energies

Article

Analysis of Different Neural Networks and a New Architecture for Short-Term Load Forecasting

Lintao Yang and Honggeng Yang *

College of Electrical Engineering and Information Technology, Sichuan University, Chengdu 610065, China; yanglintao@stu.scu.edu.cn

* Correspondence: yanghonggeng@scu.edu.cn or pqlab99@126.com

Received: 22 January 2019; Accepted: 9 April 2019; Published: 14 April 2019

Abstract: Short-term load forecasting (STLF) has been widely studied because it plays a very important role in improving the economy and security of electric system operations. Many types of neural networks have been successfully used for STLF. In most of these methods, common neural networks were used, but without a systematic comparative analysis. In this paper, we first compare the most frequently used neural networks' performance on the load dataset from the State Grid Sichuan Electric Power Company (China). Then, considering the current neural networks' disadvantages, we propose a new architecture called a gate-recurrent neural network (RNN) based on an RNN for STLF. By evaluating all the methods on our dataset, the results demonstrate that the performance of different neural network methods are related to the data time scale, and our proposed method is more accurate on a much shorter time scale, particularly when the time scale is smaller than 20 min.

Keywords: short-term load forecasting; back-propagation neural network; recurrent neural network; long-short term memory; gate-recurrent neural network

1. Introduction

Accurate short-term load forecasting (STLF) can play a significant role in power construction planning and power grid operation, and also has crucial implications for the sustainable development of power enterprises. STLF can predict future loads for minutes to weeks. Because of the nonlinearity, non-stationarity, and non-seasonality of STLF, it is very challenging to predict accurately. Inaccurate load forecasting may increase operating costs [1]. By contrast, with an accurate electric load forecasting method, fundamental operating functions, such as unit maintenance, reliability analysis, and unit commitment, can be operated more efficiently [2]. Thus, it is essential for power suppliers to build an effective model that can predict power loads, accomplish a balance between production and demand, reduce production costs, and implement pricing schemes for various demand responses. According to the length of the forecast period, power load forecasting is divided into four categories: long-term load forecasting, medium-term load forecasting, STLF, and ultra-STLF [3].

There have been many efforts to develop accurate STLF; many methods have been propsoed [4–16]. In much earlier works, researchers attempted to forecast load precisely using a mathematical statistics approach. The most representative is the regression analysis approach [4], which uses a set of functional linear regression models. In [5], the authors used the Kalman filter to develop a very short-term load predictor, and the Box–Jenkins autoregressive integrated moving average approach was proposed in [6]. In [17], the relationships between demand and driver variables were evaluated by using semi-parametric additive models. Additionally, in [18], the authors came up with a new SVD-based exponential smoothing formulation. Based on linear regression and patterns, an univariate models were proposed for 34 daily cycles of a load time series in [19]. In [20], by combining a Bayesian neural network with the hybrid Monte Carlo algorithm, the authors assumed a new model for STLF. Modified artificial bee colony algorithm, extreme learning machine, and the wavelet transform were combined to construct a novel STLF method in [21].

However, mathematical statistics approaches are based on linear analysis; thus, it is difficult for them to predict nonlinear and non-stationary problems. For better learning of nonlinear features, machine learning is a good approach. Two broad categories of methods exist: support vector regression (SVR) and artificial neural networks (ANNs). In the SVR approach, a generic strategy for STLF based on the SVR has been proposed [22–25]. In this method, there are two considerable improvements to SVR-based load forecasting methods: the procedure for generating model inputs and the subsequent model input was selected by feature selection algorithms. By combining SVR with other algorithms, many hybrid methods have been proposed. An SVR model combined with the differential empirical mode decomposition algorithm and autoregression was proposed in [26–28], and provides higher accuracy and interpretability, and better generation ability. To extend the SVR method, a chaotic genetic algorithm was presented to improve forecasting performance in [29]. In [30,31], the authors combined support vector machines (SVMs) with a genetic algorithm, and combined SVR with ant colony optimization to forecast the system load. In [32],an attempt based on double seasonal exponential smoothing was used for STLF. Besides these studies, in [7], in order to achieve better regression and forecasting performance, the proposed SVR-based STLF approach, a supervised machine learning approach with the preprocessing of input data is required. Simultaneously, STLF models were developed using fuzzy logic and an adaptive neuro-fuzzy inference system [33], with efficient load forecasting and has been the alternative approach for STLF in Turkey.

For ANNs, the backpropagation neural network (BPNN) was the first ANN method used for load forecasting. In [34], the authors presented a BPNN approach with a rough set for complicated STLF with dynamic and nonlinear factors to enhance the performance of predictions. By combining a Bayesian neural network and BPNN, Ningl et al. [35] proposed a Bayesian-backpropagation method to forecast the hour power load of weekdays and weekends. Based on the BPNN, the authors discussed the relationship between the daily load and weather factors in [36]. Because the BPNN is a type of feedforward ANN, it cannot learn the features of time sequential data, but power load data can be considered as sequential data. Recurrent neural networks (RNNs) have been introduced into STLF. In [37–40], the authors proposed using an RNN to capture a compact and robust representation. A multiscale bi-linear RNN was proposed for STLF [41]. Additionally, a long-short term memory (LSTM) network as a type of more complex RNN has been used in STLF [42–44]. However, the performance of LSTM is not very effective.

Although neural networks have been frequently used in short-term power load prediction, there barely has been a systematic comparison of the role of neural networks in this problem to determine how to solve the key problems and which approach exerted the least negative effect on the neural network. Therefore, we systematically compare the advantages and disadvantages of different types of commonly used neural network methods and then analyse the performance of neural networks for different time scales. Simultaneously, according to the difference in network performance, we design new RNN architecture that balance memory and the current scenario at any time by referring to highway neural networks [45] in the time dimension. Thus, our main contributions in this paper are as follows: first, we systematically analyze the performance of commonly used neural networks on our power load data. Second, according to the advantages and disadvantages of these networks, we propose a new neural network architecture, the gate-RNN, to forecast STLF.

2. Methods

To explore the performance of different types of neural networks applied to STLF, we use four types: three types of the most commonly used neural networks and an improved neural network that we call the gate-RNN.

2.1. BPNN

A BPNN is a type of feedforward neural network (FNN). The simple architecture of a BPNN is shown in Figure 1, where the neurons in the same layer of the BPNN are only connected with adjacent layers' neurons; there are no connections among neurons in the same layer. The BPNN contains three types of layers: the input layer, hidden layers, and output layer. Input layer inputs data into the neural network. Output layer outputs the neural network's computational results. The hidden layers are the layers between the input layer and output layer. The values of connections between different layers are weights denoted by w^i where i denotes i-th layer. All the knowledge that the neural network has learned is stored in the weights.

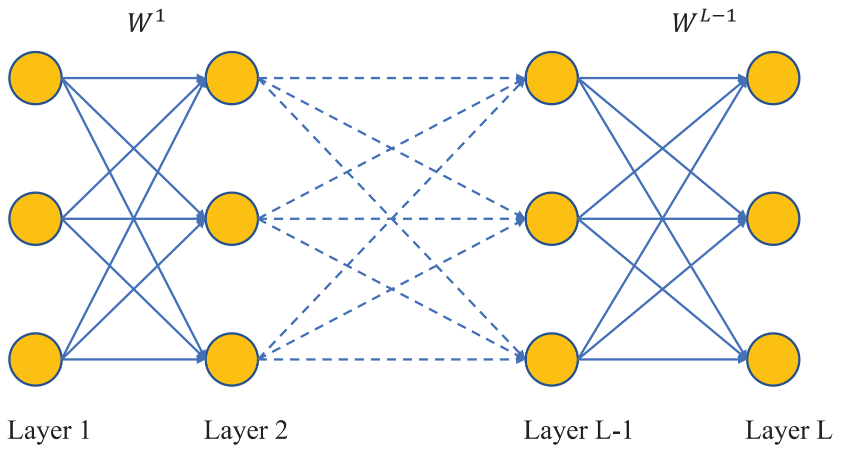

Figure 1. Architecture of a BPNN that contains L layers. Typically, Layer 1 is the input layer, which inputs data into the neural network, and the last layer, Layer L, is the output layer, which outputs the predicted values. The W between every pair of layers is the weight, which is the knowledge of the network.

The goal of training a BPNN is to determine a set of suitable W so that the network can obtain the correct output when test data is input into it by training on the train dataset. Suppose there is a training set $\langle (x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n) \rangle$, which contains n tuples. Each tuple contains input data x_i and target label y_i. To train a BPNN, the first step is forward computing, which can be computed as

$$\begin{cases} z^{l+1} = W^l a^l, \\ a^{l+1} = f(z^{l+1}), \end{cases} \tag{1}$$

where z^l is the l-th layer's input vector, a^l is the l-th layer's output vector, and $f(\cdot)$ is an active function. In this paper, we chose the rectified linear unit as our active function, which is defined as

$$f(x) = \begin{cases} x, & if \quad x >= 0, \\ 0, & else. \end{cases} \tag{2}$$

After forward computing, the BP networks need to update the network through the losses that were calculated from the target labels so that the network can determine the suitable W, where W is a set of $\{W^1, \ldots, W^L\}$ and L is the number of layers. One of the most commonly used update methods is the gradient descent method with BP. The BP process starts by defining a loss function, Loss. The loss function measures the distance between the outputs of the BP network and the true targets. The mean-square error (MSE) is a common loss function for prediction. It is defined as Equation (3):

$$Loss = \frac{1}{2n_L} \sum_{i=1}^{n_L} \left\| y_i^L(x) - a_i^L \right\|^2, \tag{3}$$

where y_i^L is the i-th output of the output layer, a_i^L is the i-th target label of output layer, and n_L is the total number of outputs in the output layer. $\frac{1}{2}$ is for better computing the derivative of loss. Because a_i^L is computed by W, loss function $Loss$ is the function of weight W.

To use the gradient descent method to determine the optimal W, we need to compute $\bigtriangledown W$ which is defined as

$$\bigtriangledown W = \frac{\partial Loss}{\partial W}, \tag{4}$$

and update W using

$$W = W - \alpha \bigtriangledown W, \tag{5}$$

where α is the learning rate, which controls the learning step during each update. We define an error term in the i-th neuron l-th layer as $\delta_i^l = \frac{\partial L}{\partial z_i^l}$ for better computation of gradient $\bigtriangledown W$ in each layer.

