Journal of Database Management
Volume 34 • Issue 1

Adaptive Modularized Recurrent Neural Networks for Electric Load Forecasting

Fangwan Huang, Fuzhou University, China
Shijie Zhuang, Fuzhou University, China
Zhiyong Yu, Fuzhou University, China
Yuzhong Chen, Fuzhou University, China
Kun Guo, Fuzhou University, China*

ABSTRACT

In order to provide more efficient and reliable power services than the traditional grid, it is necessary for the smart grid to accurately predict the electric load. Recently, recurrent neural networks (RNNs) have attracted increasing attention in this task because it can discover the temporal correlation between current load data and those long-ago through the self-connection of the hidden layer. Unfortunately, the traditional RNN is prone to the vanishing or exploding gradient problem with the increase of memory depth, which leads to the degradation of predictive accuracy. Many RNN architectures address this problem at the expense of complex internal structures and increased network parameters. Motivated by this, this article proposes two adaptive modularized RNNs to tackle the challenge, which can not only solve the gradient problem effectively with a simple architecture, but also achieve better performance with fewer parameters than other popular RNNs.

KEYWORDS

Cumulative Priority, Gate Mechanism, General framework, Long-term Dependencies, Module Update, Multi-timescale Connections, Skip Length, Weight Pruning

INTRODUCTION

Electric load forecasting plays a vital role in ensuring the security, stability, and efficiency of the smart grid. From the view of electricity generation, it helps to make a reasonable plan to provide a sufficient power supply and avoid the waste of resources caused by excessive production (Patel et al., 2019). From the view of the electricity market, it helps to set a time-of-use price to encourage off-peak power consumption (Zhao et al., 2021). Recently, the continuous emergence of various high-precision data acquisition equipment (such as smart meters) in the smart grid has provided strong support for electric load forecasting (Fekri et al., 2021).

DOI: 10.4018/JDM.323436 *Corresponding Author

As load data are usually recorded sequentially at a certain time interval, electric load forecasting can be regarded as the time series prediction in the field of data mining (Yu et al., 2020). Therefore, Recurrent Neural Network (RNN) that can well capture the time correlation of the sequence has been considered as a good choice for this task recently (Bianchi et al., 2017). As the simplest architecture, a Vanilla RNN is generally composed of three parts: the input layer, the hidden layer, and the output layer. Instead of the traditional Multi-Layer Perceptron containing only input and output connections, the Vanilla RNN introduces the recurrent connection from the previous to the current moment in the hidden layer (Elman, 1990). This means that the current input x_t and the hidden state of the previous moment h_{t-1} together affect the hidden state h_t. Because of the self-connection of the hidden layer, RNN designed for sequence modeling can be regarded as a deep network when it is unrolled along the time axis. Such a deep network structure can easily lead to the vanishing or exploding gradient problem for the long sequence in the process of parameter training using Back Propagation Through Time (BPTT) (Fernando et al., 2018). To address this challenge, different architectures were proposed to improving the trainability of RNN, but at the cost of significant computation overhead, such as Long Short-Term Memory (LSTM) (Hochreiter et al., 1997) and its variants (Yu et al., 2019). They pose a challenge in the training of many more parameters with the introduction of gates. Recently, Clock-Work RNN (CW-RNN) was proposed to utilize another simple but effective architecture to alleviate the gradient problem (Koutnik et al., 2014). It first divides the hidden layer into several modules with different updated frequencies, and then makes the slow-updating modules have recurrent connections with longer time delays. This allows data dependencies to be passed in fewer time steps to avoid excessive multiplications of the gradient. Besides, CW-RNN uses the predefined rule instead of training to determine module updates, greatly reducing the number of network parameters. But this inevitably weakens the generalization ability of the network. The motivation of this article is to refine and extend this architecture based on multi-timescale connections, aiming to resolve the contradiction between the performance and the number of parameters. The research content mainly includes the adaptive updating strategy of modules in the hidden layer and the pruning problem of recurrent connections caused by this strategy. The contributions of this article are:

- A new modularized RNN (M-RNN) is proposed to generalize the existing CW-RNN. M-RNN is a framework with the skip length of the module as the key component, which can realize the update of the hidden state by taking the module as the minimum unit.
- Two adaptive strategies for updating the hidden modules are proposed by designing two new activation functions to calculate the priority of each module. On this basis, the unordered and ordered adaptive M-RNNs (AM-RNNs) are defined respectively to achieve dynamic multi-timescale connections.
- Since the existing pruning strategy is only applicable to ordered AM-RNN, a two-way pruning strategy is designed for the unordered AM-RNN to realize the sparsification of recurrent connections.
- Both versions of AM-RNN are compared with other popular RNNs for electric load forecasting. The experimental results show that AM-RNNs can achieve better predictive accuracy with fewer network parameters than the current RNNs widely used in this field.

RELATED WORK

Considering the economic and environmental implications of even a slight improvement in accuracy, there is still a lot of research on electric load forecasting recently (Bendaoud et al., 2020). Methods to accomplish this task can be generally divided into two categories: statistics-based methods and data-driven methods (Hong et al., 2016). Although the statistics-based methods have been developed very maturely, their abilities are limited due to the nonlinearity of sequences, the randomness of user behaviors, the diversity of external factors, etc. (Hafeez et al., 2020). This makes data-driven methods

based on artificial intelligence gradually become a research hotspot, including dynamic neural network (Mordjaoui et al., 2017), extreme learning machine (Chen et al., 2018), deep belief network (Ouyang et al., 2019), convolutional neural network (Huang et al., 2020), etc. However, the above models do not take the temporal relationship of data as a definite feature of the load, which may lead to performance degradation when complex dependencies between the current load and those long-ago need to be considered. From this point, RNNs that can continuously transmit early input information through the hidden state have been widely applied to electric load forecasting, especially in the absence of feature engineering (Shi et al., 2018). Because the gradient problem mentioned above makes the Vanilla RNN perform poorly in long sequence learning, some measures including algorithm substitution, weight constraint, gate mechanism, and multi-timescale connections have been proposed for its improvement.

