Online Supplement for "An Intelligent End-to-End Neural Architecture Search Framework for Electricity Forecasting Model Development"

Appendix A: RNN and LSTM Reformulation

Function-preserving transformation is a key step in architecture design, which ensures that the architecture after transformation has the same performance as its predecessor. To realize this in RNN, we first propose a general operation scheme in this section to unify the feature learning operation in an ordinary neural network (e.g., FCN). And then, we reformulate the feature learning operation of RNN and LSTM based on this unified scheme.

A.1. Feature learning operation

RNN is able to pass information through time. In addition to receiving information from the input features at the current time step, the output features from the previous time steps are also incorporated to generate the RNN outputs. However, the function-preserving transformation method from Chen et al. (2016) only considers the temporal independent situation, which cannot change the hidden state at the current time step when the outputs from the previous time steps are changed. Therefore, the EAS framework (Cai et al. 2018) built on the Net2Net framework (Chen et al. 2016) that targets image data, cannot be intuitively used in time-series data. Regarding the excellence of RNN in preprocessing time-series data and extracting features from them, it is necessary to develop the RNN function-preserving transformation so that the EAS framework can be adapted to such data and improve the search quality of network structures for power systems. To prepare for deriving the RNN function-preserving transformation, we first propose a general two-phase formulation of the feature learning process in a deep learning model.

Suppose that for layer i in an FCN, there are p_i input features $\boldsymbol{x}^{(i)} = (x_1^{(i)},\ x_2^{(i)}, \dots, x_{p_i}^{(i)})^\top \in \mathbb{R}^{p_i \times 1}$ and p_{i+1} output features $\boldsymbol{x}^{(i+1)} = (x_1^{(i+1)},\ x_2^{(i+1)}, \dots, x_{p_{i+1}}^{(i+1)})^\top \in \mathbb{R}^{p_{i+1} \times 1}$. $\boldsymbol{x}^{(i+1)}$ will be fed into layer $i+1$ as the input. The weight matrix for layer i is $\boldsymbol{W}^{(i)} \in \mathbb{R}^{p_i \times p_{i+1}}$ and its transpose is represented by $\boldsymbol{W}^{(i)^\top} \in \mathbb{R}^{p_{i+1} \times p_i}$. The forward calculation of FCN can be represented by Eq. (A.1), where f_a is the activation function. Note that we ignore the bias term in this equation for notation convenience, and the bias can be assumed to be integrated into the weight matrix.

$$\boldsymbol{x}^{(i+1)} = f_a(\boldsymbol{W}^{(i)^\top} \boldsymbol{x}^{(i)}) \tag{A.1}$$

The Phase-1 learning is specified as a function $f_1(x_j^{(i)}, W_{j,k}^{(i)}) = x_j^{(i)} \cdot W_{j,k}^{(i)}$, for $j = 1, 2, \dots, p_i$ and $k = 1, 2, \dots, p_{i+1}$, which is a scalar multiplication. Subsequently, the Phase-1 learning results $\boldsymbol{o}_k$ for layer i are obtained as follows:

$$\boldsymbol{o}_k^{(i)} = \left(f_1(x_1^{(i)}, W_{1,k}^{(i)}), f_1(x_2^{(i)}, W_{2,k}^{(i)}), \dots, f_1(x_{p_i}^{(i)}, W_{p_i,k}^{(i)})\right), k = 1, \dots, p_{i+1} \tag{A.2}$$

The Phase-2 learning is denoted by an aggregating function that summarizes information from the Phase-1 learning process, $f_2(\boldsymbol{o}_k^{(i)}) = \sum_{j=1}^{p_i} f_1(x_j^{(i)}, W_{j,k}^{(i)})$, for $k = 1, \dots, p_{i+1}$. Then, we generate the output features $\boldsymbol{x}^{(i+1)}$, which are given by

$$\boldsymbol{x}^{(i+1)} = f_a\left(f_2(\boldsymbol{o}_1^{(i)}), f_2(\boldsymbol{o}_2^{(i)}), \dots, f_2(\boldsymbol{o}_{p_{i+1}}^{(i)})\right) \tag{A.3}$$

Fig. A.1 depicts a simple FCN example of such a feature learning process. As shown in the figure, layer i has two input and two output features. Hence, the meta operation f_1 is utilized to do the mathematical operation (scalar multiplication here) with the input features (i.e., Phase-1 learning), and the aggregation operation f_2 is employed to accumulate the learned knowledge to produce new features (i.e., Phase-2 learning). Finally, we acquire input features of the next layer through an activation function f_a.