In the output layer, we can compute the gradient of w_{ij}^L directly by combining Equation (3) and using the chain rule:

$$\frac{\partial Loss}{\partial w^L} = \frac{\partial Loss}{\partial z^L}\frac{\partial z^L}{\partial w^L} = \delta^L (a^{L-1})^T. \tag{6}$$

Then, we compute the other layers' error terms using back propagation:

$$\delta^l = \frac{\partial Loss}{\partial z^l} = \frac{\partial Loss}{\partial z^{l+1}}\frac{\partial z^{l+1}}{\partial z^l} = \dot{f}(z^l)((w^l)^T \delta_j^{l+1}). \tag{7}$$

Then, we compute the remaining layers' weight gradients from back to front in matrix form as

$$\bigtriangledown W^l = \delta^{l+1} (a^l)^T. \tag{8}$$

Thus, the learning process of the BPNN can be presented as Algorithm 1.

Algorithm 1: The BP network update process with gradient decent.

```
Input :
        The input dataset $D = \{(x, y^L)\}$;
        The learning rate $\alpha = 0.0001$;
Output:
        The weight of the BP network $W$ after training ;
1  initialize the model parameter $W$ with uniform $\left(-\sqrt{6}/k, \sqrt{6}/k\right)$, where $k$ is the sum of input
    and output dimensions;
2  For each sample $(x, y^L) \in D$, set $a^1 = x$ ;
3  for $l = 1 : L$ do
4      $z^(l+1) = w^l a^l$ ;
5      $a^{l+1} = f(z^{l+1})$ ;
6  end
7  Compute loss by Equation (3);
8  Compute the output layer error term by $\delta^L = \frac{\partial L}{\partial z^L}$;
9  for $l = L - 1 : 1$ do
10     $\delta^l = ((w^l)^T \delta^{l+1}) \dot{f}(z^l)$;
11 end
12 Compute $\bigtriangledown w$ in each layer by Equation (8);
13 Update $w$ by Equation (5) ;
14 Repeat to line 2 until converge.
```

2.2. RNN

STLF is a type of prediction based on the previous time. It is based on the history load information forecasting the next time load value. We can consider it as a sequential problem. Among all types of neural networks, the RNN is good at solving sequential problems. An RNN contains recurrent connections. A simple architecture of an RNN is shown in Figure 2 . It contains an input layer, recurrent layer, and output layer. The difference between an RNN and BPNN is that an RNN has connections among the same recurrent layer's neurons.

For a better understanding of the computation of an RNN, we can unroll the RNN on the time dimension as shown in Figure 3. In the unrolled RNN, the neurons in the hidden layer at each time step t can be considered as one layer of an FNN. If the time step is T, then the unrolled RNN has T hidden layers. Based on the unrolled network, we can train the RNN using backpropagation through time (BPTT) [46], specifically epoch-wise BPTT, using the following steps:

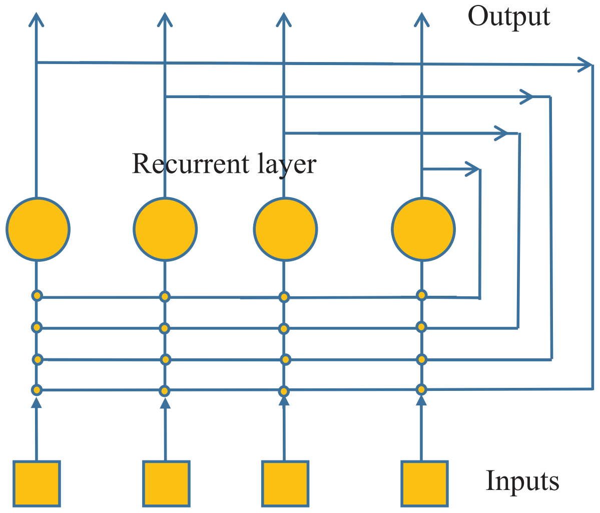

Figure 2. Architecture of a simple RNN: the inputs connect with each hidden neuron, and the hidden neurons have connections between every other neuron and have outputs.

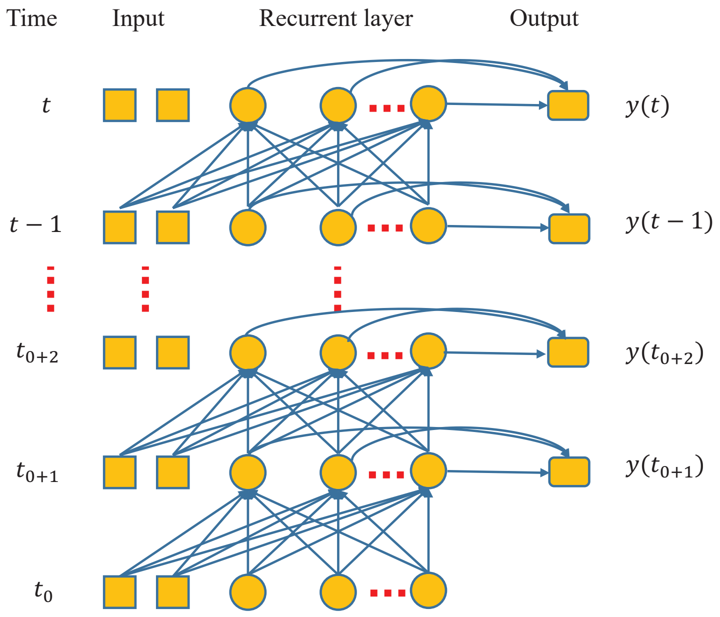

Figure 3. Unrolled architecture of an RNN: it contains t time steps. The neurons at each time step constitute one layer of the unrolled RNN. This unrolled RNN contains t layers, and at each time step t, the output is $y(t)$.

1. Forward computing: the output of network at time t can be computed as the following equation:

$$\begin{cases} s(t+1) = x(t) + wy(t), \\ y(t+1) = f(s(t+1)). \end{cases} \tag{9}$$

2. Define the loss as

$$E^{total}_{(t',t)} = \sum_{\tau=t'+1}^{t} E(\tau), \tau \in [t_0+1, t], \tag{10}$$

where $E(t) = -\frac{1}{2}\left[d(t') - y(t')\right]^2$ is the error at time t'. At each time step, $d(t')$ represents the target label and $y(t')$ is the output of the network at time t'.

3. Compute gradient $\bigtriangledown w$ of w as follows:

$$\bigtriangledown w = \frac{\partial E^{total}_{(t_0,t_1)}}{\partial w} = \sum_{\tau=t_0+1}^{t} \frac{\partial E^{total}_{(t_0,t_1)}}{\partial w(\tau)}. \tag{11}$$

Combine with the chain rule and define $e(\tau) = \frac{\partial E(\tau)}{\partial y(\tau)}$, which can be written as

$$\frac{\partial E^{total}_{(t_0,t_1)}}{\partial w(\tau)} = \delta(\tau)(y(\tau-1))^T, \tag{12}$$

where

$$\delta_k(\tau) = \begin{cases} \dot{f}(s(\tau))e(\tau), & when \quad \tau = t, \\ \dot{f}(s(\tau))(e(\tau) + w^T\delta(\tau+1)), & when \quad t_0 < \tau < t. \end{cases} \tag{13}$$

Hence, the gradients of weights can be computed as

$$\bigtriangledown w = \delta(\tau)(y(\tau-1))^T. \tag{14}$$

4. Then, update the weights in the RNN using Equation (5) until the network converges.

2.3. LSTM

LSTM is a type of RNN, but it has a more complicated architecture. A common LSTM network consists of many LSTM blocks, which are called memory cells. An LSTM cell stores the input for some period of time. The flow of information into and out of the cell was determined by the values in the cell and regulated by the three gates. A classical cell architecture is shown in Figure 4. The cell receives input x_t at time t, output h_{t-1} and state vector C_{t-1} at time $t-1$. There are three gates in the cell to control the computation of the cell: the input gate, output gate and forget gate. The input gate selectively records new information into the cell state. The output gate determines which information is worth outputting. The forget gate selectively forgets some information and retains much more valuable information. The forward pass of an LSTM unit with a forget gate can be computed as following equations

$$\begin{cases} f_t = \sigma_g(W_f x_t + U_f h_{t-1}), \\ i_t = \sigma_g(W_i x_t + U_i h_{t-1}), \\ o_t = \sigma_g(W_o x_t + U_o h_{t-1}), \\ c_t = f_t \cdot c_{t-1} + i_t \cdot \sigma_c(W_c x_t + U_c h_{t-1}), \\ h_t = o_t \cdot \sigma_h(c_t), \end{cases} \tag{15}$$

where x_t is the input vector to the cell. $W \in \mathbb{R}^{h \times d}$ and $U \in \mathbb{R}^{h \times h}$ are the weight matrices and bias vector parameters, where h and d refer to the number of input features and number of hidden units. f_t, i_i and O_t are the forget gate, input gate and output gate's output, respectively. h_t is the hidden state vector, which is also called the output vector. c_t is the cell state vector. σ_g, σ_c and σ_h are activation functions, where σ_g is the sigmoid function, σ_c is the hyperbolic tangent function and σ_h is the hyperbolic tangent.

To update the parameters of LSTM, a common way is using epoch-wise BPTT algoritm as the same with RNN, the detail can be found in [47].

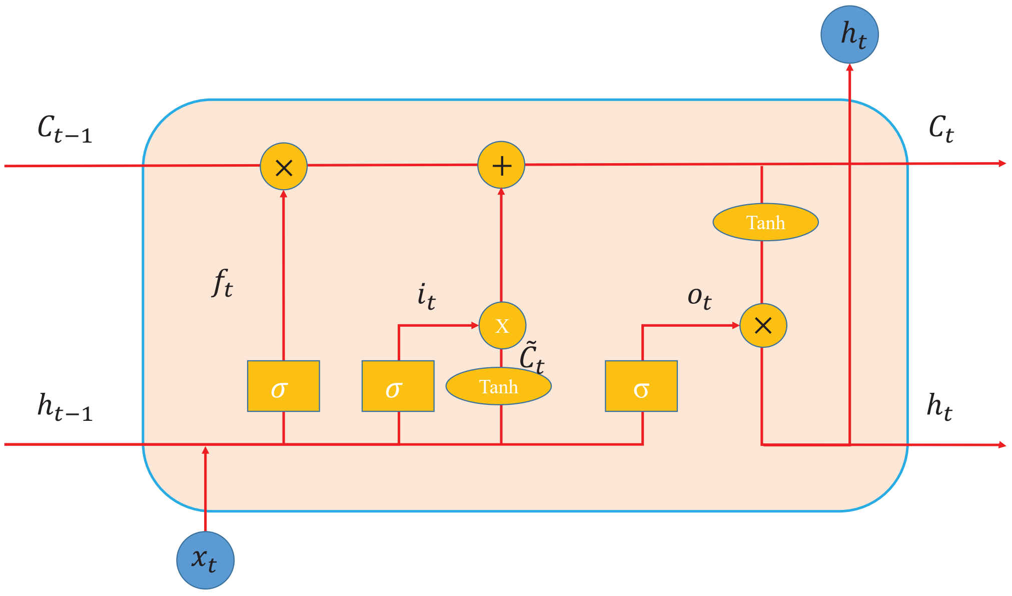

Figure 4. A cell of LSTM. The cell receives external input x_t and cell state C_{t-1} and outputs cell state C_t and current output h_t. The cell contains three gates: input gate i_t, output gate o_t and forget gate f_t.