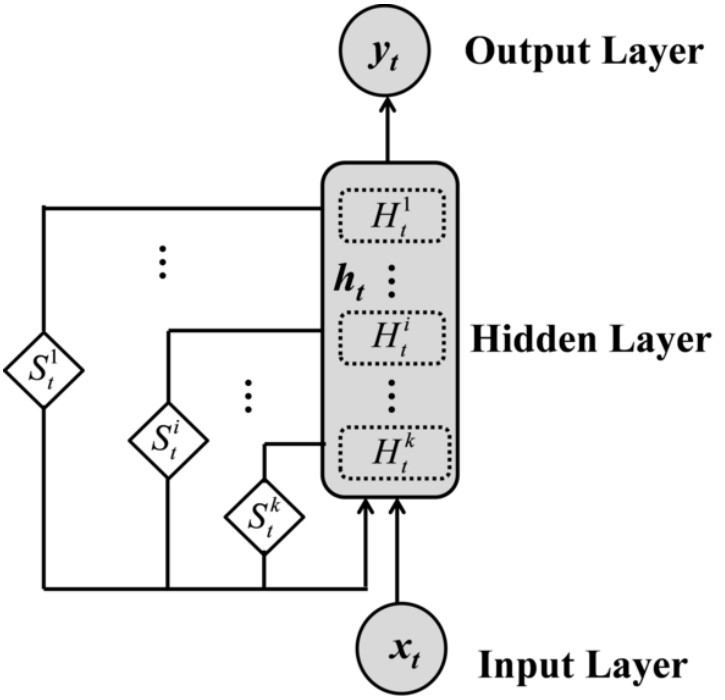
The first measure is to replace BPTT with other optimization algorithms, such as Hessian-Free (HF) method (Martens et al., 2012), Reservoir Computing (RC) (Gallicchio et al., 2017), etc. However, they are also criticized for problems like difficulty in implementation or limited learning capacity. The second measure is to constrain the recurrent weight to guarantee that the continuous multiplication of multiple matrices does not cause the gradient to approach zero or very large (Arjovsky et al., 2016). Note that the strict implementation of weight constraints may hinder training speed and generalization ability (Vorontsov et al., 2017). Gate mechanism represented by LSTM and its variants is the third measure and has been applied in numerous tasks of time series, including short-term or midterm electric load forecasting (Li et al., 2021) (Dudek et al., 2021). The hidden state of LSTM needs to be obtained through a cell state and three gate vectors, all of which require additional connection weights with the input and the previous hidden state. So a typical LSTM requires about four times as many training parameters as Vanilla RNN (Greff et al., 2016). Reducing the number of the gate is a common method that leads to faster convergence and improved generalization. As a simplified variant of LSTM, Gated Recurrent Unit (GRU) without memory cell performed better on some tasks than LSTM, even though it has only the update and reset gates (Cho et al., 2014). Furthermore, Minimal Gated Unit (MGU) which contains only one gate can be regarded as the simplest design among all gated architecture by sharing the gate vector (Zhou et al., 2016). To handle the problem of gate undertraining, a new gate mechanism was designed to perform the element-wise refining operation on the input and the output of each gate (Cheng et al., 2020). While it can be equipped with any kind of gated RNNs to improve performance, it does not reduce the number of parameters. The final common measure to improve the performance of RNN on long-term dependencies is to introduce multi-timescale connections with increasing recurrent skip lengths. Similar to the skip-connection of residual neural network (He et al., 2016), the main idea of multi-timescale connections is that the recurrent connections should exist not only in adjacent time steps but also in larger time steps. For example, Skip RNN allows the network to determine whether to fully replicate the previous state or to update it at the current moment, based on the calculated updated likelihood (Campos et al., 2018). In addition to neurons, multi-timescale connections can also be formed in modules, such as the CW-RNN described earlier (Koutnik et al., 2014). Dilated RNN stacks multiple hidden layers that work on different skip lengths to focus on different temporal dependencies (Chang et al., 2017). More recently, the latest advances have focused on integrating some of the aforementioned measures (Jing et al., 2019) (Moirangthem et al., 2021).

In summary, from the perspective of trade-off optimization between accuracy and efficiency, the authors consider the modularized RNN (M-RNN) modified on the Vanilla RNN to be a very competitive multi-timescale architecture because it does not require additional hidden layers or connection weights. However, little work has been done on this architecture, prompting them to study it in this article.

THE NEW FRAMEWORK M-RNN

The core idea of M-RNN is to obtain the short-term and long-term dependencies of the data by introducing different skip lengths of the hidden modules at different moments. According to the

Figure 1. Illustration of M-RNN where . S_t^i represents the skip length of the module H_t^i at time t



framework shown in Figure 1, the hidden layer is first divided into k modules. It is assumed that each module has m neurons, which means the number of the hidden neurons $n_h=k*m$. Secondly, skip length $\boldsymbol{S}_t^i$ (i=1, 2, …, k) is introduced to determine the time delay of the module $\boldsymbol{H}_t^i$, to clearly distinguish its responsibility for short-term or long-term dependency at time t. The module with a smaller skip length is more conducive to short-term dependency. Conversely, the longer the skip length, the fewer steps needed to transform the information, and the more favorable the long-term dependency. The equations of M-RNN are defined as (1)-(3).

$$\text{M-RNN: } h_t' = f_h(W_{ih}x_t + (W_{hh} * M)h_{t-1} + b_h), \tag{1}$$

$$h_t = (1-u_t) \odot h_{t-1} + u_t \odot h_t', \tag{2}$$

$$y_t = f_o(W_{ho}h_t + b_o). \tag{3}$$

Here, x_t, h_t, h_t', and y_t are the input, the hidden state, the candidate state, and the output at time step t; $\odot$ and $*$ are denoted as the element-wise product between two vectors and two matrices, respectively; $f_h(.)$ and $f_o(.)$ are the activation functions of the hidden and the output layer, such as tanh or ReLu. W_{ih}, W_{hh}, and W_{ho} represent the input, the recurrent, and the output weight matrices. They and the bias vectors b_h, b_o are parameters to be learned. W is used to represent all the above parameters, which are required to be the same at each time step. Note that the equations for M-RNN are very similar to those for GRU. There are two main differences between them. One is that the updated vector u_t of GRU is learned by the update gate, while the updated vector of M-RNN is easier to obtain by the updated strategy described later. The other is that GRU adopts the reset vector r_t learned by

the reset gate to obtain part of h_{t-1} for the calculation of h_t', as shown in (4). While M-RNN achieves a similar effect by a mask matrix M obtained by the pruning strategy of recurrent connections.

$$\text{GRU: } h_t' = f_h(W_{ih}x_t + W_{hh}(r_t \odot h_{t-1}) + b_h). \quad (4)$$

Besides, the elements of u_t and r_t for GRU range from 0 to 1 due to activation functions (usually sigmoid), while the elements of u_t and M for M-RNN are either 1 or 0. In other words, M-RNN can be thought of as containing a binary gate that controls the flow of information. Furthermore, M-RNN requires the same values of updated elements for all neurons in the same module to ensure that they are updated or retained at the same time. Finally, as a general framework, note that the skip length $\boldsymbol{S_t^i}$ can be either a constant or a variable, depending on the updated strategy of the module described below.

THE UPDATED STRATEGY OF HIDDEN MODULES

Different from RNN with gate mechanism, the update of M-RNN is not based on the neuron, but on the module, which aims to greatly reduce the network parameters and training time. As the simplest model of M-RNN, the computational efficiency of CW-RNN is not only much higher than gated RNNs but even higher than Vanilla RNN, because not all modules are updated at every time, but must be determined by the updated vector. Therefore, designing an appropriate strategy to obtain the updated vector is crucial to the performance of M-RNN. In this article, based on the fixed strategy adopted by CW-RNN, several other effective updating strategies are discussed to meet the challenge of finding suitable time scales.

THE EXISTING STRATEGIES AND THEIR SHORTCOMINGS

The Fixed Strategy

As a typical representative, CW-RNN utilizes a fixed strategy to determine whether each module is updated at time step t through a predefined rule. More specifically, after assigning a period T_i to each module, CW-RNN stipulates that the state of all neurons in the module can be updated only if the current time t is divisible by the period of the module; otherwise, the previous state should always be maintained. According to the above rule, the updated vector of CW-RNN in any time step can be obtained without training. If the updated vector u_t is divided into k sub-vectors according to module size, then the element values of each sub-vector are either all zeros or all ones. One means that the corresponding neuron is involved in the update, while zero means that it retains the value of the previous time. Figure 2 shows the locations of the activated modules about CW-RNN with 4 modules at different time steps, where the period $T_i=2^{i-1}$ ($1 \le \boldsymbol{i} \le \boldsymbol{k}$).

As can be seen from Figure 2, this CW-RNN can provide a set of recurrent connections with skip lengths of 1, 2, 4, and 8. Modules with longer skip lengths update slower than other modules.

