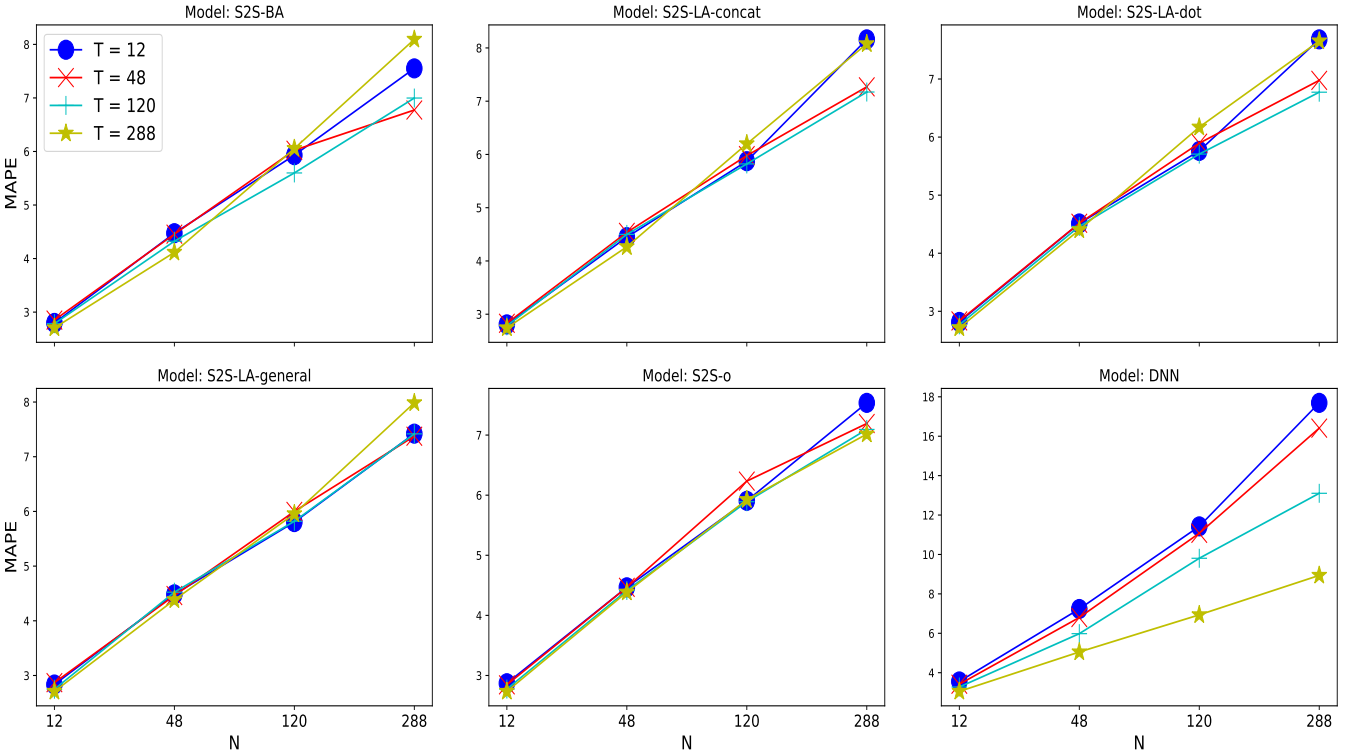

FIGURE 15. Analyzing the best MAPE achieved over varied input lengths T and prediction lengths N.

TABLE 1. Best achieved MAPE and MAE for each model and prediction length. Input length, cell used, and hidden state size not taken into consideration.

Model	MAPE (%)				MAE			
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
S2S-LA-dot	2.714	4.401	5.707	**6.773**	12.175	19.518	25.356	30.316
S2S-LA-general	2.709	4.383	5.805	7.375	**12.137**	19.421	25.836	33.316
S2S-LA-concat	2.743	4.266	5.822	7.172	12.287	19.189	25.747	32.664
S2S-BA	**2.705**	**4.116**	**5.599**	6.774	12.155	**18.383**	**24.948**	**30.069**
Non-S2S	3.380	5.606	7.372	9.147	15.146	24.668	33.249	40.559
DNN	3.051	5.051	6.936	8.938	13.424	21.436	28.966	37.389

Overall, S2S with Bahdanau attention outperformed all other models. It is important to note that with S2S attention models, accuracy does not continuously increase with the increase of input sequence length. For longer prediction length $N = 288$, attention models with input sequence $T = 48$ and $T = 120$ achieve better accuracy than the same models with $T = 288$. This demonstrated that long input sequences are not needed with S2S attention models.

Future work will evaluate the proposed approach on different datasets, investigate improving longer-term predictions, and examine local attention models.

REFERENCES

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

[2] I. E. Agency, "Energy." https://www.iea.org/weo2018/, 2018. Accessed on: July 2019.

[3] U. E. I. Administration, "Use of energy in the united states explained." https://www.eia.gov/energyexplained, 2017. Accessed on: July 2019.

[4] H. C. Akdag and T. Beldek, "Waste management in green building operations using gscm," International Journal of Supply Chain Management, vol. 6, no. 3, pp. 174–180, 2017.