2.4. Gate-RNN

As we can see, BPNN is an FNN. If BPNNs are used to predict load, they will weaken the relationships in the time dimension. Regarding RNN and LSTM, they are good at capturing temporal features. However, RNN can only memory history information through the weights among neurons, which is much is too simple to handle the input information and memory information. By contrast, during an experiment, we found that LSTM was too complicated to train stably. Thus, we propose an RNN cell that can control the computation of input information and memory information, converges well in the training process, and has excellent results in ultra STLF.

Figure 5 shows the cell architecture of our gate-RNN. At time t, the input vector of the cell is x_t and the history information is output h_{t-1} at time $t-1$. When input vector x_t is input into the cell, it is divided into three branches: two branches for computing two gates' values that are $1-G$ and G, and one for computing the cell's state that is S. Gate G controls the effect of the cell state on the output and gate $1-G$ controls the effect of history information on the output. The output of the cell h_t combines the output of both gates. We use the "$+$" operator as our combination approach, and it adds the values of both gates at the same position.

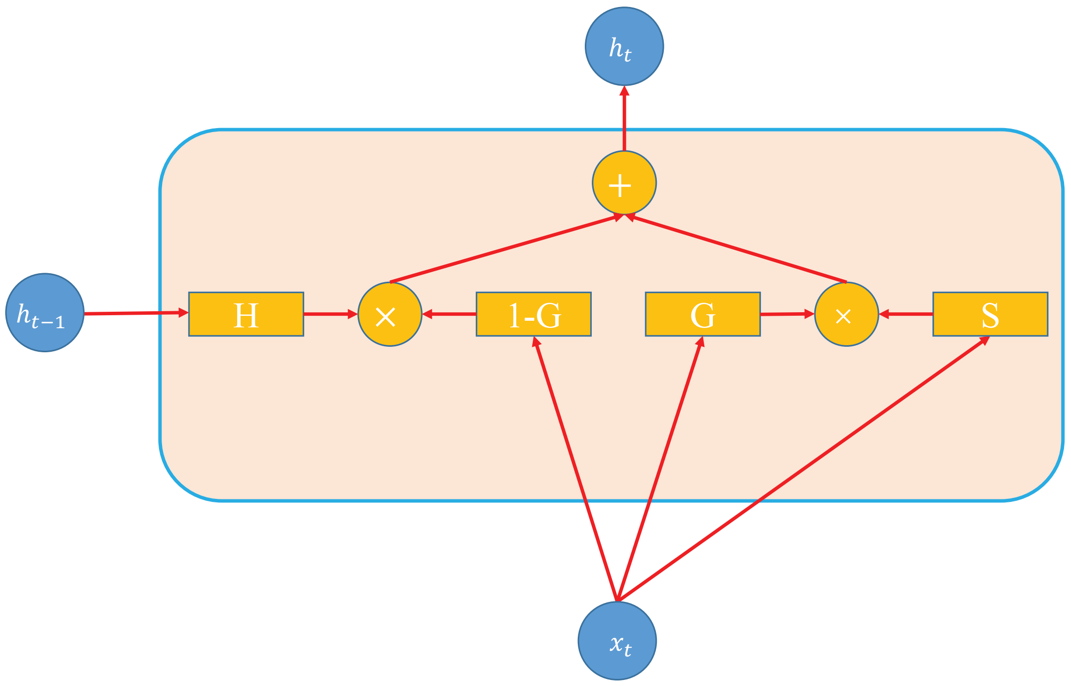

Figure 5. The cell of gate-RNN. It contains two gates to control the input information and history information. It uses G to control input information and $1 - G$ to control history information.

The computation of the cell is as follows:

$$\begin{cases} g_t = W_g x_t, \\ s_t = W_s x_t, \\ \widetilde{h}_t = W_h h_{t-1}, \\ h_t = g_t \cdot s_t + \widetilde{h}_t \cdot (1 - g_t), \end{cases} \tag{16}$$

as for LSTM, $W \in \mathbb{R}^{h \times d}$ denotes the weight matrices parameters. Many gate-RNN cells are combined to obtain a layer and then a neural network. To update the gate-RNN, we compute the gradient of the weight at each time step. We also define an error term at the t-th time step (same as the l-th layer) as $\delta_t = \frac{\partial Loss}{\partial h_t}$ to better compute the gradient of $\bigtriangledown W$ in each layer. In the t-th time step, we can compute the gradients of W_g, W_s, W_h directly by combining Equation (3) and using the chain rule, as follows:

$$\begin{aligned} \bigtriangledown W_g = \frac{\partial Loss}{\partial W_g} &= \frac{\partial Loss}{\partial h_t} \cdot \frac{\partial h_t}{\partial g_t} \cdot \frac{\partial g_t}{\partial W_g} \\ &= \delta_t \cdot ((s_t - \widetilde{h}_t) \cdot (x_t))^T, \end{aligned} \tag{17}$$

$$\begin{aligned} \bigtriangledown W_s = \frac{\partial Loss}{\partial W_s} &= \frac{\partial Loss}{\partial h_t} \cdot \frac{\partial h_t}{\partial s_t} \cdot \frac{\partial s_t}{\partial W_s} \\ &= \delta_t \cdot (g_t \cdot (x_t))^T, \end{aligned} \tag{18}$$

$$\begin{aligned} \bigtriangledown W_h = \frac{\partial Loss}{\partial W_h} &= \frac{\partial Loss}{\partial h} \cdot \frac{\partial h_t}{\partial \widetilde{h}_t} \cdot \frac{\partial \widetilde{h}_t}{\partial W_h} \\ &= \delta_t \cdot ((1 - g_t) \cdot (h_{t-1}))^T. \end{aligned} \tag{19}$$

Then, we update the entire network using Algorithm 2.

Algorithm 2: The update process of gate-RNN.

```
Input :
        The input dataset $D = \{(x, y^L)\}$;
        The learning rate $\alpha = 0.0001$;
Output:
        The weight of the gate-RNN network $\mathbf{W}$ after training ;
1 initialize the model parameter $\mathbf{W}$ with uniform $\left(-\sqrt{6}/k, \sqrt{6}/k\right)$;
2 For each sample $(x, y^L) \in D$, set $a^1 = x$ ;
3 Do forward computing by using Equation (16);
4 Compute loss by Equation (3);
5 Compute the last time step $T$ error term by $\delta_g^T = \frac{\partial Loss}{\partial g^T}, \delta_h^T = \frac{\partial Loss}{\partial h^T}, \delta_s^T = \frac{\partial L}{\partial s^T}$;
6 Compute gradients of $W_s, W_h, W_g$, by using Equation (17)–(19);
7 Update $W$ by Equation (5) ;
8 Repeat to line 2 until converge.
```

3. Experiments Results and Discussion

3.1. Experiment Data Processing and Experiment Settings

Our dataset contained an entire year's electrical load of the 2016 Electric Power Company in Sichuan Province, China. The data values distribution is shown in Figure 6. The average, variance, standard deviation and coefficient of variation are 16,965.71, 11,692,942.24, 3419.49 and 4.96, respectively. To avoid overlap between the training data and test data, all the data were sorted by time, and the first three-quarters of the total data was chosen as our training data and the remainder as our test data. It means that we used nine months' load values as the train date, the rest of the three months' load values as test sets. Because, for training neural networks, it is suggested to use raw data, we don't use any regularization or pay special attention to the special days in the whole years such as holidays and Chinese New Year day. The original data recorded electrical load with a 1 min interval. According to the data processing method in [7], the data were sampled for different time scales. For a better analysis of neural network performance, the data were sampled for 5-min, 20-min, 30-min, and 40-min time scales after dividing the training data and test data into different time scales. The data size of different time scales' train dataset and test dataset are shown in Table 1. It can be visualized as in Figure 7. As we can see, with the time scale become larger, the size of dataset become smaller. Because of sampling by different time scales, the data have been divided into different sizes. Regarding training the BPNN, the inputs of the neural networks were the last 10 samples and the output was the prediction value of the 11th moment load value. For the RNN series methods, the input was the previous nine load values and the current load value, and the output was the next time predicted load value. All the input values are the nearest ten values and ordered in time; this is the same way as mentioned in [35,43]. The complete structures of ANNs are shown in Figure 8. After predicting the eleventh load value, the second value to the eleventh value is chosen to predict the twelfth value. We iterate this procedure until finishing the prediction of the last value. The prediction mechanism of the neural network is shown in Figure 9.

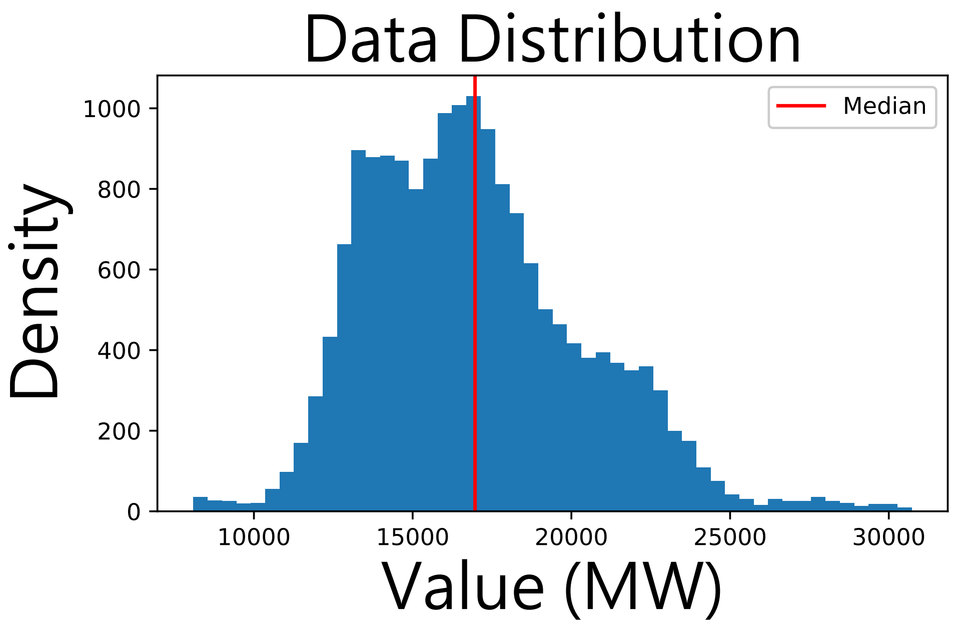

Figure 6. The distribution of raw data values.

