Online Recurrent Extreme Learning Machine and its Application to Time-series Prediction

Jin-Man Park
Robotics Program
KAIST
Daejeon 305-701, Republic of Korea
Email: jmpark@rit.kaist.ac.kr

Jong-Hwan Kim
School of Electrical Engineering
KAIST
Daejeon 305-701, Republic of Korea
Email: johkim@rit.kaist.ac.kr

Abstract—Online sequential extreme learning machine (OS-ELM) is an online learning algorithm training single-hidden layer feedforward neural networks (SLFNs), which can learn data one-by-one or chunk-by-chunk with fixed or varying data size. Due to its characteristics of online sequential learning, OS-ELM is popularly used to solve time-series prediction problem, such as stock forecast, weather forecast, passenger count forecast, etc. OS-ELM, however, has two fatal drawbacks: Its input weights cannot be adjusted and it cannot be applied to learn recurrent neural network (RNN). Therefore we propose a modified version of OS-ELM, called online recurrent extreme learning machine (OR-ELM), which is able to adjust input weights and can be applied to learn RNN, by applying ELM-auto-encoder and a normalization method called layer normalization (LN). Proposed method is used to solve a time-series prediction problem on New-York City passenger count dataset, and the results show that R-ELM outperforms OS-ELM and other online-sequential learning algorithms such as hierarchical temporal memory (HTM) and online long short-term memory (online LSTM).

Index Terms—Online sequential extreme learning machine (OS-ELM), online recurrent extreme learning machine (OR-ELM), online learning, time-series prediction, auto-encoding.

I. INTRODUCTION

An online learning algorithm is a method of machine learning, which observes a stream of examples and make a prediction for each element in the stream. The algorithm receives immediate feedback about each prediction and uses this feedback to improve its accuracy on subsequent predictions. Therefore the algorithm is used to solve time-series prediction problems, where an algorithm should be able to dynamically adapt to new patterns in the data, e.g. predicting stock market trends, weather forecasting, traffic forecasting, etc.

Online Sequential Extreme Learning Machine (OS-ELM) [1], [2], one of the online learning algorithms, is a known to be a good solution to time-series prediction. OS-ELM is a sequential variant of extreme learning machine (ELM), which trains single-hidden layer feedforward neural networks (SLFNs) in an online manner. OS-ELM enables the network to learn data one-by-one or chunk-by-chunk by randomly initializing input weights and updating output weights using recursive least square (RLS) method. By applying forgetting factor of RLS, OS-ELM can quickly adapt to new input patterns; therefore it shows better prediction performance than other online learning algorithms.

OS-ELM, however, has some drawbacks; Updating input weights of OS-ELM using auto-encoder based feature extraction techniques results in worse performance than that of randomly initialized weights, while adjusting input weights of ELM using ELM-auto-encoder (ELM-AE) can further increase the performance of ELM[3], [4]. Another drawback of OS-ELM is that applying OS-ELM to train recurrent neural network results in lower performance than the performance of the SLFN of the same size.

We argue that the reason of these two degradations of the performance can be explained by the internal covariate shift phenomenon, defined by [5]. According to the phenomenon, the update of weight of a certain layer shifts its output feature distributions, which interferes with learning the weights of adjacent higher layer. In case of OS-ELM, we suspect that the update of input weight shifts hidden feature layer distributions, which results in lower performance. The authors in [5] solved the problem by adding batch normalization layers just before the non-linear activation layers. In case of online learning, however, batch normalization is impossible because the mini-batch size is 1. Other normalization method is needed to solve internal covariate shift problem.

Therefore, we propose a new variant of OS-ELM, called online recurrent extreme learning machine (OR-ELM), which shows enhanced prediction performance by updating input weight, and which is suitable for learning recurrent neural network. The proposed OR-ELM successfully solved the problem of internal covariate shift, by adding whitening layer, or called layer normalization (LN), just before the non-linear activation layer. In LN layer of OR-ELM, the input features are simply normalized by subtracting the mean of the features and then by dividing the variance of the features. In OR-ELM, the input weight is updated whenever new input is arrived, using the ELM-based auto-encoding technique proposed in [3]. Furthermore, the proposed online learning method is applied to train recurrent neural network. the hidden weight connecting hidden layers is also updated whenever new input is arrived, using the same technique used to update the input weight.