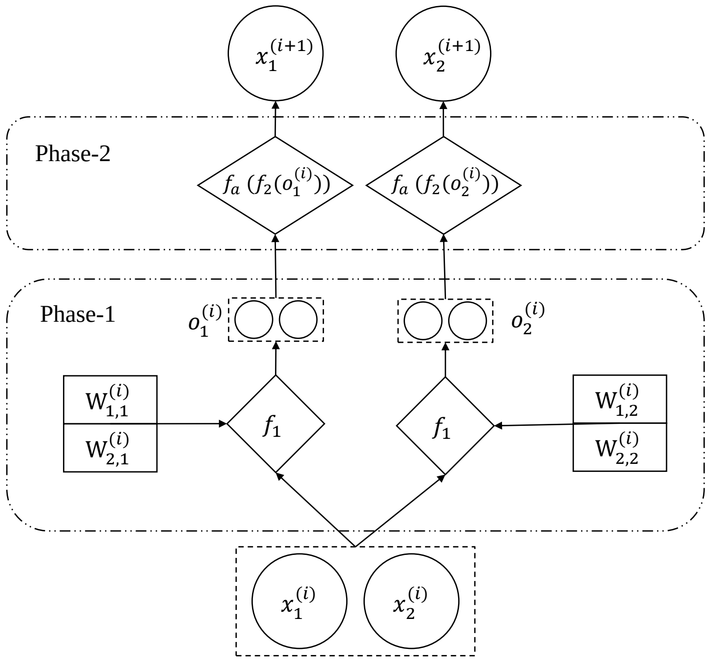


Figure A.1 An example of two-phase feature learning operation in FCN

As noted, such a two-phase description of the feature learning process can be easily extended to CNN. Briefly, for the Phase-1 learning of a CNN layer, a convolution operation is performed on each input channel with a kernel, generating the same number of intermediate results as that of the input features. For the Phase-2 learning, all the intermediate results are summed up to generate one new input feature for the next layer, and this process is repeated multiple times to obtain all the output features. For instance, there are p_i channels of inputs and p_{i+1} channels of outputs for layer i. Hence, we employ p_i kernels to get the intermediate results from inputs $\boldsymbol{x}^{(i)}$. We then have intermediate results:

$$o\boldsymbol{K}_{j,k}^{(i)} = f_1(\boldsymbol{x}_j^{(i)}, \boldsymbol{k}_{j,k}^{(i)}), j = 1,\ldots,p_i, k = 1,\ldots,p_{i+1}, \tag{A.4}$$

where f_1 is a convolution function, $\boldsymbol{x}_j^{(i)}$ is the j^{th} channel of input tensor $\boldsymbol{x}^{(i)}$ and $\boldsymbol{k}_{j,k}^{(i)}$ is a convolution kernel. Thus, we have the Phase-1 learning results $\boldsymbol{o}_k^{(i)} = (o\boldsymbol{K}_{1,k}^{(i)},\ldots,o\boldsymbol{K}_{p_i,k}^{(i)})$. After that, all the intermediate results are summed up to form one output feature in Phase-2 learning:

$$\boldsymbol{x}_k^{(i+1)} = f_2(\boldsymbol{o}_k^{(i)}) = \sum_{j=1}^{p_i} o\boldsymbol{K}_{j,k}^{(i)}, k = 1,\ldots,p_{i+1}. \tag{A.5}$$

This process is repeated p_{i+1} times to achieve p_{i+1} output features. In total, we use $p_i \times p_{i+1}$ kernels to extract the features from the inputs $\boldsymbol{x}^{(i)}$. With these two-phase feature learning process described above, we can obtain the outputs $\boldsymbol{x}^{(i+1)}$ as new input features for layer $i+1$.