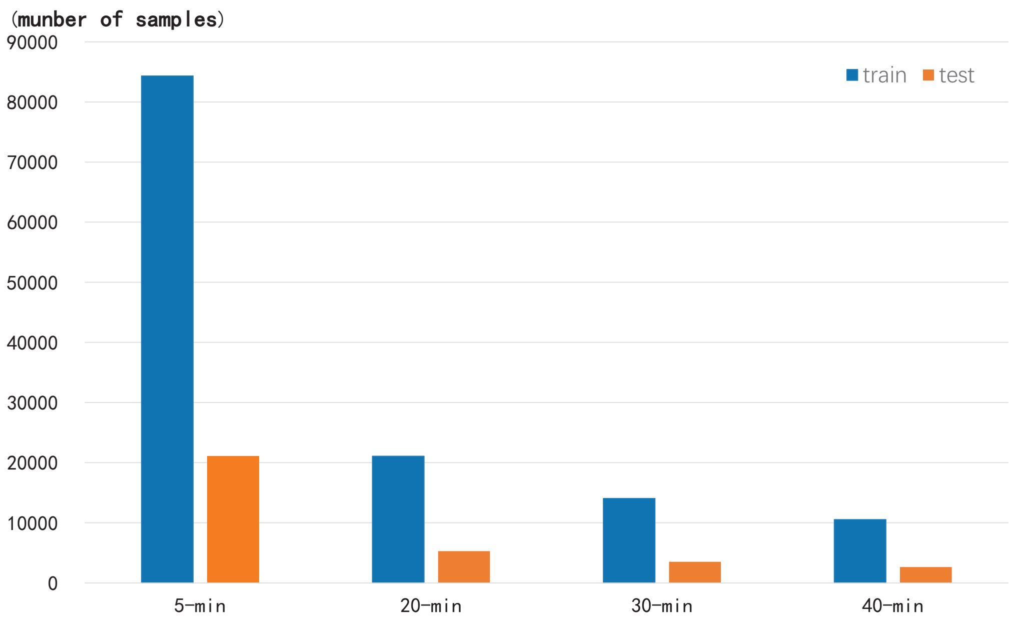

Figure 7. The numbers of training data and test data distribution at different time scales.

Table 1. The data size of different time scales.

Dataset	5 min	20 min	30 min	40 min
train	78,624	19,656	13,104	9828
test	26,496	6624	4416	3312

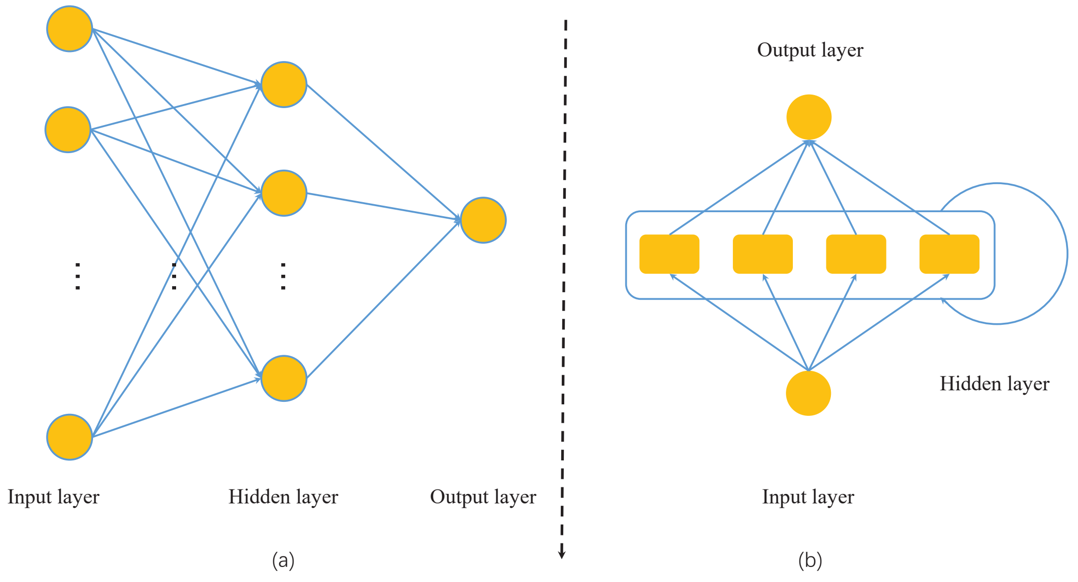

Figure 8. The complete structures of BP neural network and RNNs. (**a**) the structure of BP neural network, the input layer contains 10 neurons, hidden layer contains 64 neurons and output layer contains only one neuron; (**b**) the structure of RNNs, the input contains one neuron that input the current load value, the hidden layer contains four units and output contains one neuron which output the predict load value. If each unit in the right figure is neuron, it is the structure of RNN, if each unit is LSTM cell, then it is the structure of LSTM and if each unit is gate-RNN cell, then it is the structure of gate-RNN.

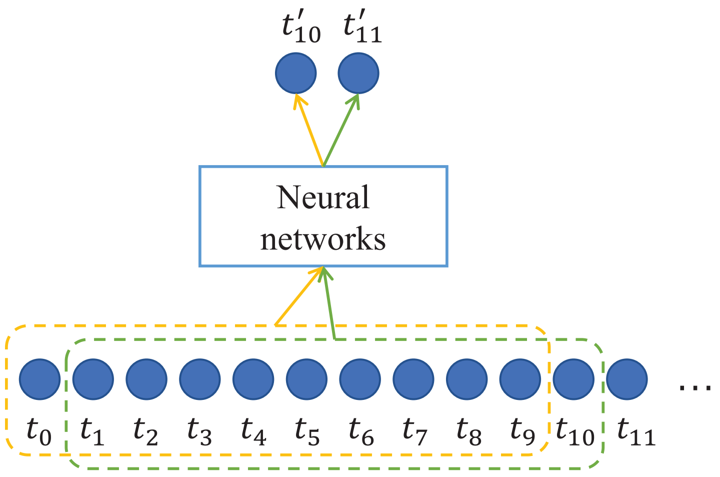

Figure 9. The predict mechanism of the neural network.

In the experiment, the following criteria were used to evaluate all the mentioned methods: the root mean square error (RMSE), mean absolute error (MAE) and mean absolute prercentage error (MAPE), which are widely used in STLF [1,7,17]. They are calculated as follows:

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \dot{x}_i)^2}, \tag{20}$$

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|x_i - \dot{x}_i|, \tag{21}$$

$$MAPE = \frac{1}{N}\sum_{i=1}^{N}\left|\frac{x_i - \dot{x}_i}{x_i}\right| \times 100\%, \tag{22}$$

where x_i is the actual load value, $\dot{x}_i$ is the predicted load value, N is the number of test samples. These criteria represent three types of deviation between the forecast and actual values: the smaller the criteria, the higher the forecasting accuracy. MAE is the basic metric for STLF, RMSE is sensitive to the regression point with large deviation, and MAPE considers both the error and the ratio which are between the predicted value and true value.

In order to balance performance and accuracy, we tried several trials to decide the size of all the neural networks in a 30-min time scale data set, the trails' results on 30-min test set are shown in Table 2. The number after the methods is the number of neurons. We have tried a three-layer BPNN architecture, and the results show it can forecast the load values well when the number of hidden neurons is 64. More or less hidden neurons will cause a decrease in performance. It means that a three-layer BPNN with 64 hidden neurons is a good choice for this load forecast task. Therefore, we have not added more layers into BPNN. The same as with BPNN, from the results, we can see that one recurrent layer with four neurons is a good choice for this load forecast task. The specific configuration of the mentioned neuron networks for STLF was as follows: The BPNN contained three layers. Specifically, the first layer was the input layer, which contained 10 neurons, and each input was a sampled load value; the second layer was a hidden layer whose size was 64; and the last layer was the output layer, which only contained one neuron, which represented the predicted value. The RNN, LSTM, and gate-RNN input layers contained one neuron, which was input into a 4-neuron RNN, LSTM, or gate-RNN cell, and the output was the same as that of the BPNN. The RNN cell calculated the value of the last 10 moments to predict the value of the next moment. All the networks in this paper were implemented using the Keras framework. The loss function of these methods was Equation (22). The optimizer of the neural networks was RMSProp and its parameters were set as Keras default parameters except for the learning rate. During the training process, the learning rate of epoch 1 to 5 was 0.01 and the remaining epoches learning rate was 5×10^{-5}. The total number of epoch was 55 and the value of weight decay is 5×10^{-5}.

Table 2. Results of different hidden layer sizes in 30-min time scale test set.

Number of Hidden Neurons	RMSE	MAPE	MAE
BP_32	585.22	2.38	439.96
BP_64	529.23	2.12	393.11
BP_128	581.59	2.21	408.03
GATE_2	524.75	2.02	374.16
GATE_4	530.09	2.00	372.40
GATE_8	520.05	1.83	337.12
LSTM_2	1323.30	5.58	1041.50
LSTM_4	1443.63	6.16	1161.02
LSTM_8	1800.25	8.17	1467.79
RNN_2	518.58	1.97	365.87
RNN_4	506.58	1.85	341.61
RNN_8	518.02	1.87	343.78

3.2. Comparison of the Models with the Baseline Method

To evaluate the effectiveness of the neural network methods mentioned in the section above, several most commonly used traditional methods were chosen as the baseline methods. They are SVR [22], decision tree (DT) [9], autoregressive integrated moving average model (ARIMA) [48], and one random forest (RF) proposed in [49]. SVR is one of the most popular methods that is widely used to forecast short-term load. DT, ARIMA and RF introduced in [49] are useful methods. All these methods have been used in STLF in different time scales, 5-min, 20-min, 30-min and 40-min time scale [9,48,49]. The results of all the methods test on the test set of three months are shown in Tables 3–6. From these tables, we can see that among the eight methods, the neural network methods achieved better comprehensive performance compared with other traditional methods in all different time scales, except LSTM. In a smaller time scale, specifically in 5-min and 20-min, gate-RNN achieved better performance than any other methods. ARIMA achieved the best performance among all the traditional methods. On a larger time scale, 30-min and 40-min, BP achieved better comprehensive performance than any other methods. Meanwhile, DT achieved the best performance among all traditional methods. It has shown that the performance of methods is influenced by the time scale.

Table 3. Results of eight methods on the 5-min time scale in a three months' test dataset.

Methods	RMSE	MAPE (%)	MAE
SVR [22]	913.61	4.61	766.82
DT [9]	145.26	0.63	133.68
ARIMA [48]	131.42	0.53	131.43
RF [49]	159.95	0.68	122.67
BP	137.69	0.59	108.62
RNN	130.69	0.55	100.12
LSTM	667.49	2.75	507.67
gate-RNN	116.94	0.49	89.36

Table 4. Results of eight methods on the 20-min time scale in three months' test dataset.