We applied OR-ELM to time-series prediction with the New-York city (NYC) taxi passenger count dataset, where the number of NYC taxi passenger is counted every 30 min window for approximately 2 years. The predicted output is
compared to the true data and the prediction accuracy is calculated using two kinds of error metric: normalized root mean square error (NRMSE) and mean absolute percentage error (MAPE). Then the accuracy of R-ELM is compared to that of OS-ELM and other online learning algorithms or other sequential learning algorithms, such as hierarchical temporal memory (HTM) and long short-term memory (LSTM). The result shows that the prediction accuracy of R-ELM is far better than OS-ELM, HTM and LSTM.

The rest of this paper is organized as follows: Section II gives a brief review of the basic concepts and related works of OS-ELM. A new online learning algorithm, called OR-ELM, is proposed in III, by applying ELM-AE with LN layer, and by applying recurrent neural network architecture. Performance evaluation of OR-ELM on its accuracy in time-series prediction is given in Section IV. Conclusions based on the study are highlighted in Section V.

II. BACKGROUNDS

This section briefly reviews the basic concepts and related works of ELM, ELM-AE, and OS-ELM to provide the necessary backgrounds for the development of OR-ELM in Section III.

A. Extreme Learning Machine

ELM is a learning algorithm training an SLFN, which consists of three layers: input layer, hidden layer, and output layer as shown in Fig. 1. The input weights and the corresponding bias values are randomly generated and fixed throughout the learning process. The output weights are analytically computed using pseudo-inverse.

For \( N \) training samples \( (x_j, t_j) \) where \( x_j \in \mathbb{R}^n \) and \( t_j \in \mathbb{R}^m \), the output of an ELM with \( L \) hidden nodes can be represented by

\[
\sum_{i=1}^{L} \beta_i (a_i \cdot x_j + b_i) = t_j, \quad j = 1, 2, \cdots, N \tag{1}
\]

where \( \beta_i \in \mathbb{R}^m \) is the output weight, \( a_i \in \mathbb{R}^n \) and \( b_i \in \mathbb{R} \) are respectively input weights and bias values of hidden nodes, and \( g : \mathbb{R} \to \mathbb{R} \) denotes a non-linear activation function and \( a_i \cdot x \) is the inner product of vectors \( a_i \) and \( x \) in \( \mathbb{R}^n \). Eq. 1 can be further expressed compactly as

\[
\mathbf{H}\beta = \mathbf{T}, \tag{2}
\]

where

\[
\mathbf{H} = \begin{bmatrix}
g(a_1 \cdot x_1 + b_1) & \cdots & g(a_L \cdot x_1 + b_L) \\
\vdots & \ddots & \vdots \\
g(a_1 \cdot x_N + b_1) & \cdots & g(a_L \cdot x_N + b_L)
\end{bmatrix}_{N \times L}, \tag{3}
\]

\[
\beta = \begin{bmatrix}
\beta^T_1 \\
\vdots \\
\beta^T_L
\end{bmatrix}_{L \times m}
\]

\[
\mathbf{T} = \begin{bmatrix}
t_1 \\
\vdots \\
t_L
\end{bmatrix}_{N \times m}
\]

Named by Huang et al.[6], \( \mathbf{H} \) is called the hidden-layer output matrix of the SLFN or ELM. In the basic ELM proposed in [6], \( a_i \) and \( b_i \) is randomly assigned and maintained during the whole learning process. As a result, Eq.2 changes into a linear system and the output weight \( \beta \) can be calculated as follows:

\[
\hat{\beta} = \hat{\mathbf{H}}^\dagger \mathbf{T}, \quad \hat{\mathbf{H}}^\dagger = \left( \mathbf{H}^T \mathbf{H} + \frac{1}{C} \right)^{-1} \mathbf{H}^T \tag{4}
\]

where \( \mathbf{H}^\dagger \) is the pseudo-inverse of the hidden-layer output matrix \( \mathbf{H} \), and \( C \) is a regularization constant, added to prevent \( \mathbf{H}^T \mathbf{H} \) from being a singular matrix and to improve the stability of the ELM.