A.2. RNN reformulation

Following the settings of the feature learning process in spatial networks (e.g., FCN), we assume that at time t, an RNN layer i has p_i input features $\boldsymbol{x}_t^{(i)} = (x_{1;t}^{(i)},\ x_{2;t}^{(i)}, \dots, x_{p_i;t}^{(i)})^\top \in \mathbb{R}^{p_i \times 1}$ and p_{i+1} output features $\boldsymbol{x}_t^{(i+1)} = (x_{1;t}^{(i+1)},\ x_{2;t}^{(i+1)}, \dots, x_{p_{i+1};t}^{(i+1)})^\top \in \mathbb{R}^{p_{i+1} \times 1}$. As seen, we have used a semicolon in subscript notation to distinguish the time information from other information. We assume the input weight matrix as $\boldsymbol{W}^{(i)} \in \mathbb{R}^{p_i \times p_{i+1}}$ and the output hidden state weight matrix as $\boldsymbol{H}^{(i)} \in \mathbb{R}^{p_{i+1} \times p_{i+1}}$. Then, the ordinary RNN feature learning process is defined as:

$$\boldsymbol{x}_t^{(i+1)} = f_a({\boldsymbol{W}^{(i)}}^\top \boldsymbol{x}_t^{(i)} + {\boldsymbol{H}^{(i)}}^\top \boldsymbol{x}_{t-1}^{(i+1)}). \tag{A.6}$$

In the following, we present how to reformulate the RNN feature learning process with the proposed two-phase operation. The meta operation f_1 for RNN is a scalar multiplication and can be used to extract the information from $\boldsymbol{x}_t^{(i)}$ and $\boldsymbol{x}_{t-1}^{(i+1)}$. Therefore, we specify the Phase-1 learning results of layer i at time t related to $\boldsymbol{W}^{(i)}$ and $\boldsymbol{H}^{(i)}$ as follows:

$$oW_{j,k;t}^{(i)} = f_1(x_{j;t}^{(i)}, W_{j,k}^{(i)}), j = 1, 2, \dots, p_i, k = 1, \dots, p_{i+1}, \tag{A.7}$$

$$oH_{l,k;t}^{(i)} = f_1(x_{l;t-1}^{(i+1)}, H_{l,k}^{(i)}), l = 1, 2, \dots, p_{i+1}, k = 1, \dots, p_{i+1}. \tag{A.8}$$

After that, we obtain the Phase-1 learning results $\boldsymbol{o}_{k;t}^{(i)}$ of layer i at time t:

$$\boldsymbol{o}_{k;t}^{(i)} = (oW_{1,k;t}^{(i)}, \dots oW_{p_i,k;t}^{(i)}, oH_{1,k;t}^{(i)}, \dots, oH_{p_{i+1},k;t}^{(i)}), k = 1, \dots, p_{i+1}. \tag{A.9}$$

In Phase-2 learning, we sum up the information from the Phase-1 learning results with the use of the aggregating function $f_2(\boldsymbol{o}_{k;t}^{(i)}) = \sum_{j=1}^{p_i} oW_{j,k;t}^{(i)} + \sum_{l=1}^{p_{i+1}} oH_{l,k;t}^{(i)}$, for $k = 1, \dots, p_{i+1}$. Hence, we generate the new outputs $\boldsymbol{x}_t^{(i+1)}$ at time t through the following equation:

$$\boldsymbol{x}_t^{(i+1)} = (f_a(f_2(\boldsymbol{o}_{1;t}^{(i)})), f_a(f_2(\boldsymbol{o}_{2;t}^{(i)})), \dots, f_a(f_2(\boldsymbol{o}_{p_{i+1};t}^{(i)}))). \tag{A.10}$$

An example of such a two-phase feature learning in RNN is depicted in Fig. A.2. As shown, we have two input features at time t and three output features at time $t-1$. Both input and output features perform the element-wise meta operation f_1 with its corresponding weight matrices, and then we acquire the intermediate results. Subsequently, we utilize the aggregating function f_2 to generate new output features of RNN at time t. In the following, we present how to reformulate the LSTM with the two-phase feature learning process.