Figure 2. Illustration of the locations of the activated modules about CW-RNN with 4 modules, where gray indicates that this module participates in the update at the current moment

	t	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
T_1=1	module1	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■
T_2=2	module2		■		■		■		■		■		■		■		■
T_3=4	module3				■				■				■				■
T_4=8	module4								■								■

By taking advantage of these slow-updating modules, information that takes a lot of time steps to transmit has other paths with fewer time steps. This effectively alleviates the gradient problem caused by too long time steps. The benefit of the fixed strategy is that its simplicity reduces the number of network parameters and saves a lot of training time. However, it is undeniable that the generalization ability of fixed strategies is unsatisfactory.

The Random Strategy

Inspired by Zoneout which regularizes LSTM by randomly preserving hidden states (Krueger et al., 2017), the authors designed an updated strategy for the modules based on random probability in their early work (Zhuang et al., 2020). Different from traditional Zoneout based on neurons, this strategy randomly assigns some modules to participate in the update at each time step. Firstly, the module H_t^i is randomly assigned an updated probability P_t^i at each time step. Secondly, given an updated threshold ε ($0<\varepsilon<1$), the elements in the updated vector corresponding to the neurons in H_t^i are set to 1 only if $P_t^i > \varepsilon$, otherwise are set to 0. In this way, the updated vector for any time step can be obtained. Figure 3 shows an example of Zoneout M-RNN (ZM-RNN) based on the random strategy.

It can be concluded from Figure 3 that the skip length of each module is no longer a constant but a variable, which means that each module is sometimes updated faster and sometimes slower. This means a strict division between faster and slower modules can be avoided. Each module can accommodate both short-term and long-term dependencies. Although the random strategy is simple to implement, it inevitably introduces an additional hyper-parameter, namely the updated threshold, which can be adjusted by the validation set. Besides, the performance fluctuation caused by random probability is also one of the problems to be considered.

THE PROPOSED STRATEGIES

The Adaptive Strategy Without Order

Compared with the existing strategies, the adaptive strategy can be regarded as a more active and effective strategy. The key is to determine which modules are worth updating and which ones are worth retaining at each time step. In this article, a new concept of "module priority" is proposed to measure the importance of each module participating in the update. Firstly, the element-wise vector based on (5) is implemented for the candidate state h_t' to obtain the priority of the hidden neurons. Secondly, as shown in (6), the priority of each neuron in the same module is accumulated to obtain the priority of each module.

$$e_t = soft\max(h_t') . \tag{5}$$

$$a_t^i = \sum_{j=(i-1)m+1}^{im} e_t^j \quad i = 1, 2, \cdots, k . \tag{6}$$

Figure 3. Illustration of the locations of the activated modules about ZM-RNN with 4 modules, where gray indicates that this module participates in the update at the current moment

t	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
module1																
module2																
module3																
module4																

Here, k represents the number of modules in the hidden layer; m represents the number of neurons in each module; e_t^j refers to the jth element of the vector e_t; and a_t^i indicates the priority of module H_t^i at time t. Finally, according to the priority of each module, the updated sub-vector of each module $u_t^i \in R^m$ can be easily obtained by (7).

$$u_t^i = \begin{cases} (1,1,\cdots,1)^T & if\ a_t^i > \dfrac{1}{k+1}, \\ (0,0,\cdots,0)^T & else. \end{cases} \tag{7}$$

The updated threshold is set to $1/(k+1)$ to ensure that all modules have the opportunity to participate in the update at some particular moment. Finally, a fully updated vector u_t can be obtained by splicing all the updated sub-vectors up and down. For the convenience of expression, the adaptive M-RNN based on this strategy is called AM-RNN-I. Figure 4 shows several examples of AM-RNN-I with 4 modules obtaining the updated vectors based on (5)-(7). The skip length obtained by AM-RNN-I is similar to that obtained by ZM-RNN, which is also variable and does not strictly distinguish long/short dependencies. Therefore, Figure 3 can also be used to illustrate the locations of the activated modules about AM-RNN-I. The difference between them is that ZM-RNN is based on random probability to determine whether a module is updated, while AM-RNN-I is based on the module priority. By comparing Figure 2 and Figure 3, it can be observed that ZM-RNN and AM-RNN-I do not have an obvious hierarchy like CW-RNN because the update of the module is independent of its index.

The Adaptive Strategy With Order

The latest research has an explicit preference for the hierarchical hidden layer (Shen et al., 2019) (Schoene et al., 2021). This leads the authors to further consider how to improve the above adaptive strategy to ensure that the updated frequency of modules can be decreased in order. In other words, when the module H_t^i is updated, all modules with smaller indexes (H_t^j, $1 \le j < i$) are updated at the same time. To achieve this goal, the learning process of the updated vector needs to perform the following two steps after gaining the priority of each module:

Step 1: Calculate the cumulative priority of each module c_t^i in descending order of the index, as shown in (8). Obviously, $c_t^i < c_t^{i-1}$ and $c_t^1 = 1$ because the softmax function in (5) guarantees that the sum of all neurons' priority is 1.

$$c_t^i = \sum_{j=i}^{k} a_t^j \quad i = k, k-1, \cdots, 1. \tag{8}$$

Figure 4. Several examples of AM-RNN-I with 4 modules obtaining the updated vectors, where the updated threshold is set to 1/5

	a_t^i	u_t^i	a_t^i	u_t^i	a_t^i	u_t^i	a_t^i	u_t^i
Module1	0.5	$(1,1,\ldots,1)^T$	0.2	$(0,0,\ldots,0)^T$	0.3	$(1,1,\ldots,1)^T$	0.25	$(1,1,\ldots,1)^T$
Module2	0.2	$(0,0,\ldots,0)^T$	0.3	$(1,1,\ldots,1)^T$	0.3	$(1,1,\ldots,1)^T$	0.25	$(1,1,\ldots,1)^T$
Module3	0.1	$(0,0,\ldots,0)^T$	0.2	$(0,0,\ldots,0)^T$	0.1	$(0,0,\ldots,0)^T$	0.25	$(1,1,\ldots,1)^T$
Module4	0.2	$(0,0,\ldots,0)^T$	0.3	$(1,1,\ldots,1)^T$	0.3	$(1,1,\ldots,1)^T$	0.25	$(1,1,\ldots,1)^T$

Step 2: Set an updated threshold ε ($0<\varepsilon<1$) to determine the number of modules to be updated per time step. The updated vector can be obtained by (9).

$$u_t = (\underbrace{1,\cdots,1}_{m\times i},\underbrace{0,\cdots,0}_{m\times(k-i)})^T \quad if\ c_t^i \le \varepsilon < c_t^{i-1}\ (2 \le i \le k)\,. \tag{9}$$

More specifically, the updated vector is split into two segments: the 1-segment and the 0-segment. The length of the segment is variable, depending on the number and size of the updated modules. The AM-RNN based on the above-updated rule is called AM-RNN-II. Figure 5 shows several examples of AM-RNN-II with 4 modules obtaining the updated vectors where the updated threshold ε=0.5.

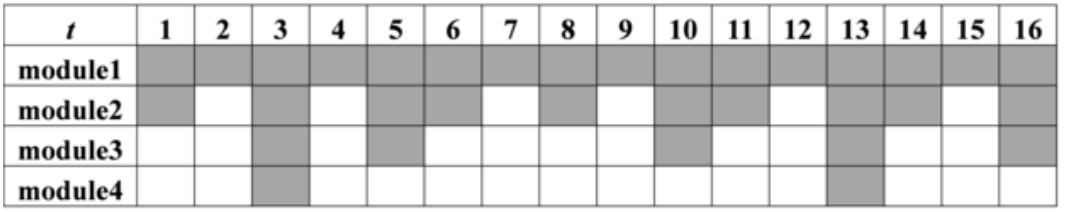
It can be concluded from Figure 5 that module H_t^1 is updated at each time step, which allows it to be treated as a Vanilla RNN. The updated frequency of modules keeps decreasing, making the module with a larger index more conducive to long-term dependencies. Figure 6 shows more details about the location of the activated modules about an AM-RNN-II with 4 modules. For example, the updated vectors at time t=2, 6, 10, 13 can be considered similar to the four examples in Figure 5, respectively. Similar to CW-RNN, AM-RNN-II has strict differences in updated frequency between modules. However, except that the skip length of module H_t^1 is a constant equal to 1, those of all other modules are a variable learned in a data-driven way.