Methods	RMSE	MAPE (%)	MAE
SVR [22]	880.66	4.04	729.80
DT [9]	386.30	1.63	301.47
ARIMA [48]	402.06	1.47	269.82
RF [49]	310.2	1.69	427.38
BP	451.72	1.81	334.30
RNN	403.14	1.65	304.58
LSTM	828.09	3.37	626.32
gate-RNN	337.00	1.31	242.75

Table 5. Results of eight methods on the 30-min time scale in three months' test dataset.

Methods	RMSE	MAPE (%)	MAE
SVR [22]	830.00	3.80	682.88
DT [9]	517.70	2.14	394.55
ARIMA [48]	670.24	2.41	442.21
RF [49]	646.02	2.41	447.30
BP	529.23	2.12	393.11
RNN	506.58	1.85	341.61
LSTM	1443.63	6.16	1161.02
gate-RNN	530.09	2.00	372.40

Table 6. Results of eight methods on the 40-min time scale in three months' test dataset.

Methods	RMSE	MAPE (%)	MAE
SVR [22]	1155.91	5.65	979.10
DT [9]	701.78	3.02	553.725
ARIMA [48]	818.07	3.43	628.24
RF [49]	943.62	3.48	647.44
BP	715.68	2.85	527.99
RNN	762.60	2.91	537.40
LSTM	2056.35	8.49	1598.33
gate-RNN	792.15	3.23	601.51

To clearly illustrate the forecast value, from the test set of three months, we chose 1000 time scales' actual and predicted electric load values obtained by the eight methods to draw a figure on a 30-min time scale, which is shown in Figure 10. The orange line stands for real load values, the blue line stands for prediction values and the green line stands for the average value of real load values. Furthermore, 1000-time scales are nearly one month, from Figure 10, we can see the blue line is far away from the green line and close to the orange line. All of the methods' prediction values are much closer to the real values; this means that the prediction values have no relations to the average value for some months and even one year. Specifically, it can be seen that the predicted values obtained by the LSTM method have the maximum error for the original data compared with the other seven methods; particularly, the other three types of neural network methods. This is because the structure of LSTM is much more complicated than the other three types of neural networks. To achieve equivalent performance, LSTM needs much more data than BP, RNN, and gate-RNN. It is difficult to train an LSTM for STLF using only one year of load data. Additionally, we plot the error distribution of each model in 30-min time scale, which is shown in Figure 11. RNN and gate-RNN are more accurate and effective, while the errors of other methods are scattered in a wider distribution space.

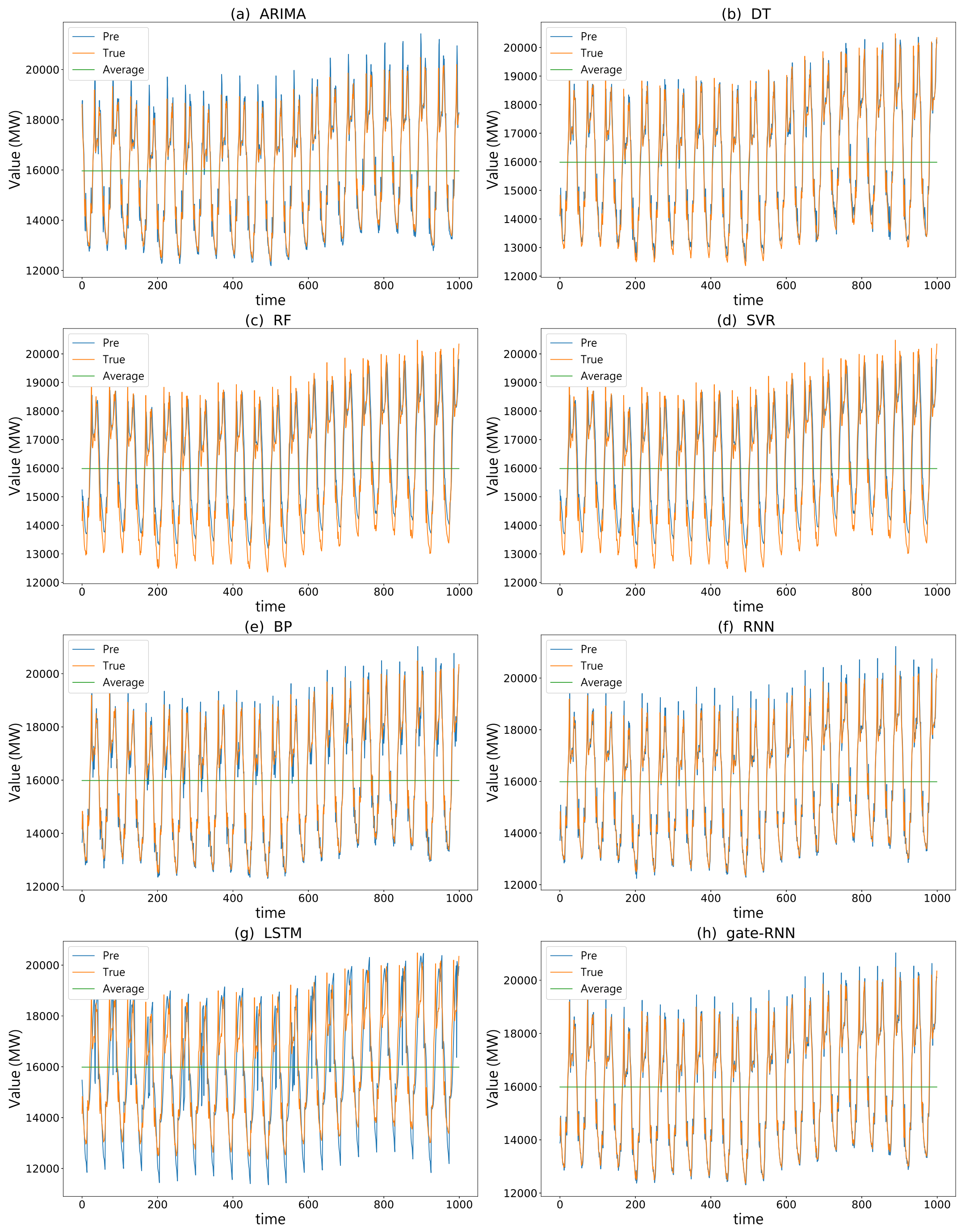

Figure 10. Visualization of the results of four types of neural networks and other four comparative approach in 30-min time scale: (**a**) results of ARIMA; (**b**) results of DT; (**c**) results of RF; (**d**) results of SVR; (**e**) results of BP; (**f**) results of RNN; (**g**) results of LSTM; (**h**) results of gate-RNN. The blue line indicates the outputs of the neural network and the orange line is the true value of the load.

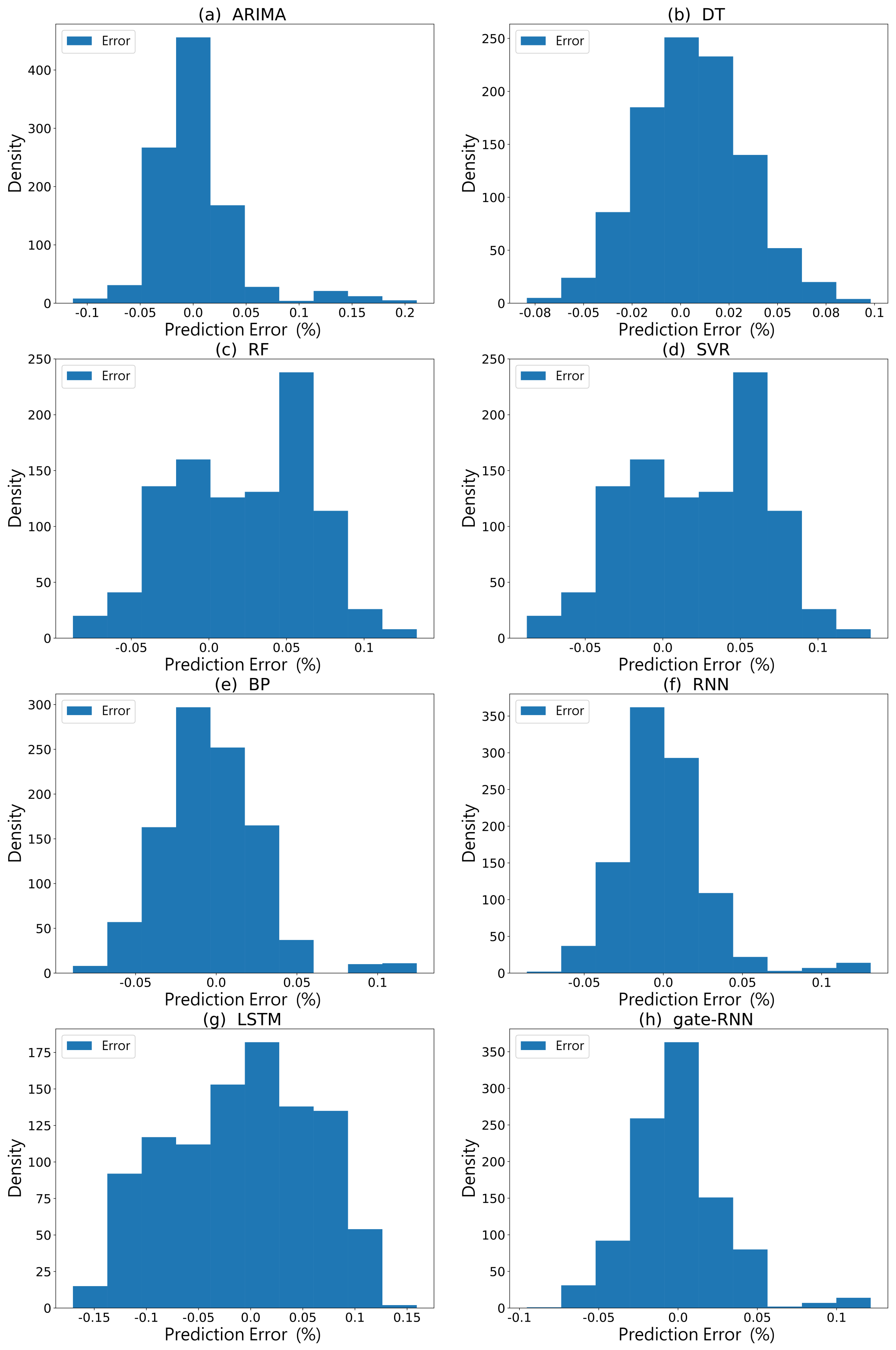

Figure 11. All eight methods' prediction error distributions in 30-min time scale. (**a**) distribution of ARIMA; (**b**) distribution of DT; (**c**) distribution of RF; (**d**) distribution of SVR; (**e**) distribution of BP; (**f**) distribution of RNN; (**g**) distribution of LSTM; (**h**) distribution of gate-RNN.