A.3. LSTM reformulation

Different from the general RNN architecture, LSTM is able to explore the long-term dependencies in the data by employing a series of memory cells, leading to a much more complicated function-preserving transformation than that of RNN in the Net2Net framework (Chen et al. 2016). Similar to RNN, we also assume that at time t, an LSTM layer i has p_i input features $\boldsymbol{x}_t^{(i)} = (x_{1;t}^{(i)},\ x_{2;t}^{(i)}, \dots, x_{p_i;t}^{(i)})^\top \in \mathbb{R}^{p_i \times 1}$ and p_{i+1} output

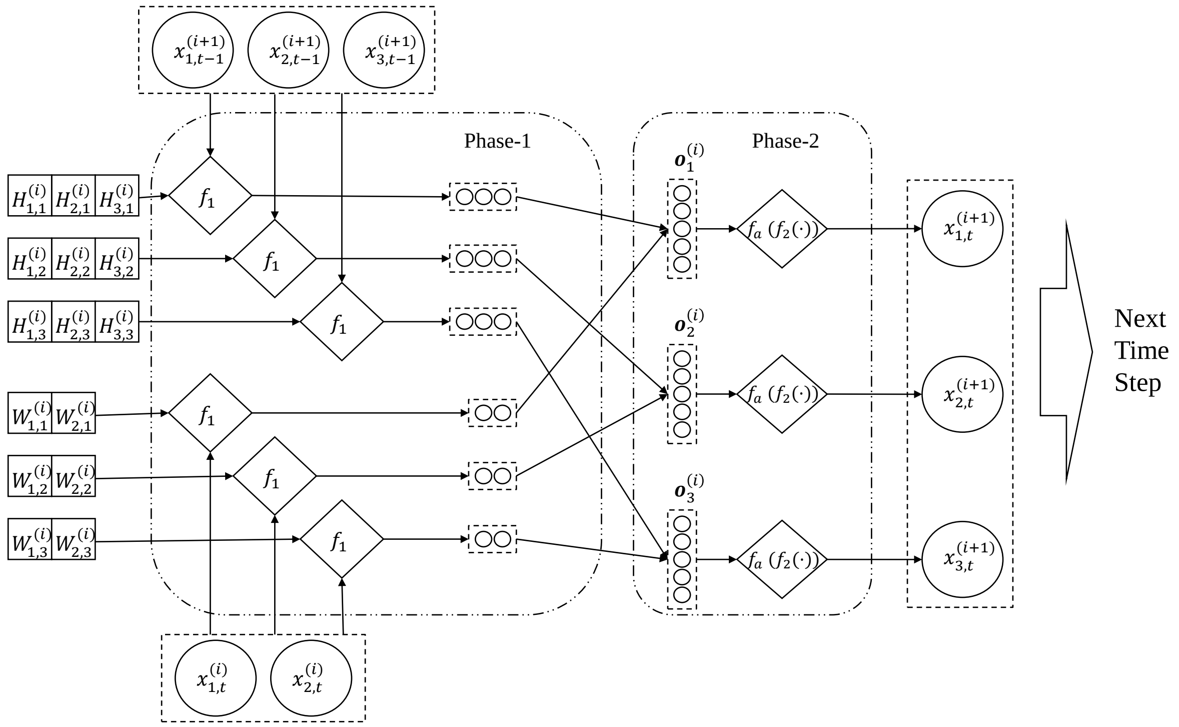


Figure A.2 **An example of a two-phase feature learning operation in RNN**

features $\boldsymbol{x}_t^{(i+1)} = (x_{1;t}^{(i+1)},\ x_{2;t}^{(i+1)}, \dots, x_{p_{i+1};t}^{(i+1)})^\top \in \mathbb{R}^{p_{i+1}\times 1}$. The calculation of the LSTM process is shown in Eq. (A.11).