In summary, M-RNN can contain multiple models based on the above-updated strategies, as shown in Figure 7. The characteristic of CW-RNN based on the fixed strategy is that the skip length of each module is a preset constant. So the Vanilla RNN can be regarded as a special case of CW-RNN retaining all recurrent connections when the hidden layer is a module with skip length 1. Different from CW-RNN, the module of ZM-RNN based on the random strategy is sometimes updated fast and sometimes updated slowly. The probability change of skip lengths can achieve dynamic updating of modules. Finally, AM-RNN-I and AM-RNN-II based on the adaptive strategy can be regarded as the improvement of ZM-RNN and CW-RNN respectively. Their common feature is that the skip lengths can be adjusted according to the input information, rather than determined in advance or randomly.

Figure 5. Several examples of AM-RNN-II with 4 modules obtaining the updated vectors, where the updated threshold is set to 0.5

	a_t^i	c_t^i	u_t^i	a_t^i	c_t^i	u_t^i	a_t^i	c_t^i	u_t^i	a_t^i	c_t^i	u_t^i
Module1	0.6	1	$(1,1,\ldots,1)^T$	0.4	1	$(1,1,\ldots,1)^T$	0.2	1	$(1,1,\ldots,1)^T$	0.1	1	$(1,1,\ldots,1)^T$
Module2	0.1	0.4	$(0,0,\ldots,0)^T$	0.3	0.6	$(1,1,\ldots,1)^T$	0.2	0.8	$(1,1,\ldots,1)^T$	0.1	0.9	$(1,1,\ldots,1)^T$
Module3	0.1	0.3	$(0,0,\ldots,0)^T$	0.1	0.3	$(0,0,\ldots,0)^T$	0.3	0.6	$(1,1,\ldots,1)^T$	0.1	0.8	$(1,1,\ldots,1)^T$
Module4	0.2	0.2	$(0,0,\ldots,0)^T$	0.2	0.2	$(0,0,\ldots,0)^T$	0.3	0.3	$(0,0,\ldots,0)^T$	0.7	0.7	$(1,1,\ldots,1)^T$

Figure 6. Illustration of the locations of the activated modules about AM-RNN-II with 4 modules, where gray indicates that this module participates in the update at the current moment



THE PRUNING STRATEGY OF RECURRENT CONNECTIONS

In addition to the updated strategy of hidden modules, another key issue for M-RNN is the pruning strategy of recurrent connections. According to different pruning strategies, different mask matrices M in (1) can be obtained to determine which information of h_{t-1} is used to calculate the candidate state h_t'. It should be pointed out that the purpose of M-RNN adopting a pruning strategy instead of the reset gate in GRU is to further reduce network parameters and avoid over-fitting.

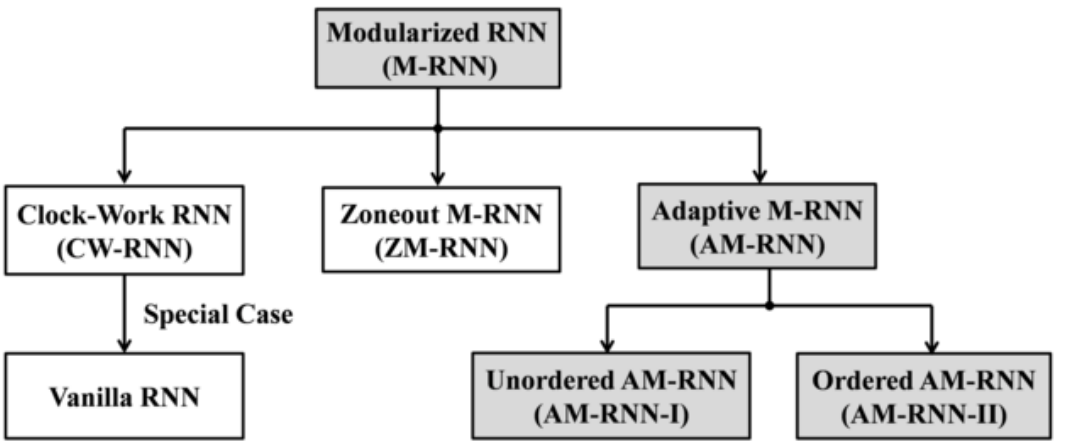
THE EXISTING STRATEGY FOR ONE-WAY PRUNING AND ITS LIMITATIONS

CW-RNN emphasizes that a given hidden neuron at time t-1 is only allowed to establish connections with those hidden neurons running at the same or faster-updated frequency at time t. Since a larger module index means a slower-updated frequency, the neurons of the module H_{t-1}^{j} can be fully connected to the neurons of the module H_t^i only if $j \geq i$. Figure 8(a) shows an illustration of this strategy. According to this slow-to-fast connection, if the mask matrix M are partitioned into $k \times k$ blocks as shown in (10), then $M_{ij} \in R^{m \times m}$ ($i \leq j$) is a sub-matrix with all elements of value 1, otherwise, it must be a sub-matrix with all elements values of 0.

$$M = \begin{bmatrix} M_{11} & \cdots & M_{1k} \\ \vdots & \ddots & \vdots \\ M_{k1} & \cdots & M_{kk} \end{bmatrix} s.t. \quad M_{ij} = \begin{bmatrix} 1 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \cdots & 1 \end{bmatrix} or \begin{bmatrix} 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \end{bmatrix}. \tag{10}$$

Since M is a block-upper triangular matrix, the weights below the block-diagonal of the Hadamard product W_{hh}*M in (1) are all 0, to realize the pruning of the recurrent connections. This means that only $m^2 \times k \times (k+1)/2$ non-zero parameters in W_{hh} need to be trained, which is much smaller than $(m \times k)^2$ parameters of Vanilla RNN. It can be concluded from (1) that this strategy makes the slow-updating modules retain less information about h_{t-1} and rely more on the input x_t when calculating the candidate state h_t'. However, this strategy needs to determine which module updates faster or slower to

Figure 7. Illustration of multiple models belonging to M-RNN, where the main work of this paper is highlighted with grey shadow



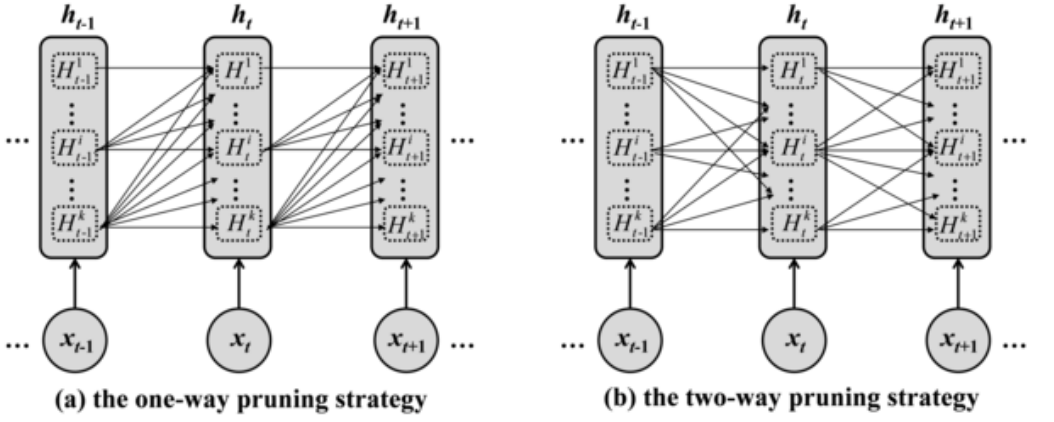
achieve one-way pruning of the recurrent connections. Therefore, it is only suitable for ordered M-RNNs (CW-RNN or AM-RNN-II). The unordered M-RNNs (ZM-RNN or AM-RNN-I) cannot meet the above requirement due to the variable skip length of each module at different moments. A two-way strategy is designed to solve the problem that the updating speed of modules cannot be strictly distinguished.