3.3. Performance Analysis of the Neural Networks

In this section, we analyze the performance of four neural network methods which include the training process of different neural networks and the performance of the neural networks. To better compare these methods, we used four time scales to process the data: 5 min, 20 min, 30 min, and 40 min.

3.3.1. Training Process Analysis for the Neural Networks

To assess whether the neural networks were trained and learned the distribution of data, a direct approach is to assess whether the train loss and test loss converged. From Figures 12–15, we can see that the RNN and gate-RNN train loss and test loss converged well for all time scales; LSTM had large fluctuations and BP had slight fluctuations. For the 5-min time scale, even though LSTM had a large fluctuation, all the methods achieved a small train loss and test loss, and the trends of the train loss and test loss were the same. This means that all the methods learned the distribution of train data and achieved good performance on the test set. When the time scale was larger than 5 min, even though the trends of the train loss and test loss were the same, the convergence values of the train loss and test loss were different, and the losses increased as the time scale increased. This means that, as the time scale increased, the performance of the neural network became worse.

From Figures 12–15, we also can find that our proposed method converged after training in approximately five epochs, which was faster than any of the other methods. BP converged in approximately ten epochs and RNN converged in approximately seven epochs. This means that our proposed method had significant power to forecast ultra short-term load. We believe that it benefits from the suitable gate mechanism. The RNN method has no gate mechanism; it cannot choose the memory information, but the LSTM method was too complicated to train on the data.

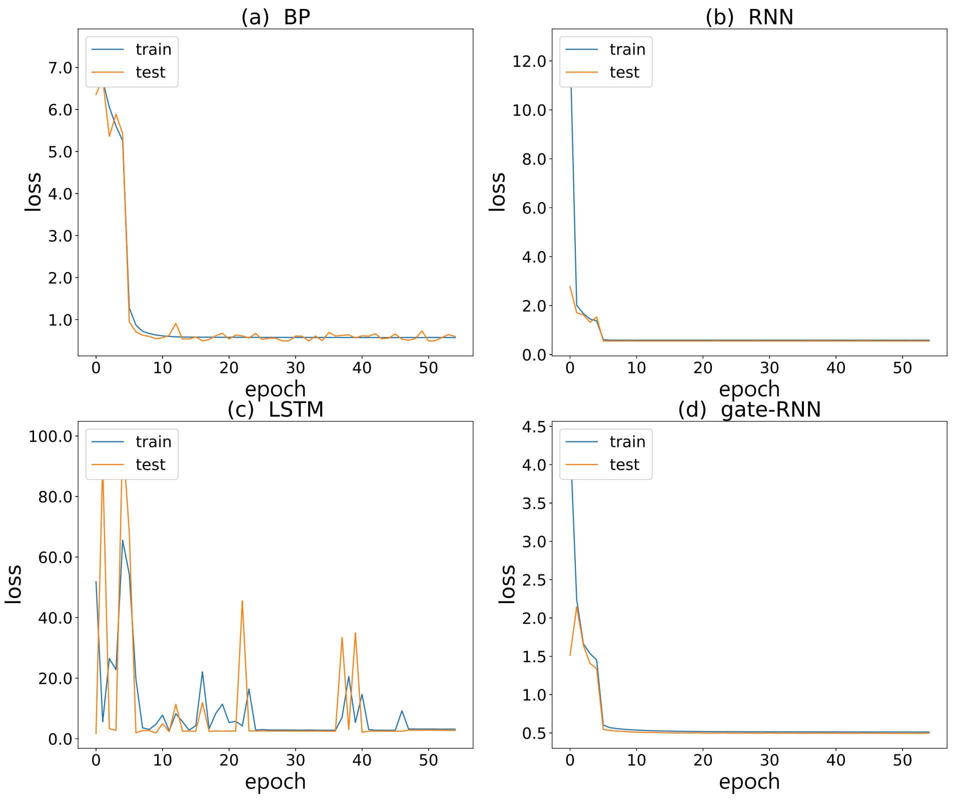

Figure 12. Neural networks' losses on the train and test datasets for 5-min time scales: (**a**) loss of BPNN; (**b**) loss of RNN; (**c**) loss of LSTM; and (**d**) loss of gate-RNN. The blue line is the train loss curve and the orange line is the test loss curve.

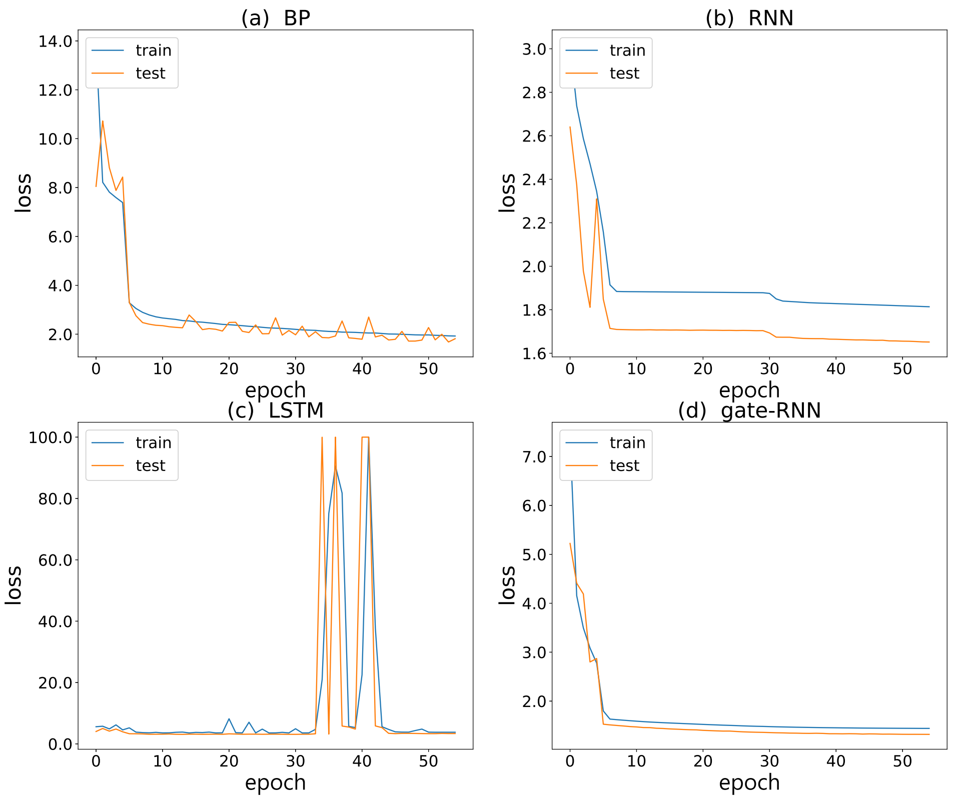

Figure 13. Neural networks' losses on the train and test datasets for 20-min time scales: (**a**) loss of BPNN; (**b**) loss of RNN; (**c**) loss of LSTM; and (**d**) loss of gate-RNN. The blue line is the train loss curve and the orange line is the test loss curve.

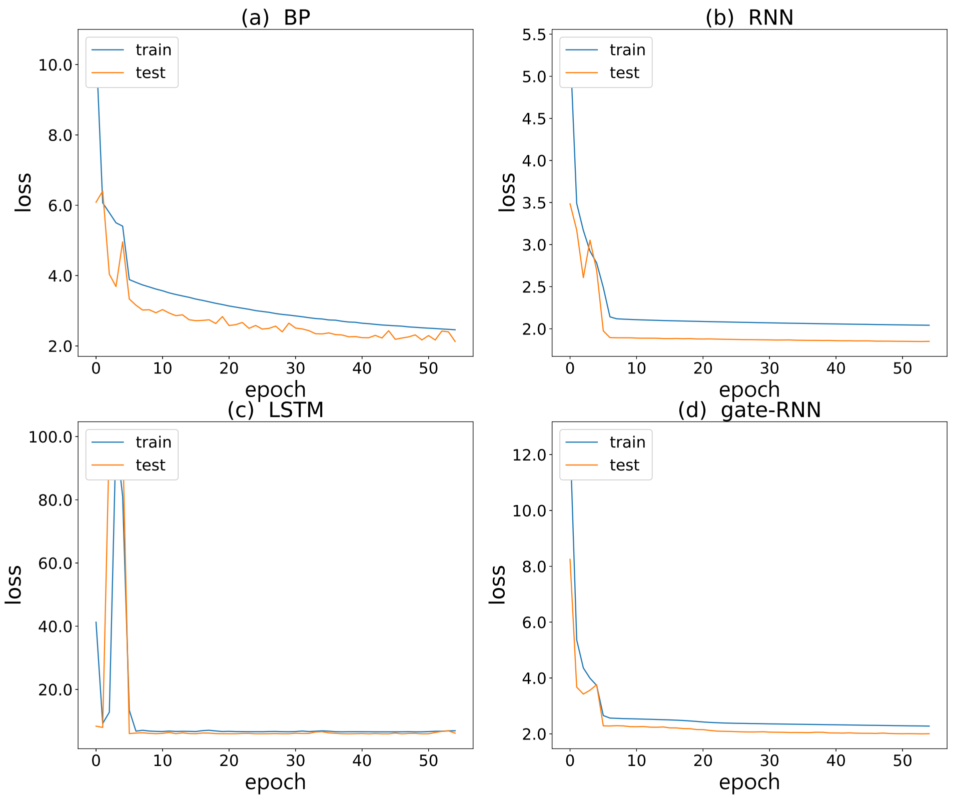

Figure 14. Neural networks' losses on the train and test datasets for 30-min time scales: (**a**) loss of BPNN; (**b**) loss of RNN; (**c**) loss of LSTM; and (**d**) loss of gate-RNN. The blue line is the train loss curve and the orange line is the test loss curve.