$$\begin{aligned}
\boldsymbol{f}_t^{(i)} &= \sigma_g(\boldsymbol{W}_f^{(i)\top}\boldsymbol{x}_t^{(i)} + \boldsymbol{H}_f^{(i)\top}\boldsymbol{x}_{t-1}^{(i+1)})\\
\boldsymbol{i}_t^{(i)} &= \sigma_g(\boldsymbol{W}_i^{(i)\top}\boldsymbol{x}_t^{(i)} + \boldsymbol{H}_i^{(i)\top}\boldsymbol{x}_{t-1}^{(i+1)})\\
\boldsymbol{o}_t^{(i)} &= \sigma_g(\boldsymbol{W}_o^{(i)\top}\boldsymbol{x}_t^{(i)} + \boldsymbol{H}_o^{(i)\top}\boldsymbol{x}_{t-1}^{(i+1)})\\
\tilde{\boldsymbol{c}}_t^{(i)} &= \sigma_c(\boldsymbol{W}_c^{(i)\top}\boldsymbol{x}_t^{(i)} + \boldsymbol{H}_c^{(i)\top}\boldsymbol{x}_{t-1}^{(i+1)})\\
\boldsymbol{c}_t^{(i)} &= \boldsymbol{f}_t^{(i)}\circ\boldsymbol{c}_{t-1}^{(i)} + \boldsymbol{i}_t^{(i)}\circ\tilde{\boldsymbol{c}}_t^{(i)}\\
\boldsymbol{x}_t^{(i+1)} &= \boldsymbol{o}_t^{(i)}\circ\sigma_c(\boldsymbol{c}_t^{(i)})
\end{aligned} \tag{A.11}$$

In the LSTM formulations, $\boldsymbol{f}_t^{(i)}, \boldsymbol{i}_t^{(i)}, \boldsymbol{o}_t^{(i)}$ and $\boldsymbol{c}_t^{(i)}$ denote the forget gate, input gate, output gate, and memory cell vectors at time t, respectively; $\tilde{\boldsymbol{c}}_t^{(i)}$ denotes the cell input activation vector; $\boldsymbol{x}_t^{(i+1)}$ also denotes the hidden state for layer i; operator $\circ$ represents the Hadamard product, which calculates the element-wise products of two same dimensional vectors or matrices; σ_g and σ_c represent the Sigmoid activation function and the hyperbolic tangent function, respectively; $\boldsymbol{W}_f^{(i)} \in \mathbb{R}^{p_i\times p_{i+1}}, \boldsymbol{W}_i^{(i)} \in \mathbb{R}^{p_i\times p_{i+1}}$, $\boldsymbol{W}_o^{(i)} \in \mathbb{R}^{p_i\times p_{i+1}}$, $\boldsymbol{W}_c^{(i)} \in \mathbb{R}^{p_i\times p_{i+1}}$, $\boldsymbol{H}_f^{(i)} \in \mathbb{R}^{p_{i+1}\times p_{i+1}}$, $\boldsymbol{H}_i^{(i)} \in \mathbb{R}^{p_{i+1}\times p_{i+1}}$, $\boldsymbol{H}_o^{(i)} \in \mathbb{R}^{p_{i+1}\times p_{i+1}}$ and $\boldsymbol{H}_c^{(i)} \in \mathbb{R}^{p_{i+1}\times p_{i+1}}$ are weight parameters.

As introduced in Chen et al. (2016), the identity mapping function is paramount to the function-preserving transformation of deepening the networks. However, the LSTM calculation process in Eq. (A.11) contains a lot of activation functions to calculate $\boldsymbol{x}_t^{(i+1)}$, which directly prevent the application of identity mapping. Hence, we restructure the forward propagation of LSTM in Eq. (A.11) by altering the formulations of both $\boldsymbol{o}_t^{(i)}$ and $\boldsymbol{x}_t^{(i+1)}$ as

$$\begin{aligned}\boldsymbol{o}_t^{(i)} &= \min\{\max\{\boldsymbol{W}_o^{(i)\top}\boldsymbol{x}_t^{(i)} + \boldsymbol{H}_o^{(i)\top}\boldsymbol{x}_{t-1}^{(i+1)}, 1\}, 0\},\\ \boldsymbol{x}_t^{(i+1)} &= \min\{\max\{\boldsymbol{o}_t^{(i)} \circ \sigma_c(\boldsymbol{c}_t^{(i)}) + \boldsymbol{o}_t^{(i)}, 1\}, -1\},\end{aligned} \tag{A.12}$$