THE PROPOSED STRATEGY FOR TWO-WAY PRUNING

Previous work by the authors has shown that CW-RNN can also perform well when there is not only the slow-to-fast connection but also the fast-to-slow connection (Huang et al., 2019). The fast-to-slow connection means a module at time t-1 can be connected to a module with the slower-updated frequency at time t. On this basis, a two-way strategy is designed for the unordered M-RNNs, whose principle is to decide which sub-matrices in M have all 1 or all 0 elements according to the pruning threshold p. This strategy is similar to the sparse connections of the reservoir in ESN (Hu et al., 2020), except that it is to prune the recurrent connections between modules directly rather than between neurons. To ensure the basic performance of the model, the two-way strategy first retains the recurrent connections of the same module at adjacent moments (i.e., all elements of M_{ii} are 1). The goal is to form a simple fixed, non-random topology in the network to ensure that each module is fully connected at different times (Rodan et al., 2010). Secondly, the remaining sub-matrices (M_{ij}, i^1j) are determined according to their random probabilities to control the sparsification of recurrent connections. The goal is to make use of random connections between different modules to achieve undifferentiated pruning. This poses a new question: is it better to generate a mask matrix M for the entire sequence or a new M for each step? Considering that the candidate state affected by M will later be used for learning the module priority, it is recommended to generate a new M for each step to ensure sufficient dynamic changes in the model (Qiao et al., 2016). Figure 8(b) shows an illustration of this strategy. Different from Figure 8(a), the updated speed of each module in 8(b) is variable at different moments, so there are both slow-to-fast and fast-to-slow connections in the hidden layer of adjacent moments. Besides, the one-way strategy forces the same pruning of the recurrent connections, while the two-way strategy allows different pruning at different times.

To sum up, as a general framework with modules as the minimum updated unit, M-RNN needs to pay attention to the following two parts. One is the updated strategy of hidden modules, including fixed strategy, random strategy, and adaptive strategy without order or with the order. The other is the

Figure 8. Illustration of two pruning strategies of recurrent connections for M-RNN; for (a), the larger the index of the module, the slower the update; for (b), the fast and slow modules cannot be strictly separated, because the updated frequency of the module at different times is different



pruning strategy of recurrent connections, including one-way strategy and two-way strategy. According to different combinations of the above two parts, four models can be obtained as shown in Table 1.

EXPERIMENTS AND RESULTS

In this section, all the above M-RNNs are compared with some popular gated RNNs for electric load forecasting. All models are implemented with Tensorflow. The experiments are performed according to the following principles: 1) all models contain only one hidden layer and have the same number of neurons; 2) the output layer is added only to the last moment of the sequence; 3) no tricks, such as recurrent dropout (Semeniuta et al., 2016), batch normalization (Laurent et al., 2016), gradient clipping (Pascanu et al., 2013), etc., are adopted.

Experimental Settings

A real-world dataset provided by the Australian Energy Market Operator is used for the one-day-ahead prediction of the load. The daily maximum load values from 2014 to 2017 in Queensland are selected for the experiment, among which the data from 2014 to 2016 are taken as the training set, the first six months of 2017 as the validation set, and the last six months as the test set. Z-Score standardization is adopted for data preprocessing to make its distribution more regular. To investigate the memory capacity of each model for long-term information, the time step is set to 365(the length of a year). The number of hidden neurons is set to 210. For M-RNN, the hidden layer is divided into 7 modules, with 30 neurons in each module. The initial weights are drawn from a truncated normal distribution with zero mean and a standard deviation of 0.01. All models are trained for 20000 epochs using RMSProp optimizer, where the learning rate could be optimized in $\{10^{-3}, 10^{-4}, 10^{-5}\}$ and the decay rate is set to 0.9. Other settings are as follows: the batch size=128; the updated threshold ε=0.5; and the pruning threshold p=0.5. Finally, the number of parameters (NP), mean absolute percentage error (MAPE), and normalized root means square error (NRMSE) are used to comprehensively evaluate the performance of each model. The equations of MAPE and NRMSE are as follows:

$$MAPE = \left\langle \frac{\left| y_t - y_t^* \right|}{y_t^*} \right\rangle \times 100\%, \tag{11}$$

$$NRMSE = \frac{\sqrt{\left\langle (y_t - y_t^*)^2 \right\rangle}}{\sqrt{\left\langle (y_t - \left\langle y_t^* \right\rangle)^2 \right\rangle}}, \tag{12}$$

where $\langle \bullet \rangle$ is the operation of the mean; y_t and y_t^* represent the predicted value and the ground-truth value respectively. The smaller the value of the above indicators is, the higher the predictive accuracy will be.

Table 1. Summary of multiple models belonging to M-RNN obtained from the combination of different strategies

Model	Updated Strategy of Hidden Modules	Pruning Strategy of Recurrent Connections
CW-RNN	The fixed strategy	The one-way pruning strategy
ZM-RNN	The random strategy	The two-way pruning strategy
AM-RNN-I	The adaptive strategy without order	The two-way pruning strategy
AM-RNN-II	The adaptive strategy with order	The one-way pruning strategy

Experimental Results

Table 2 shows the performance of each model over the test set. In terms of predictive accuracy, Vanilla RNN is the worst performing model due to the gradient problem of the long sequence. Among the three models based on gate mechanism, GRU has the best performance, followed by MGU, and LSTM has the worst performance. This indicates that more network parameters do not mean better performance. Finally, among the four models based on multi-timescale connections, CW-RNN is inferior to LSTM in MAPE but superior to it in NRMSE. ZM-RNN performs better than LSTM but worse than MGU and GRU. This fully illustrates the inadequacy of module updates using fixed or random strategies. Both versions of the proposed AM-RNN are superior to the aforementioned models. This indicates that the modeling capability of AM-RNN is significantly enhanced by introducing adaptive updating of modules in a data-driven way. In terms of training complexity, the difference in network architecture determines the difference in the number of training parameters. According to the structure of each model, the number of parameters can be deduced by (13)-(18).where n_i=1, n_h=210, and n_o=1 represent the number of neurons in the input, hidden and output layers respectively; k=7 refers to the number of modules into which the hidden layer is divided; n_M ($k \le n_M \le k^2$) is the number of nonzero sub-matrices in mask matrix M obtained by using the two-way pruning strategy. For the gated RNNs (including LSTM, GRU, and MGU), the number of parameters increases with the number of gates. While the number of parameters of M-RNN is not only much less than gated RNN but even less than Vanilla RNN. This is thanks to the partial update of modules and pruning of recurrent connections taken by M-RNN. Note that since ZM-RNN and AM-RNN-I prune the recurrent connection according to probability, the number of parameters is an indeterminate value. When they happen to have the same amount of weight pruning as CW-RNN, the number of parameters can be considered equal. Finally, AM-RNN-II has the same number of parameters as CW-RNN because both adopt a slow-to-fast strategy for pruning the recurrent connection. As can be seen from Table 2, AM-RNN-II requires only 58% of