B. ELM-auto-encoder

Many studies have shown that the ELM achieves better performance if well extracted hidden layer features are given [3],[4],[7]. ELM-AE[3] is proposed by L. Kasun et al. to extract better hidden layer features rather than the random features of basic ELM. The main objective of ELM-AE is to convert the input features in compressed or sparse representations, by adjusting the number of the nodes of the hidden layer. In order to perform unsupervised learning, the ELM is modified as follows:

- The input data is used as the target data \( t = x \).
- ELM-AE’s input weight \( \hat{\mathbf{a}} \in \mathbb{R}^{n \times L} \) and bias values of its hidden layer \( \hat{\mathbf{b}} \in \mathbb{R}^{1 \times 1} \) are randomly assigned, then orthogonalized:

\[
\hat{\mathbf{a}}^T \hat{\mathbf{a}} = \mathbf{I}, \quad \hat{\mathbf{b}}^T \hat{\mathbf{b}} = 1, \tag{5}
\]

According to [3], orthogonalization of these hidden layer parameters tends to enhance the generalization performance of ELM-AE.

Then the output weight of ELM-AE \( \hat{\beta} \) is calculated as the same way of basic ELM:

\[
\hat{\beta} = \hat{\mathbf{H}}^\dagger \mathbf{T}, \quad \hat{\mathbf{H}}^\dagger = \left( \hat{\mathbf{H}}^T \hat{\mathbf{H}} + \frac{1}{C} \right)^{-1} \hat{\mathbf{H}}^T \tag{6}
\]

where \( \hat{\mathbf{H}}^\dagger \) is the pseudo-inverse of the ELM-AE’s hidden-layer output matrix \( \hat{\mathbf{H}} \), and \( C \) is the regularization constant of the ELM-AE. As a result, \( \beta \) learns the transformation from hidden feature space to input data. In other words, \( \hat{\beta}^T \) is responsible for the transformation from the input data to the hidden feature space. Therefore we can use \( \hat{\beta}^T \) as the input weight of the ELM, to extract better hidden layer features rather than random features as follows:

\[
a = \hat{\beta}^T, \quad b = 0. \tag{7}
\]
OS-ELM consists of two phases: initialization phases and sequential learning phase.

1) Initialization phase: In initialization, basic ELM is used to train a SLFN with a small chunk of initial training data. For example, the output weight $\beta_0$ for an initial training dataset with $N_0$ training samples is obtained as

$$\beta_0 = P_0H_0T_0, \quad P_0 = \left(H_0^TH_0 + \frac{I}{C}\right)^{-1}.$$  \hfill (8)

Note that this initialization with the initial training dataset is off-line learning. In order to overcome this problem, Wong, et al.[8] proposed a fully online initialization method, named fully online sequential extreme learning machine (FOS-ELM), where no initial dataset is demanded as below:

$$\beta_0 = 0 \quad P_0 = \left(\frac{I}{C}\right)^{-1}$$  \hfill (9)

We adopted this fully online initialization method to the proposed OR-ELM algorithm due to its fully online characteristics and its compact implementation.

2) Online sequential learning phase: In the online sequential learning phase, the output weights are updated whenever a new chunk of input data with $N_{k+1}$ training samples arrives:

$$\beta_{k+1} = \beta_k + P_{k+1}H_{k+1}^T(T_{k+1} - H_{k+1}\beta_k),$$  \hfill (10)

$$P_{k+1} = P_k - P_kH_{k+1}(1 + H_{k+1}P_kH_{k+1}^T)^{-1}H_{k+1}P_k,$$  \hfill (11)

where $k+1$ indicates the $(k+1)$th chunk of input data with $k$ increasing from zero, and $H_{k+1}$ represents the hidden layer output for the $(k+1)$th chunk of input data.

The short-term prediction performance of the OS-ELM can be further increased with the use of forgetting factor, originally proposed in RLS[11] and applied to OS-ELM in [12]. With a constant forgetting factor $\lambda \in (0, 1]$, Eq.11 is modified as follows:

$$P_{k+1} = \frac{1}{\lambda}P_k - P_kH_{k+1}(\lambda^2 + \lambda H_{k+1}P_kH_{k+1}^T)^{-1}H_{k+1}P_k,$$  \hfill (12)

Note that if $\lambda = 1$, Eq. 12 is the same as Eq. 11, which means the OS-ELM does not forget anything. The forgetting factor $\lambda$ enables OS-ELM to continually forget the outdated input data in process of learning, to reduce their bad affection to the following learning.