[5] I. E. S. Operator, "Global adjustment and peak demand factor." http://www.ieso.ca/Sector-Participants/Settlements/Global-Adjustment-and-Peak-Demand-Factor, 2019. Accessed on: July 2019.

[6] R. Jiao, T. Zhang, Y. Jiang, and H. He, "Short-term non-residential load forecasting based on multiple sequences lstm recurrent neural network," IEEE Access, vol. 6, pp. 59438–59448, 2018.

[7] H. Jiang, Y. Zhang, E. Muljadi, J. J. Zhang, and D. W. Gao, "A short-term and high-resolution distribution system load forecasting approach using support vector regression with hybrid parameters optimization," IEEE Transactions on Smart Grid, vol. 9, no. 4, pp. 3341–3350, 2018.

[8] K. Chen, K. Chen, Q. Wang, Z. He, J. Hu, and J. He, "Short-term load forecasting with deep residual networks," IEEE Transactions on Smart Grid, vol. 10, no. 4, pp. 3943–3952, 2019.

[9] K. Grolinger, A. L'Heureux, M. A. Capretz, and L. Seewald, "Energy forecasting for event venues: Big data and prediction accuracy," Energy and Buildings, vol. 112, pp. 222–233, 2016.

[10] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," Nature, vol. 521, pp. 436–444, 2015.

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

[12] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in Advances in Neural Information Processing Systems (NIPS), pp. 3104–3112, 2014.
[13] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," in International Conference on Learning Representations (ICLR), 2015.
[14] M. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," 2015. arXiv preprint arXiv:1508.04025.
[15] P. J. Werbos, "Backpropagation through time: what it does and how to do it," Proceedings of the IEEE, vol. 78, no. 10, pp. 1097–1105, 1990.
[16] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," in International Conference on Machine Learning (ICML), pp. 1310–1318, 2012.
[17] S. Hochreiter and J. Schmidhuber, "Long short-term memory," Neural Computation, vol. 9, no. 8, pp. 1735–1780, 1997.
[18] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," arXiv preprint arXiv:1406.1078, 2014.
[19] D. L. Marino, K. Amarasinghe, and M. Manic, "Building energy load forecasting using deep neural networks," in IEEE Industrial Electronics Society (IECON), pp. 7046–7051, 2016.
[20] 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, no. 1, pp. 91–99, 2016.
[21] 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, no. 1, pp. 5–25, 2018.
[22] J. G. Jetcheva, M. Majidpour, and W. Chen, "Neural network model ensembles for building-level electricity load forecasts," Energy and Buildings, vol. 84, no. 1, pp. 214–223, 2014.
[23] 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, no. 1, pp. 184–194, 2016.
[24] D. B. Araya, K. Grolinger, H. ElYamany, M. Capretz, and G. Bitsuamlak, "An ensemble learning framework for anomaly detection in building energy consumption," Energy and Buildings, vol. 144, no. 1, pp. 191–206, 2017.
[25] N. L. Tasfi, W. A. Higashino, K. Grolinger, and M. A. Capretz, "Deep neural networks with confidence sampling for electrical anomaly detection," in IEEE International Conference on Smart Data, pp. 1038–1045, 2017.
[26] K. Amarasinghe, D. L. Marino, and M. Manic, "Deep neural networks for energy load forecasting," in IEEE International Symposium on Industrial Electronics (ISIE), pp. 1483–1488, 2017.
[27] 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, 2019.
[28] H. Shi, M. Xu, and R. Li, "Deep learning for household load forecasting - a novel pooling deep rnn," IEEE Transactions on Smart Grid, vol. 9, no. 5, pp. 5271–5280, 2018.
[29] 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, no. 7, pp. 1636–1656, 2018.
[30] 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.
[31] L. Han, Y. Peng, Y. Li, B. Yong, Q. Zhou, and L. Shu, "Enhanced deep networks for short-term and medium-term load forecasting," IEEE Access, vol. 7, pp. 4045–4055, 2018.
[32] 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, no. 8, p. 1168, 2017.
[33] A. Rahman, V. Srikumar, and A. D. Smith, "Predicting electricity consumption for commercial and residential buildings using deep recurrent neural networks," Applied energy, vol. Vol. 212, pp. 372–385, 2018.
[34] L. Sehovac, C. Nesen, and K. Grolinger, "Forecasting building energy consumption with deep learning: A sequence to sequence approach," in International Congress on Internet of Things (ICIOT), 2019.
[35] S. Venugopalan, M. Rohrbach, J. Donahue, R. Mooney, T. Darrell, and K. Saenko, "Sequence to sequence-video to text," in IEEE international conference on computer vision, pp. 4534–4542, 2015.
[36] K. Kawano, S. Koide, and C. Imamura, "Seq2seq fingerprint with byte-pair encoding for predicting changes in protein stability upon single point mutation," IEEE/ACM Transactions on Computational Biology and Bioinformatics, 2019.
[37] A. Paszke, S. Gross, S. Chintala, and G. Chanan, "Pytorch: from research to production." https://pytorch.org, 2017. Accessed on: July 2019.