Table 2. Experimental results on the Queensland load dataset (number of modules k=7)

Model	# parameters	MAPE (%)	NRMSE
Vanilla RNN	≈45K	2.6698	0.6817
Gated RNNs			
LSTM	≈181K	2.2442	0.6393
GRU	≈135K	1.8895	0.5537
MGU	≈90K	2.1157	0.5879
Modularized RNNs			
CW-RNN	≈26K	2.6414	0.6215
ZM-RNN	≈7K~45K	2.1782	0.6348
AM-RNN-I	≈7K~45K	1.6833	**0.5148**
AM-RNN-II	≈26K	**1.6764**	0.5176

$$NP_{VRNN} = n_h(n_i+1)+n_h^2+n_o(n_h+1)\text{ , (13)}$$
$$NP_{LSTM} = 4n_h(n_i+4)+4n_h^2+n_o(n_h+1)\text{ , (14)}$$
$$NP_{GRU} = 3n_h(n_i+3)+3n_h^2+n_o(n_h+1)\text{ , (15)}$$
$$NP_{MGU} = 2n_h(n_i+2)+2n_h^2+n_o(n_h+1)\text{ , (16)}$$
$$NP_{AM-RNN-II}=NP_{CW-RNN} = n_h(n_i+1)+(n_h/k)^2\,k(k+1)/2+n_o(n_h+1)\text{ , (17)}$$
$$NP_{AM-RNN-I} = NP_{ZM-RNN} = n_h(n_i+1)+(n_h/k)^2\,n_M+n_o(n_h+1)\text{ , (18)}$$

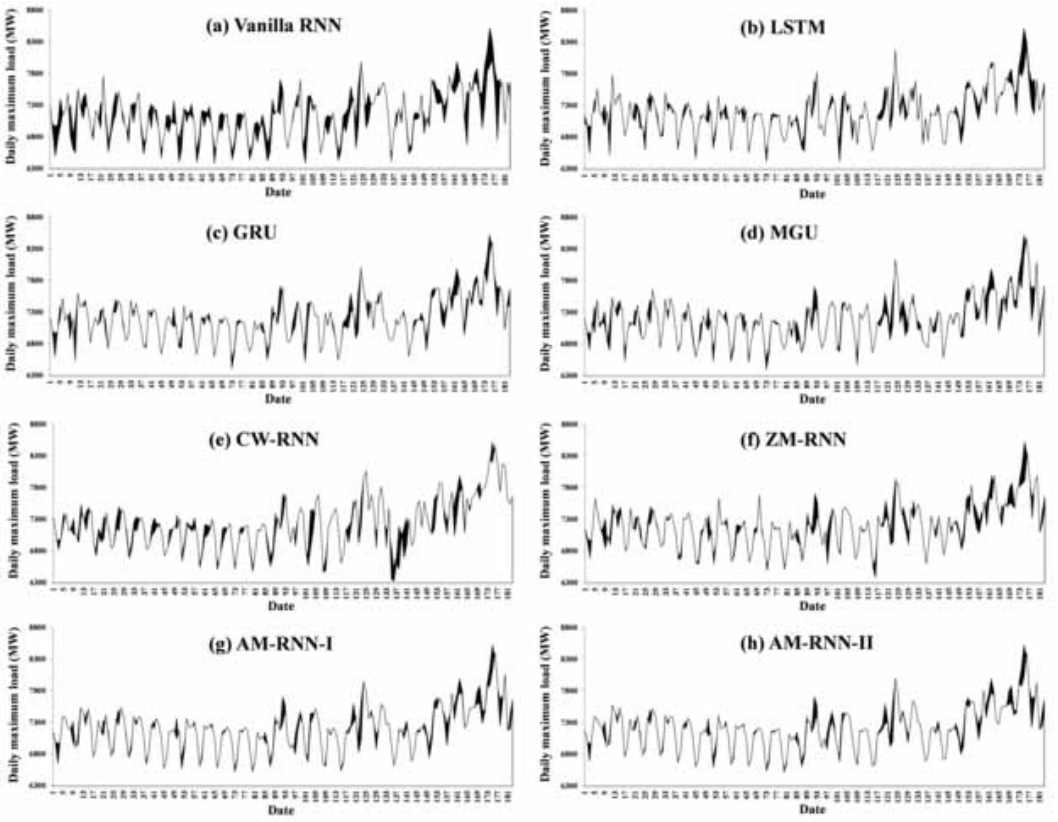
Vanilla RNN's parameters to reduce MAPE by 37%. Compared to the best-performing GRU in the gated RNN, it takes only 19% of GRU's parameters to reduce MAPE by 11%.

Figure 9 shows more detail of the residuals between the ground-truth value and the predicted value of all models. The larger the black area in the figure indicates the larger the predictive error. In particular, it can be observed that the areas with large residuals tend to be relative to the same intervals. These intervals are characterized by large abnormal fluctuations that make load changes particularly difficult to predict. Especially in the last segment, which corresponds to December 2017, due to the summer in Australia, the load shows a significant upward trend. Nevertheless, both versions of the proposed AM-RNN can still be very close to the target time series and obtain the best predictive accuracy. Note that their performance is very similar, with only minor differences. The consistent high performance of AM-RNN across different versions can be explained by its adaptive multi-timescale connectivity.

DISCUSSIONS

For M-RNN, the most important parameter is considered to be the number of modules in the hidden layer, because it determines the diversity of multi-timescale connections. To better determine the applicability of the proposed AM-RNNs, comparative experiments with different numbers of modules were carried out on the Queensland load dataset with other experimental settings unchanged. Since this parameter does not exist for Vanilla RNN and gated RNNs, Table 3 shows only the performance of multiple models belonging to M-RNN. The experimental results show that both versions of AM-RNN

Figure 9. Illustration of the residuals between the predicted value of each model and the ground-truth value of the Queensland load dataset, where the larger the black area indicates the larger the residual



continue to outperform CW-RNN and ZM-RNN based on different numbers of modules. Overall, AM-RNN-II is slightly better than AM-RNN-I.

To further evaluate the performance of AM-RNNs, additional experiments are carried out on a baseline dataset for chaotic time series prediction. Firstly, a time series containing 1700 sampling points is generated using the following Mackey-Glass equation:

$$\frac{dx(t)}{dt} = \frac{0.2x(t-\tau)}{1+x(t-\tau)^{10}} - 0.1x(t) \tag{19}$$

where the initial condition $x(0)=0.5$ and the chaotic attractor $\tau = 17$. The first 1500 points are used for training, while the last 200 points are divided equally for validation and testing. The number of hidden neurons is set to 12 and can be equally divided into 6 modules for M-RNN. The initial weights are drawn from a truncated normal distribution with zero mean and a standard deviation of 0.1. All models are trained using Adam optimizer with the learning rate 10^{-4}. All other settings remain the same as the previous experiment except that the batch size=100. To assess the dependencies of different lengths, time steps are set to 50, 100, and 150 respectively, with the results shown in Table 4. Consistent with the previous experimental results, both versions of AM-RNN show good performance. The performance of AM-RNN-II is particularly outstanding. It almost always has the best predictive accuracy regardless of the time-step size.