III. ONLINE RECURRENT EXTREME LEARNING MACHINE

In this chapter, The degradation problem of OS-ELM, which occurs when its input weight is updated, is covered. Then an improved OS-ELM, namely, online recurrent extreme learning machine (OR-ELM) is proposed.

A. Degradation Problem of OS-ELM

The problem of OS-ELM is that if the input weight is updated using any possible methods, such as auto-encoding technique, the performance of OS-ELM decreases as shown in Fig. 3. In Fig. 3, the OS-ELM is used to predict a five-step future value of a time-series sequence, given a sequence of past values. The time-series dataset used in this experiment is New-York City (NYC) taxi passenger count, provided by the New York City Taxi and Parade Commission.
The authors in [5] solved the corresponding layer output, which interferes with learning the weight change leads to the shift of the distribution of its similar solution can be applied. According to [5], the internal covariate shift phenomenon, proposed in [5]; therefore the learning of output weight is disturbed and the performance of OS-ELM gets worse. In case of the RNN trained using OS-ELM, it is sure that the recurrent input of the RNN caused the shift of the distribution of the hidden layer output.

Therefore, we claim the normalization method should be applied together to increase the performance of OS-ELM, when updating the input weight of the OS-ELM using ELM-AE. One left problem is that in case of online prediction problem, usually the input data arrives one-by-one, which means applying batch normalization is impossible because batch normalization requires batch size bigger than 1 at least. To avoid the problem, we applied a different normalization method, called layer normalization (LN). The details will be discussed in the following subsection.

### B. Online Recurrent Extreme Learning Machine

OR-ELM is an improved OS-ELM, where two new approaches are applied to the conventional OS-ELM: auto-encoding with normalization and feedback input for the RNN structure. The overall structure of OR-ELM is shown in Fig. 4.

We start by describing the configuration of OR-ELM. OR-ELM consists of three networks: an RNN, which is a main network for the prediction, and two SLFNs, which are auxiliary ELM-AE networks for learning RNN’s input weight and hidden weight. In case of the RNN, its $n$ dimensional input layer is connected to an $L$ dimensional hidden layer by an input weight $W \in \mathbb{R}^{L \times n}$, and the hidden layer is connected to an $m$ dimensional output layer by an output weight $\beta \in \mathbb{R}^{m \times L}$. Furthermore, the hidden layer is also connected to itself, by a hidden weight $V \in \mathbb{R}^{L \times L}$.

We also define two auxiliary ELM-AEs, one for updating the input weight $W$ and the other for updating the hidden weight $V$. We call the former ELM-AE for input weight (ELM-AE-IW), and the latter ELM-AE for hidden weight (ELM-AE-HW). In case of the ELM-AE-IW, its $n$ dimensional input layer is connected to an $L$ dimensional hidden layer by an input weight $W^i \in \mathbb{R}^{L \times n}$, and the hidden layer is connected to an $n$ dimensional output layer by an output weight $\beta^i \in \mathbb{R}^{n \times L}$. Note that ELM-AE-IW’s input layer dimension and output layer dimension are the same as the input layer dimension of OR-ELM, and ELM-AE-IW’s hidden layer dimension is the same as that of OR-ELM. In case of the ELM-AE-HW, its $L$ dimensional input layer is connected to an $L$ dimensional hidden layer by an input weight $W^h \in \mathbb{R}^{L \times L}$, and the hidden layer is connected to an $L$ dimensional output layer by an output weight $\beta^h \in \mathbb{R}^{L \times L}$. Note that ELM-AE-HW’s input layer dimension, output layer dimension, and
hidden layer dimension are the same as the hidden layer dimension of OR-ELM.

OR-ELM consists of two phases: initialization phase and sequential learning phase.

C. Initialization phase

Now we initialize OR-ELM. For the RNN, we set its initial output weight $\beta_0$ and an initial auxiliary matrix $P_0$ using Eq. 9. The initial value of its hidden layer output $H_0$ is randomly generated with a mean-zero and a standard deviation of one. Then we also initialize ELM-AEs. ELM-AE-IW’s input weight $W^i$ and ELM-AE-HW’s input weight $W^h$ are also randomly assigned with a mean-zero and a standard deviation of one. Their output weights $\beta_{0i}, \beta_{0h}$ and the corresponding auxiliary matrices $P_0, P_0^i$ are initialized using Eq. 9.