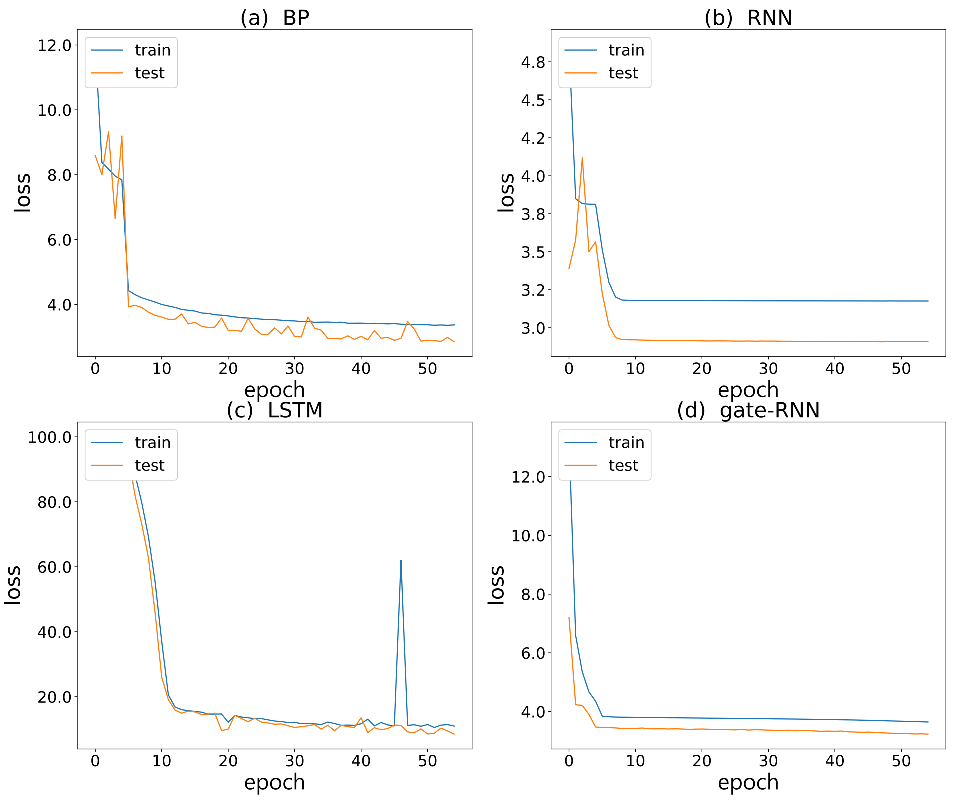

Figure 15. Neural networks' losses on the train and test datasets for 40-min time scales: (**a**) loss of BPNN; (**b**) loss of RNN; (**c**) loss of LSTM; and (**d**) loss of gate-RNN. The blue line is the train loss curve and the orange line is the test loss curve.

3.3.2. Examination of All the Neural Networks

After analyzing the train loss and test loss, we analyzed the criteria for each method for different time scales. The results for the four neural networks are shown in Tables 7–9. Tables 7–9 which present the RMSE, MAPE, and MAE, respectively, for four time scales. Table 7 shows that gate-RNN achieved the best performance for the 5-min and 20-min time scales; RNN achieved the best performance for the 30-min time scale; and BP achieved the best performance for the 40-min time scale. The results in Tables 8 and 9 show the same performance as Table 7 where gate-RNN achieved the best performance for the 5-min and 20-min time scales; RNN achieved the best performance for the 30-min time scale; and BP achieved the best performance for the 40-min time scale. Overall, gate-RNN achieved good results among all the four methods. This proves that our proposed method is better at forecasting ultra short-term load again.

Table 7. Results of the RMSE for different time scales.

Methods	5 min	20 min	30 min	40 min
BP	137.69	451.72	529.23	715.68
RNN	130.69	403.14	506.58	762.60
LSTM	667.49	828.09	1443.63	2056.35
gate-RNN	116.94	337.00	530.09	792.15

Table 8. Results of the MAPE for different time scales.

Methods	5 min	20 min	30 min	40 min
BP	0.59	1.81	2.12	2.85
RNN	0.55	1.65	1.85	2.91
LSTM	2.75	3.37	6.16	8.49
gate-RNN	0.49	1.31	2.00	3.23

Table 9. Results of the MAE for different time scales.

Methods	**5 min**	**20 min**	**30 min**	**40 min**
BP	108.62	334.3	393.11	527.99
RNN	100.12	304.58	341.61	537.40
LSTM	507.67	626.32	1161.02	1598.33
gate-RNN	89.36	242.75	372.40	601.51

Further visualization results are shown in Figure 16. We can see it intuitively that the performance of all the methods became worse as the time scale increased. Overall, all the methods had a similar trend: they began at a relatively small value and ended with a large value. Figure 16 also shows that our method achieved the best results for the criteria for small time scales, but, as the time scales increased, our method's performance became slightly worse. Overall, the performance of LSTM for all criteria for different time scales was worse, and the values for the criteria were far from those of the other neural network methods.

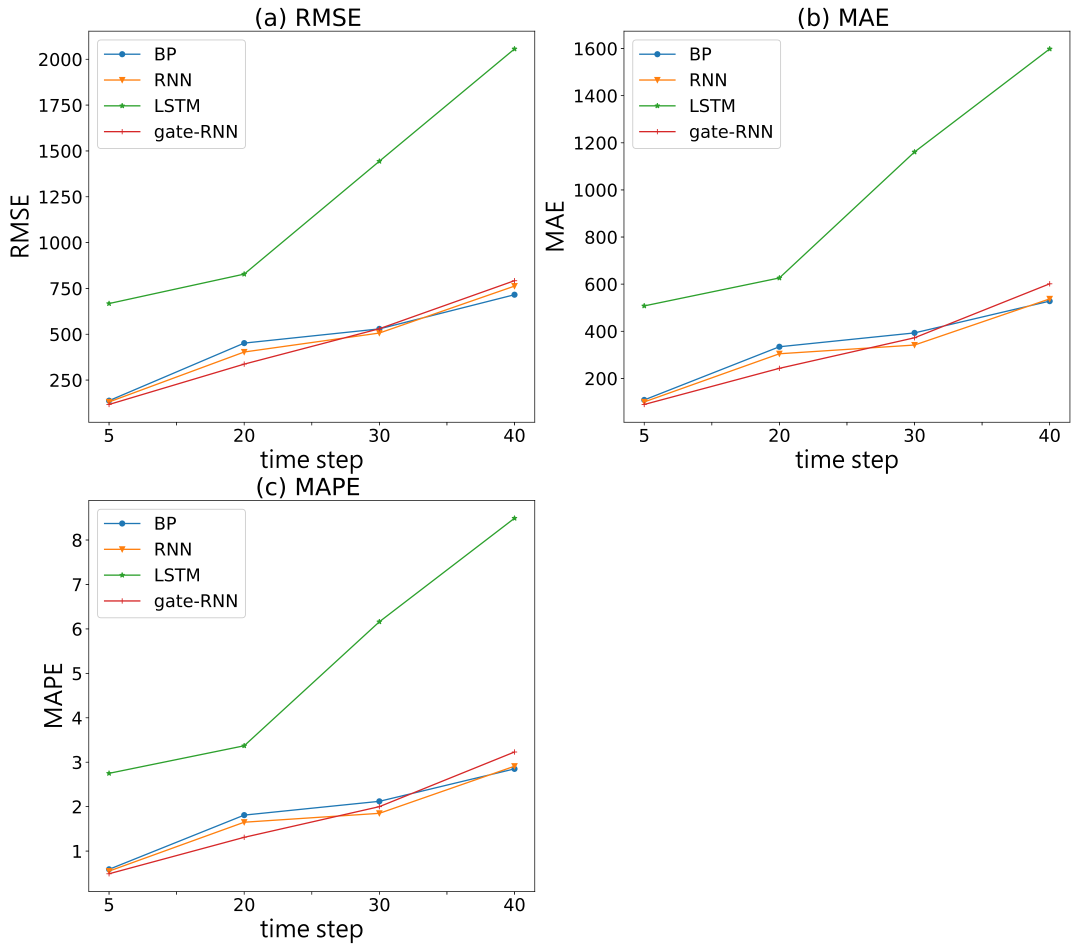

Figure 16. Visualization of the three criteria for the neural networks for different time scales: (**a**) RMSE of the neural networks for different time scales; (**b**) MAE of the neural networks for different time scales; and (**c**) MAPE of the neural networks for different time scales.

We believe that this scenario was caused by the following main points. First, we consider the reason that the data is time scale. The larger the time scale, the more likely that unpredictable states will affect the load value, such as sudden weather changes and holidays. Even though the neural networks could predict the trend of the load, the values of the load would have a large turbulence. Second, we consider the neural networks' features. As we know, the complexity of the structures increases gradually in the following order: BP, RNN, gate-RNN, and LSTM. The more complex the architecture of a neural network, the more difficult it is to train it, and the more data are needed to train it. After we process the data into different time scales, the data size changes according to the time scale. For example, the number of total data items is T^{total} recorded every 1 min, so, when our time scale is 5 min, the data items become T^{5min}, which is $\frac{1}{5}$ of the total items, that is, five times smaller than T^{total}. All of the time scales can be represented by $T^{total} = 5 \times T^{5min} = 20 \times T^{20min} = 30 \times T^{30min} = 40 \times T^{40min}$. Thus, when using these different time scales to process data to train the networks, the actual data size decreased with the time scale, and then the performance became worse with an increase in the time scale.

4. Conclusions

The aim of our work was to explore the performance of different types of neural networks in the STLF, and, based on the insufficiency of current neural networks, we proposed a new neural network architecture called the gate-RNN. Our proposed method is extremely suitable for handling ultra STLF. From the experimental results, we found that neural networks have the STLF ability. They can achieve better performance than the tranditional methods. However, the more complex the neural network's architecture, the more data are needed to train the neural network. Considering that different neural networks have different characteristics, for our dataset, the gate-RNN achieved the highest score overall. For a large time scale, the BP achieved better performance than RNNs.

In future research, we plan to improve our work in two aspects. First, we will extend our data to multivariate data. Collecting more electrical load data and adding more relevant factors can influence load forecasting, for example, weather and some breaking news information. Second, based on the gate-RNN, to train the neural network more efficiently, we plan to design a loss function that can reflect the forecasting value online.

Author Contributions: Data curation, L.Y.; Formal analysis, L.Y.; Funding acquisition, H.Y.; Methodology, L.Y.; Supervision, H.Y.; Writing—original draft, L.Y.; Writing—review and editing, L.Y.

Funding: This research was funded by the National Natural Science Foundation of China Grant No. 51477105.