Table 3. Experimental results for different numbers of modules on the Queensland load dataset

Model	Number of Modules k=5		Number of Modules k=7		Number of Modules k=10	
	MAPE (%)	NRMSE	MAPE (%)	NRMSE	MAPE (%)	NRMSE
CW-RNN	2.1468	0.6599	2.6414	0.6215	2.7419	0.6707
ZM-RNN	1.9327	0.5572	2.1782	0.6348	2.2624	0.6237
AM-RNN-I	1.8055	0.5652	1.6833	**0.5148**	**1.7231**	0.5369
AM-RNN-II	**1.7138**	**0.5404**	**1.6764**	0.5176	1.7270	**0.5368**

Table 4. Experimental results of different time steps on the MG dataset

Model	Time step=50		Time step=100		Time step=150	
	MAPE (%)	NRMSE	MAPE (%)	NRMSE	MAPE (%)	NRMSE
Vanilla RNN	1.20E-1	5.68E-3	7.10E-2	3.31E-3	8.39E-2	4.02E-3
Gated RNNs						
LSTM	1.13E-1	5.17E-3	6.94E-2	3.38E-3	7.21E-2	3.70E-3
GRU	3.85E-2	2.17E-3	5.03E-2	2.42E-3	2.37E-2	1.33E-3
MGU	4.05E-2	1.88E-3	5.19E-2	2.44E-3	3.29E-2	1.55E-3
Modularized RNNs						
CW-RNN	3.77E-2	1.83E-3	7.02E-2	3.26E-3	6.42E-2	3.07E-3
ZM-RNN	3.31E-2	1.61E-3	4.35E-2	2.11E-3	5.34E-2	2.57E-3
AM-RNN-I	3.24E-2	**1.55E-3**	3.32E-2	1.61E-3	3.28E-2	1.68E-3
AM-RNN-II	**3.12E-2**	1.57E-3	**2.15E-2**	**9.80E-4**	**1.77E-2**	**8.50E-4**

CONCLUSION

RNNs have gained widespread attention in many time-related tasks, such as electric load forecasting studied in this article. However, capturing complex time dependencies in sequence data, especially long-term dependencies, remains an open challenge for RNNs. By introducing the skip length of the module, this article mainly studies the general framework called the modularized RNN (M-RNN). The adaptive M-RNN (AM-RNN) is designed under this framework, which can capture long-term dependencies adaptively thanks to the multi-timescale recurrent connections. The main feature of AM-RNN is to dynamically adjust the updated frequency of each module by calculating its priority. By reducing the number of the module being updated to obtain longer skip lengths, AM-RNN provides shortcuts for gradient propagation. Finally, two versions of AM-RNN are obtained by combining the updated strategy of hidden modules and the pruning strategy of recurrent connections, and their superiority is proved by experiments. The experimental results demonstrate that AM-RNN-II is superior not only to the Vanilla RNN and the popular gated RNNs but also to the existing multi-timescale RNNs. In the future, the authors will further study the application of AM-RNN in other fields of smart cities, such as traffic flow prediction and air quality prediction.

ACKNOWLEDGMENT

The authors wish to thank the reviewers for their thorough and helpful remarks. This research was supported by the National Natural Science Foundation of China [grant numbers 61772136, 61672159].

REFERENCES

Arjovsky, M., Shah, A., & Bengio, Y. (2016). Unitary evolution recurrent neural networks. In *Proceedings of International Conference on Machine Learning* (pp. 1120-1128).

Bendaoud, N. M. M., & Farah, N. (2020). Using deep learning for short-term load forecasting. *Neural Computing & Applications*, *32*(18), 15029–15041. doi:10.1007/s00521-020-04856-0

Bianchi, F. M., Maiorino, E., Kampffmeyer, M. C., Rizzi, A., & Jenssen, R. (2017). *Recurrent neural networks for short-term load forecasting: an overview and comparative analysis*. Springer. doi:10.1007/978-3-319-70338-1

Campos, V., Jou, B., Giró-i-Nieto, X., Torres, J., & Chang, S. F. (2018). Skip RNN: Learning to skip state updates in recurrent neural networks. In *Proceedings of International Conference on Learning Representations* (pp. 1-17).

Chang, S., Zhang, Y., Han, W., Yu, M., Guo, X., Tan, W., & Huang, T. S. (2017). Dilated recurrent neural networks. In Proceedings of Advances in Neural Information Processing Systems (pp. 77-87).

Chen, Y., Kloft, M., Yang, Y., Li, C., & Li, L. (2018). Mixed kernel based extreme learning machine for electric load forecasting. *Neurocomputing*, *312*, 90–106. doi:10.1016/j.neucom.2018.05.068

Cheng, Z., Xu, Y., Cheng, M., Qiao, Y., Pu, S., Niu, Y., & Wu, F. (2020). Refined gate: A simple and effective gating mechanism for recurrent units. *arXiv preprint arXiv:2002.11338*.

Cho, K., van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing* (pp. 1724-1734). ACL. doi:10.3115/v1/D14-1179

Dudek, G., Pełka, P., & Smyl, S. (2021). A hybrid residual dilated LSTM and exponential smoothing model for midterm electric load forecasting. *IEEE Transactions on Neural Networks and Learning Systems*. doi:10.1109/TNNLS.2020.3046629 PMID:33417572

Elman, J. L. (1990). Finding structure in time. *Cognitive Science*, *14*(2), 179–211. doi:10.1207/s15516709cog1402_1

Fekri, M. N., Patel, H., Grolinger, K., & Sharma, V. (2021). Deep learning for load forecasting with smart meter data: Online adaptive recurrent neural network. *Applied Energy*, *282*, 116177. doi:10.1016/j.apenergy.2020.116177

Fernando, T., Denman, S., McFadyen, A., Sridharan, S., & Fookes, C. (2018). Tree memory networks for modelling long-term temporal dependencies. *Neurocomputing*, *304*, 64–81. doi:10.1016/j.neucom.2018.03.040

Gallicchio, C., Micheli, A., & Pedrelli, L. (2017). Deep reservoir computing: A critical experimental analysis. *Neurocomputing*, *268*, 87–99. doi:10.1016/j.neucom.2016.12.089

Greff, K., Srivastava, R. K., Koutník, J., Steunebrink, B. R., & Schmidhuber, J. (2016). LSTM: A search space odyssey. *IEEE Transactions on Neural Networks and Learning Systems*, *28*(10), 2222–2232. doi:10.1109/TNNLS.2016.2582924 PMID:27411231

Hafeez, G., Alimgeer, K. S., & Khan, I. (2020). Electric load forecasting based on deep learning and optimized by heuristic algorithm in smart grid. *Applied Energy*, *269*, 114915. doi:10.1016/j.apenergy.2020.114915

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 770-778). IEEE.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, *9*(8), 1735–1780. doi:10.1162/neco.1997.9.8.1735 PMID:9377276

Hong, T., & Fan, S. (2016). Probabilistic electric load forecasting: A tutorial review. *International Journal of Forecasting*, *32*(3), 914–938. doi:10.1016/j.ijforecast.2015.11.011

Hu, H., Wang, L., Peng, L., & Zeng, Y. R. (2020). Effective energy consumption forecasting using enhanced bagged echo state network. *Energy*, *193*, 116778. doi:10.1016/j.energy.2019.116778

Huang, F., Zhuang, S., & Yu, Z. (2019). Power load prediction based on an improved clock-work RNN. In *Proceedings of International Conference on Ubiquitous Intelligence and Computing* (pp. 596-601). IEEE. doi:10.1109/SmartWorld-UIC-ATC-SCALCOM-IOP-SCI.2019.00140

Huang, Q., Li, J., & Zhu, M. (2020). An improved convolutional neural network with load range discretization for probabilistic load forecasting. *Energy*, *203*, 117902. doi:10.1016/j.energy.2020.117902

Jing, L., Gulcehre, C., Peurifoy, J., Shen, Y., Tegmark, M., Soljacic, M., & Bengio, Y. (2019). Gated orthogonal recurrent units: On learning to forget. *Neural Computation*, *31*(4), 765–783. doi:10.1162/neco_a_01174 PMID:30764742

Koutnik, J., Greff, K., Gomez, F., & Schmidhuber, J. (2014). A clockwork RNN. In *Proceedings of International Conference on Machine Learning* (pp. 1863-1871). IEEE.