D. Online sequential learning phase

Whenever a new chunk of input data with $N_{k+1}$ training samples arrives, where where $k+1$ indicates the $(k+1)$th chunk of input data with $k$ increasing from zero, below learning procedure is conducted. For mathematical simplicity, the chunk size $N_{k+1}$ is set to one.

a) Update the input weight: First of all, the input weight of OR-ELM is updated using ELM-AE-IW. ELM-AE-IW propagates the $(k+1)$th input sample $x(k+1) \in \mathbb{R}^{n \times 1}$ to the hidden layer so that its hidden-layer output matrix $H_{k+1}^i$ is calculated as below:

$$H_{k+1}^i = g(\text{norm}(W_{k+1}^i x(k+1)))$$

(13)

where

$$\text{norm}(x) = \frac{x - \mu^i}{\sqrt{\sigma^i^2 + \epsilon}},$$

$$\mu^i = \frac{1}{L} \sum_{j=1}^L x_j,$$

$$\sigma^i = \frac{1}{L} \sum_{j=1}^L (x_j - \mu^i)^2$$

(16)

Note that the $\text{norm}$ function is added before the non-linear activation as a LN procedure, to prevent the internal covariate shift problem. Then we calculate ELM-AE-IW’s output weight $\beta_{k+1}^i$ using RLS:

$$\beta_{k+1}^i = \beta_k^i + P_{k+1}^i H_{k+1}^i (x(k+1) - H_k^i \beta_k^i),$$

(17)

$$P_{k+1}^i = \frac{1}{\lambda} P_k^i - P_k^i H_k^i (\lambda^2 + \lambda H_k^i P_k^i H_k^i)^{-1} H_k^i P_k^i$$

(18)

where $T_{k+1}$ in Eq. 10 is replaced by $x(k+1)$ to perform unsupervised auto-encoding. The transpose of $\beta_{k+1}^i$ is used as the input weight of OR-ELM $W_{k+1}^i$:

$$W_{k+1}^i = \beta_{k+1}^i$$

(19)

b) Update the hidden weight: In the same way, the hidden weight of OR-ELM is updated using ELM-AE-HW. ELM-AE-HW propagates the OR-ELM’s $(k)$th hidden layer output $H_k \in \mathbb{R}^L$ to its hidden layer so that its hidden-layer output matrix $H_{k+1}^h$ is calculated as below:

$$H_{k+1}^h = g(\text{norm}(W_{k+1}^h H_k))$$

(20)

Then we calculate ELM-AE-HW’s output weight $\beta_{k+1}^h$ using RLS:

$$\beta_{k+1}^h = \beta_k^h + P_{k+1}^h H_{k+1}^h (H_k - H_{k+1}^h \beta_k^h),$$

(21)

$$P_{k+1}^h = \frac{1}{\lambda} P_k^h - P_k^h H_k^h (\lambda^2 + \lambda H_k^h P_k^h H_k^h)^{-1} H_k^h P_k^h$$

(22)
ELM’s output weight

ELM’s hidden-layer output matrix

as the hidden weight of OR-ELM

measured in every 30 min interval.

Fig. 5: An example portion of NYC Taxi passenger count, measured in every 30 min interval.

where \( T_{k+1} \) in Eq. 10 is replaced by \( H_k \) to perform unsupervised auto-encoding. The transpose of \( \beta_{k+1}^h \) is used as the hidden weight of OR-ELM \( V_{k+1} \):

\[
V_{k+1} = \beta_{k+1}^h H_k^T
\]  

where \( \beta_{k+1}^h \) is the hidden output weight of OR-ELM and \( H_k \) is the hidden-layer output matrix of OR-ELM.

c) Feedforward propagation: Now we calculate the OR-ELM’s hidden-layer output matrix \( H_{k+1} \) for the \((k+1)\)th input \( x(k+1) \) as follows:

\[
H_{k+1} = g(\text{norm}(W_{k+1} x(k+1) + V_{k+1} H_k))
\]

d) Update the output weight: Finally, we update the OR-ELM’s output weight \( \beta_{k+1} \) using Eq. 10, and Eq. 11.