Acknowledgments: We thank those who provided constructive comments and feedback to help improve this paper.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

STLF	Short-term load forecasting
ANN	artificial neural network
SVM	Support vector machine
SVR	Support vector regression
BPNN	Back propagation neural network
RNN	Recurrent neural network
LSTM	Long-short-term memory
gate-RNN	Gate-recurrent neural network
MAE	Mean absolute error
RMSE	Root mean square error
MAPE	Mean absolute percentage error
DT	Decision tree
ARIMA	Autoregressive integrated moving average model
RF	Random forest

References

1. Bunn, D.W. Forecasting loads and prices in competitive power markets. *Proc. IEEE* **2000**, *88*, 163–169. [CrossRef]
2. Senjyu, T.; Mandal, P.; Uezato, K.; Funabashi, T. Next Day Load Curve Forecasting Using Hybrid Correction Method. *IEEE Trans. Power Syst.* **2005**, *20*, 102–109. [CrossRef]
3. Nataraja, C.; Gorawar, M.; Shilpa, G.; Harsha, J.S. Short term load forecasting using time series analysis: A case study for Karnataka, India. *Int. J. Eng. Sci. Innov. Technol.* **2012**, *1*, 45–53.
4. Goia, A.; May, C.; Fusai, G. Functional clustering and linear regression for peak load forecasting. *Int. J. Forecast.* **2010**, *26*, 700–711. [CrossRef]
5. Trudnowski, D.J.; Mcreynolds, W.L.; Johnson, J.M. Real-time very short-term load prediction for power-system automatic generation control. *IEEE Trans. Control Syst. Technol.* **2001**, *9*, 254–260. [CrossRef]
6. Wang, H.; Schulz, N.N. Using AMR data for load estimation for distribution system analysis. *Electr. Power Syst. Res.* **2006**, *76*, 336–342. [CrossRef]
7. Jiang, H.; Zhang, Y.; Muljadi, E.; Zhang, J.J.; Gao, D.W. A Short-Term and High-Resolution Distribution System Load Forecasting Approach Using Support Vector Regression with Hybrid Parameters Optimization. *IEEE Trans. Smart Grid* **2017**, *9*, 3341–3350. [CrossRef]
8. Capizzi, G.; Sciuto, G.L.; Napoli, C.; Tramontana, E. Advanced and adaptive dispatch for smart grids by means of predictive models. *IEEE Trans. Smart Grid* **2018**, *9*, 6684–6691. [CrossRef]
9. Hambali, A.O.J.; Akinyemi, M.; JYusuf, N. Electric Power Load Forecast Using Decision Tree Algorithms. *Comput. Inf. Syst. Dev. Inform. Allied Res. J.* **2016**, *4*, 29–42.
10. Bonanno, F.; Capizzi, G.; Sciuto, G.L. A neuro wavelet-based approach for short-term load forecasting in integrated generation systems. In Proceedings of the 2013 International Conference on Clean Electrical Power (ICCEP), Alghero, Italy, 11–13 June 2013; pp. 772–776.
11. Tian, C.; Ma, J.; Zhang, C.; Zhan, P. A Deep Neural Network Model for Short-Term Load Forecast Based on Long Short-Term Memory Network and Convolutional Neural Network. *Energies* **2018**, *11*, 3493. [CrossRef]
12. Tian, C.; Hao, Y. A novel nonlinear combined forecasting system for short-term load forecasting. *Energies* **2018**, *11*, 712. [CrossRef]
13. Zhang, X. Short-term load forecasting for electric bus charging stations based on fuzzy clustering and least squares support vector machine optimized by wolf pack algorithm. *Energies* **2018**, *11*, 1449. [CrossRef]
14. Merkel, G.; Povinelli, R.; Brown, R. Short-term load forecasting of natural gas with deep neural network regression. *Energies* **2018**, *11*, 2008. [CrossRef]
15. Li, G.; Li, B.-J.; Yu, X.-G.; Cheng, C.-T. Echo state network with bayesian regularization for forecasting short-term power production of small hydropower plants. *Energies* **2015**, *11*, 12228–12241. [CrossRef]
16. Hu, C.; Luo, S.; Li, Z.; Wang, X.; Sun, L. Energy coordinative optimization of wind-storage-load microgrids based on short-term prediction. *Energies* **2015**, *8*, 1505–1528. [CrossRef]
17. Fan, S.; Hyndman, R.J. Short-term load forecasting based on a semi-parametric additive model. *Monash Econom. Bus. Stat. Work. Pap.* **2010**, *27*, 134–141. [CrossRef]
18. Taylor, J.W. Short-term load forecasting with exponentially weighted methods. *IEEE Trans. Power Syst.* **2012**, *27*, 458–464. [CrossRef]
19. Dudek, G. Pattern-based local linear regression models for short-term load forecasting. *Electr. Power Syst. Res.* **2016**, *130*, 139–147. [CrossRef]
20. Niu, D.X.; Shi, H.F.; Wu, D.D. Short-term load forecasting using bayesian neural networks learned by Hybrid Monte Carlo algorithm. *Appl. Soft Comput.* **2012**, *12*, 1822–1827. [CrossRef]
21. Song, L.; Peng, W.; Goel, L. Short-term load forecasting by wavelet transform and evolutionary extreme learning machine. *Electr. Power Syst. Res.* **2015**, *122*, 96–103.
22. Ceperic, E.; Ceperic, V.; Baric, A. A strategy for short-term load forecasting by support vector regression machines. *IEEE Trans. Power Syst.* **2013**, *28*, 4356–4364. [CrossRef]
23. Min, Z.; Tao, H. Short Term Load Forecasting with Least Square Support Vector Regression and PSO. *Commun. Comput. Inf. Sci.* **2011**, *228*, 124–132.
24. Li, Y.C.; Chen, P. A Parallel SVR Model for Short Term Load Forecasting Based on Windows Azure Platform. In Proceedings of the Power & Energy Engineering Conference, Shanghai, China, 27–29 March 2012.

25. Chen, Y.; Peng, X.; Chu, Y.; Li, W.; Wu, Y.; Ni, L.; Yi, B.; Wang, K. Short-term electrical load forecasting using the Support Vector Regression (SVR) model to calculate the demand response baseline for office buildings. *Appl. Energy* **2017**, *195*, 659–670. [CrossRef]
26. Fan, G.F.; Peng, L.L.; Hong, W.C.; Fan, S. Electric load forecasting by the SVR model with differential empirical mode decomposition and auto regression. *Neurocomputing* **2016**, *173*, 958–970. [CrossRef]
27. Fan, X.; Zhu, Y. The application of Empirical Mode Decomposition and Gene Expression Programming to short-term load forecasting. In Proceedings of the Sixth International Conference on Natural Computation, Yantai, China, 10–12 August 2010.
28. Ghelardoni, L.; Ghio, A.; Anguita, D. Energy Load Forecasting Using Empirical Mode Decomposition and Support Vector Regression. *IEEE Trans. Smart Grid* **2013**, *4*, 549–556. [CrossRef]
29. Hong, W.-C.; Dong, Y.; Zhang, W.Y.; Chen, L.-Y.; Panigrahi, B.K. Cyclic electric load forecasting by seasonal SVR with chaotic genetic;algorithm. *Int. J. Electr. Power Energy Syst.* **2013**, *44*, 604–614. [CrossRef]
30. Pai, P.F.; Hong, W.C. Support vector machines with simulated annealing algorithms in electricity load forecasting. *Energy Convers. Manag.* **2005**, *46*, 2669–2688. [CrossRef]
31. Niu, D.; Wang, Y.; Wu, D.D. Power load forecasting using support vector machine and ant colony optimization. *Expert Syst. Appl.* **2010**, *37*, 2531–2539. [CrossRef]
32. Taylor, J.W. Short-term electricity demand forecasting using double seasonal exponential smoothing. *J. Oper. Res. Soc.* **2003**, *54*, 799–805. [CrossRef]
33. Çevik, H.H.; Çunkaş, M. Short-term load forecasting using fuzzy logic and ANFIS. *Neural Comput. Appl.* **2015**, *26*, 1355–1367. [CrossRef]
34. Xiao, Z.; Ye, S.J.; Zhong, B.; Sun, C.X. BP neural network with rough set for short term load forecasting. *Expert Syst. Appl. Int. J.* **2009**, *36*, 273–279. [CrossRef]
35. Yuan, N.; Liu, Y.; Qiang, J. Bayesian—BP Neural Network based Short-Term Load Forecasting for power system. In Proceedings of the International Conference on Advanced Computer Theory & Engineering, Chengdu, China, 20–22 August 2010.
36. Hou, B.; Zu, Y.X.; Zhang, C. A Forecasting Method of Short-Term Electric Power Load Based on BP Neural Network. *Appl. Mech. Mater.* **2014**, *538*, 247–250.
37. Vermaak, J.; Botha, E.C. Recurrent neural networks for short-term load forecasting. *IEEE Trans. Power Syst.* **2002**, *13*, 126–132. [CrossRef]
38. Khan, G.M.; Zafari, F.; Mahmud, S.A. Very Short Term Load Forecasting Using Cartesian Genetic Programming Evolved Recurrent Neural Networks (CGPRNN). In Proceedings of the International Conference on Machine Learning & Applications, Detroit, MI, USA, 3–5 December 2014.
39. Tokgóz, A.; Únal, G. A RNN based time series approach for forecasting turkish electricity load. In Proceedings of the 2018 26th Signal Processing and Communications Applications Conference (SIU), Izmir, Turkey, 2–5 May 2018; pp. 1–4.
40. Shi, H.; Xu, M.; Li, R. Deep learning for household load forecasting-A novel pooling deep RNN. *IEEE Trans. Smart Grid* **2018**, *9*, 5271–5280. [CrossRef]
41. Park, D.C.; Tran, C.N.; Lee, Y. Short-Term Load Forecasting Using Multiscale BiLinear Recurrent Neural Network. *Lect. Notes Comput. Sci.* **2006**, *4099*, 497–506.
42. Li, T.; Wang, B.; Zhou, M.; Watada, J. Short-term load forecasting using optimized LSTM networks based on EMD. *arXiv* **2018**, arXiv:1809.10108.
43. Kong, W.; Dong, Z.Y.; Jia, Y.; Hill, D.J.; Xu, Y.; Zhang, Y. Short-term residential load forecasting based on LSTM recurrent neural network. *IEEE T. Smart Grid.* **2017**, *10*, 1. [CrossRef]
44. Kumar, J.; Goomer, R.; Singh, A.K. Long Short Term Memory Recurrent Neural Network (LSTM-RNN) Based Workload Forecasting Model For Cloud Datacenters. *Procedia Comput. Sci.* **2018**, *125*, 676–682. [CrossRef]
45. Srivastava, R.K.; Greff, K.; Schmidhuber, J. Highway networks. *arXiv* **2015**, arXiv:1505.00387.
46. Williams, R.J.; Zipser, D. Gradient-based learning algorithms for recurrent networks and their computational complexity. *Backpropag. Theory Archit. Appl.* **1995**, *1*, 433–486.
47. Hochreiter, S.; Schmidhuber, J. Long Short-Term Memory. *Neural Comput.* **1997**, *9*, 1735–1780.

[CrossRef]

48. Al-Musaylh, M.S.; Deo, R.C.; Adamowski, J.F.; Li, Y. Short-term electricity demand forecasting with MARS, SVR and ARIMA models using aggregated demand data in Queensland, Australia. *Adv. Eng. Inform.* **2018**, *35*, 1–16. [CrossRef]
49. Huo, J.; Shi, T.; Jing, C. Comparison of Random Forest and SVM for electrical short-term load forecast with different data sources. In Proceedings of the IEEE International Conference on Software Engineering and Service Science, Beijing, China, 20–22 November 2017.