Krueger, D., Maharaj, T., Kramár, J., Pezeshki, M., Ballas, N., Ke, N. R., & Pal, C. (2017). Zoneout: Regularizing rnns by randomly preserving hidden activations. In *Proceedings of International Conference on Learning Representations* (pp. 1-11). IEEE.

Laurent, C., Pereyra, G., Brakel, P., Zhang, Y., & Bengio, Y. (2016). Batch normalized recurrent neural networks. In *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing* (pp. 2657-2661). IEEE.

Li, J., Deng, D., Zhao, J., Cai, D., Hu, W., Zhang, M., & Huang, Q. (2021). A novel hybrid short-term load forecasting method of smart grid using MLR and LSTM neural Network. *IEEE Transactions on Industrial Informatics*, *17*(4), 2443–2452. doi:10.1109/TII.2020.3000184

Martens, J., & Sutskever, I. (2012). Training deep and recurrent networks with hessian-free optimization. In *Neural networks: Tricks of the trade* (pp. 479–535). Springer. doi:10.1007/978-3-642-35289-8_27

Moirangthem, D. S., & Lee, M. (2021). Hierarchical and lateral multiple timescales gated recurrent units with pre-trained encoder for long text classification. *Expert Systems with Applications*, *165*, 113898. doi:10.1016/j.eswa.2020.113898

Mordjaoui, M., Haddad, S., Medoued, A., & Laouafi, A. (2017). Electric load forecasting by using dynamic neural network. *International Journal of Hydrogen Energy*, *42*(28), 17655–17663. doi:10.1016/j.ijhydene.2017.03.101

Ouyang, T., He, Y., Li, H., Sun, Z., & Baek, S. (2019). Modeling and forecasting short-term power load with copula model and deep belief network. *IEEE Transactions on Emerging Topics in Computational Intelligence*, *3*(2), 127–136. doi:10.1109/TETCI.2018.2880511

Pascanu, R., Mikolov, T., & Bengio, Y. (2013). On the difficulty of training recurrent neural networks. In *Proceedings of International Conference on Machine Learning* (pp. 1310-1318).

Patel, R., Patel, M. R., & Patel, R. V. (2019). A review: Introduction and understanding of load forecasting. *Journal of Applied Science and Computations*, *4*, 1449–1457.

Qiao, J., Li, F., Han, H., & Li, W. (2016). Growing echo-state network with multiple subreservoirs. *IEEE Transactions on Neural Networks and Learning Systems*, *28*(2), 391–404. doi:10.1109/TNNLS.2016.2514275 PMID:26800553

Rodan, A., & Tino, P. (2010). Minimum complexity echo state network. *IEEE Transactions on Neural Networks*, *22*(1), 131–144. doi:10.1109/TNN.2010.2089641 PMID:21075721

Schoene, A. M., Turner, A., De Mel, G. R., & Dethlefs, N. (2021). Hierarchical Multiscale Recurrent Neural Networks for Detecting Suicide Notes. *IEEE Transactions on Affective Computing*. Advance online publication. doi:10.1109/TAFFC.2021.3057105

Semeniuta, S., Severyn, A., & Barth, E. (2016). Recurrent dropout without memory loss. In *Proceedings of International Conference on Computational Linguistics: Technical Papers* (pp. 1757-1766). ACL.

Shen, Y., Tan, S., Sordoni, A., & Courville, A. (2019). Ordered neurons: Integrating tree structures into recurrent neural networks. In *Proceedings of International Conference on Learning Representations* (pp. 1-14). OpenReview.

Shi, H., Xu, M., & Li, R. (2018). Deep learning for household load forecasting—A novel pooling deep RNN. *IEEE Transactions on Smart Grid*, *9*(5), 5271–5280. doi:10.1109/TSG.2017.2686012

Vorontsov, E., Trabelsi, C., Kadoury, S., & Pal, C. (2017). On orthogonality and learning recurrent networks with long term dependencies. In *Proceedings of International Conference on Machine Learning* (pp. 3570-3578). MLR Press.

Yu, Y., Si, X., Hu, C., & Zhang, J. (2019). A review of recurrent neural networks: LSTM cells and network architectures. *Neural Computation*, *31*(7), 1235–1270. doi:10.1162/neco_a_01199 PMID:31113301

Yu, Z., Zheng, X., Huang, F., Guo, W., Sun, L., & Yu, Z. (2020). A framework based on sparse representation model for time series prediction in smart city. *Frontiers of Computer Science*, *15*(1), 1–13.

Zhao, X., Gao, W., Qian, F., & Ge, J. (2021). Electricity cost comparison of dynamic pricing model based on load forecasting in home energy management system. *Energy*, *229*, 120538. doi:10.1016/j.energy.2021.120538

Zhou, G. B., Wu, J., Zhang, C. L., & Zhou, Z. H. (2016). Minimal gated unit for recurrent neural networks. *International Journal of Automation and Computing*, *13*(3), 226–234. doi:10.1007/s11633-016-1006-2

Zhuang, S., & Yu, Z., Guo w., & Huang, F. (2020). Short term load forecasting via zoneout-based multi-time scale recurrent neural network. *Computer Science*, *47*(9), 105–109.

Fangwan Huang is a senior lecturer at the College of Mathematics and Computer Science, Fuzhou University, China. She received the B.S. and M.E. degrees in Computer Science and technology from Fuzhou University, China in 2002 and 2005. Fuzhou University, China in 2022. Her research interests include computational intelligence, big data analysis, and mobile crowdsensing.

Shijie Zhuang is a student at the College of Mathematics and Computer Science, Fuzhou University, China. He received the B.S. and M.E. degrees in Computer Science and technology from Fuzhou University, China in 2017 and 2020. His research interests include computational intelligence and deep learning.

Zhiyong Yu is a full professor in the College of Mathematics and Computer Science, Fuzhou University, Fuzhou, China. He received the M.E. and Ph.D. degrees in computer science and technology from Northwestern Polytechnical University, Xi'an, China in 2007 and 2011, respectively. He was also a visiting student at Kyoto University, Kyoto, Japan, from 2007 to 2009, and a visiting researcher at the Institut Mines-Telecom, TELECOM SudParis, Evry, France, from 2012 to 2013. His current research interests include pervasive computing, mobile social networks, and mobile crowdsensing.

Yuzhong Chen received the B.S. and Ph.D. degrees in communication and information system from the University of Science and Technology of China, in 2000 and 2005, respectively. He is currently a Professor with the College of Mathematics and Computer Science, Fuzhou University. His current research interests include computational intelligence, natural language processing, and data mining. He is also the Vice Chief of the Fujian Provincial Key Laboratory of Network Computing and Intelligent Information Processing.

Kun Guo is currently an associate professor with the College of Mathematics and Computer Science at Fuzhou University. He received the M.E. and Ph.D. degrees in computer science and technology and management from Fuzhou University, Fuzhou, China in 2005 and 2012, respectively. He was also a visiting scholar at Hong Kong University of Science and Technology, Hong Kong, China, from 2019 to 2020. He is a member of China Computer Federation (CCF) and Fujian Provincial Key Laboratory of Network Computing and Intelligent Information Processing. His research interests include complex network data mining, distributed parallel computation and grey system theory.