Now that the proposed OR-ELM is not limited to single-hidden layer, but can have \( m \) hidden layers. In this case, the OR-ELM has \( m \) ELM-AE-IWs, \( m \) ELM-AE-HWs. In this deeper structure, OR-ELM is able to learn more complex patterns.

IV. EXPERIMENTS AND RESULTS

We evaluate the proposed OR-ELM on the NYC taxi passenger count dataset [9] that consists of a time-series sequence of length 17520. In the dataset, NYC taxi passenger count is aggregated in every 30 min window. An example portion of the dataset is shown in Fig. 5, depicting 17 days from Jan. 30th, 2015 to Feb. 15th, 2015. The goal with this dataset is to predict the NYC taxi passenger demand of 2.5 hours (5 days) using a time lag of 100 steps. The output dimension of OR-ELM was set to 1. In other words, the proposed model is continually trained with a sequence of past 100 steps as an input, and with a 5 step future value as a target output.

To compare the performance of OR-ELM with the other neural network models, we used two error metrics: normalized root mean square error (NRMSE), and mean absolute percentage error (MAPE). NRMSE error is known as the most popular error metric to measure the difference between values predicted by a model and the values actually observed. MAPE is an error metric that is less sensitive to outliers than NRMSE, which is calculated as follows:

\[
MAPE = \frac{\sum_{k=1}^{N} |T_k - y(k)|}{\sum_{k=1}^{N} |T_k|}
\]

where \( N \) is the length of the dataset, \( T_k \) is the observed data at time \( k \), and \( y(k) \) is the prediction of the model at time \( k \).

A. Comparison with basic OS-ELM

We compared the performance of the proposed OR-ELM with the basic OS-ELM varying the forgetting factor \( \lambda \) from 1 to 0.9, as shown in Fig. 6. In Fig. 6, the red bars are the errors of the basic OS-ELM, the green bars are the errors of OS-ELM with LN method applied, the blue bars are the errors of OS-ELM with LN and auto-encoding applied, and the gray bars are the errors of the proposed OR-ELM, where LN, auto-encoding, and recurrent input is applied. In case of the third option where LN and auto-encoding applied, the hidden weight \( V \) in OR-ELM is excluded. For each \( \lambda \), we varied the number of hidden nodes from 10 to 1600.

We analyzed the result in terms of following four perspectives: forgetting factor, LN, auto-encoding, and recurrent input:

1) Forgetting factor:
   - When \( \lambda \) equals 1, the error tends to converge to its minimum as the number of the hidden nodes increases.
   - When \( \lambda \) is lower than 1, the error rapidly decreases as the number of the hidden nodes increases, then the error explodes if the number of the hidden nodes is too huge.
   - The smaller the \( \lambda \) is, the faster the error decreases.

2) Layer normalization:
   - LN method tends to increase the performance of OSELM except some rare cases.

3) Auto-encoding:
   - Auto-encoding method works only when applied together with LN method.
   - When forgetting factor equals 1, auto-encoding method works only if the number of the hidden nodes is small enough.
   - When forgetting factor is lower than 1, auto-encoding method tends to work in almost cases, and shows better performance than the basic OS-ELM.

4) Recurrent input:
   - Recurrent input works only when applied together with LN and auto-encoding method.
   - When applied with LN and auto-encoding methods, the recurrent input further increases the performance of OS-ELM.

B. Comparison with other sequential learning methods

Finally, we compared the performance of the proposed OR-ELM with other sequential learning algorithms: OS-ELM, HTM and LSTM. For HTM and LSTM, we used the implementation [9] where the source code is available in [10]. For OS-ELM and OR-ELM, we used the forgetting factor \( \lambda = 0.91 \), and the number of hidden nodes \( L = 23 \), which
showed the best performance in the NYC taxi passenger count dataset. We also tested additional two options: OS-ELM with LN applied, and OS-ELM with LN and input weight auto-encoding. The parameters of those methods are chosen the same as those of OR-ELM. The results are described in Table I. According to the results, the proposed OR-ELM outperforms the conventional OS-ELM, with the sacrifice of computation time. OS-ELM showed better performance if LN is applied. Then its performance has further enhanced when input weight or hidden weight is updated using auto-encoders. We also found that the proposed OR-ELM outperforms the conventional OS-ELM, and the OR-ELM is comparable or slightly better than other state-of-the-art sequential learning algorithms. For both NRMSE and MAPE metrics OR-ELM showed lower errors than those of OS-ELM, HTM and LSTM. Although OR-ELM showed slower computation speed than that of the basic OS-ELM, OR-ELM is still much more faster than HTM and LSTM.

**TABLE I:** The comparison between the propose OR-ELM and other sequential learning methods.

<table>
<thead>
<tr>
<th>Methods</th>
<th>NRMSE</th>
<th>MAPE (%)</th>
<th>Computation time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>OS-ELM</td>
<td>0.3856</td>
<td>12.5</td>
<td>2.83</td>
</tr>
<tr>
<td>OS-ELM with LN</td>
<td>0.3090</td>
<td>9.8</td>
<td>4.03</td>
</tr>
<tr>
<td>OS-ELM with LN, and AE-IW</td>
<td>0.2566</td>
<td>8.4</td>
<td>9.24</td>
</tr>
<tr>
<td>LSTM</td>
<td>0.2542</td>
<td>8.2</td>
<td>133.19</td>
</tr>
<tr>
<td>HTM</td>
<td>0.2704</td>
<td>7.8</td>
<td>223.12</td>
</tr>
<tr>
<td>OR-ELM</td>
<td>0.2096</td>
<td>6.8%</td>
<td>13.74</td>
</tr>
</tbody>
</table>

1) **Learning rate:** The next thing we observed is the converging speed or learning rate of the proposed algorithm. In Fig. 7a, the error curve in the period of the first 40 days of the entire sequence is shown. We recorded the mean NRMSE
of the past 100 samples. According to the result, the initial error of the OR-ELM is the lowest, and its error is further decreased and maintained, whereas that of HTM is initially very high and maintained higher than that of OR-ELM. The tendency of the error curve of LSTM is shown similar as that of OR-ELM, but that of LSTM is slightly higher than that of OR-ELM.

2) Adaptation ability to new patterns: We also observed the proposed model’s prediction ability when the input data flow is suddenly changing: the result is shown in Fig. 7b. In Fig. 7b, the y-axis is NRMSE over the last 100 samples. Note that the x-axis of Fig. 7b is the same as that of Fig. 5. We can find that some sudden pattern change occurred near Feb. 2nd, 2015 in Fig. 5. The result shows that OR-ELM rapidly adapted to the pattern change so that OR-ELM minimized the prediction error, while HTM and LSTM had difficulties adapting to the new pattern, which resulted in high NRMSE.

Further increases the performance of OR-ELM. Performance evaluation of the OR-ELM is conducted to compare the proposed method with other sequential learning algorithms such as OS-ELM, HTM, and LSTM on the time-series prediction of NYC taxi passenger demand. According to the results, OR-ELM showed far better performance than conventional OS-ELM. comparable or slightly better performance than HTM, and LSTM. Furthermore, the results showed that OR-ELM not only quickly learn input patterns in the initial learning phase, OR-ELM is also able to quickly adapt to the change of input patterns than HTM and LSTM. Therefore, we can conclude that the proposed OR-ELM is more suitable than other state-of-the-art sequential learning algorithms such as HTM and LSTM, for online learning based time-series prediction, where the ability of quick adaptation to new patterns is demanded.

ACKNOWLEDGMENT

This work was supported by the ICT R&D program of MSIP/IITP. [2016-0-00563, Research on Adaptive Machine Learning Technology Development for Intelligent Autonomous Digital Companion]

REFERENCES


Fig. 7: Prediction error (NRMSE) curve.

V. CONCLUSION

In this paper, an improved OS-ELM, named OR-ELM, is proposed to overcome the drawbacks of OS-ELM. The conventional OS-ELM is not suitable for updating input weight, or having a recurrent input like RNN, due to the internal covariate shift problem. Therefore we solved the problem in OR-ELM by applying LN layers. In OR-ELM, ELM-AE is used to extract better hidden-layer features rather than random hidden-layer features of the conventional OS-ELM. Furthermore, Recurrent input with hidden weight in OR-ELM