where $\min(\boldsymbol{a}, a)$ ($\max(\boldsymbol{a}, a)$) of vector $\boldsymbol{a}$ and constant a gives the element-wise minimum (maximum) value. Note that the output ranges of $o_t^{(i)}$ and $x_t^{(i+1)}$ in Eq. (A.12) are the same as the original ranges [0, 1] and [-1, 1] by the truncate operation, respectively. Moreover, we add an additional $\boldsymbol{o}^{(i)}$ in the second term for the following reasons. The original output $\boldsymbol{o}_t^{(i)} \circ \sigma_c(\boldsymbol{c}_t^{(i)})$ always modifies the input $\boldsymbol{o}^{(i)}$ because the multiplication term $\sigma_c(\boldsymbol{c}_t^{(i)})$ cannot be set to 1. To preserve the capability of passing the input unchanged to the next layer, we add a new term $\boldsymbol{o}^{(i)}$. In this case, when the multiplication factor $\sigma_c(\boldsymbol{c}_t^{(i)})$ is set to 0, the modified LSTM is an identity mapping to preserve and pass the input information, which is especially important in building deeper models. For LSTM, the meta operation f_1 is also a scalar multiplication and can be used to extract the information from $\boldsymbol{x}_t^{(i)}$ and $\boldsymbol{x}_{t-1}^{(i+1)}$ (i.e., $f_1(a,b) = a \cdot b$). We specify the intermediate results as follows:

$$\begin{aligned}
oW_{f;j,k;t}^{(i)} &= f_1(x_{j;t}^{(i)}, W_{f;j,k}^{(i)}), j = 1,2,\ldots,p_i, k = 1,\ldots,p_{i+1};\\
oW_{i;j,k;t}^{(i)} &= f_1(x_{j;t}^{(i)}, W_{i;j,k}^{(i)}), j = 1,2,\ldots,p_i, k = 1,\ldots,p_{i+1};\\
oW_{o;j,k;t}^{(i)} &= f_1(x_{j;t}^{(i)}, W_{o;j,k}^{(i)}), j = 1,2,\ldots,p_i, k = 1,\ldots,p_{i+1};\\
oW_{c;j,k;t}^{(i)} &= f_1(x_{j;t}^{(i)}, W_{c;j,k}^{(i)}), j = 1,2,\ldots,p_i, k = 1,\ldots,p_{i+1};\\
oH_{f;l,k;t}^{(i)} &= f_1(x_{l;t-1}^{(i+1)}, H_{f;l,k}^{(i)}), l = 1,2,\ldots,p_{i+1}, k = 1,\ldots,p_{i+1};\\
oH_{i;l,k;t}^{(i)} &= f_1(x_{l;t-1}^{(i+1)}, H_{i;l,k}^{(i)}), l = 1,2,\ldots,p_{i+1}, k = 1,\ldots,p_{i+1};\\
oH_{c;l,k;t}^{(i)} &= f_1(x_{l;t-1}^{(i+1)}, H_{c;l,k}^{(i)}), l = 1,2,\ldots,p_{i+1}, k = 1,\ldots,p_{i+1};\\
oH_{o;l,k;t}^{(i)} &= f_1(x_{l;t-1}^{(i+1)}, H_{o;l,k}^{(i)}), l = 1,2,\ldots,p_{i+1}, k = 1,\ldots,p_{i+1}.
\end{aligned} \tag{A.13}$$

After that, we obtain the Phase-1 learning results $\boldsymbol{o}_{k;t}^{(i)}$ of LSTM layer i at time t:

$$\begin{aligned}
\boldsymbol{o}_{k;t}^{(i)} = (&oW_{f;1,k;t}^{(i)}, \ldots oW_{f;p_i,k;t}^{(i)}, oW_{i;1,k;t}^{(i)}, \ldots oW_{i;p_i,k;t}^{(i)},\\
&oW_{c;1,k;t}^{(i)}, \ldots oW_{c;p_i,k;t}^{(i)}, oW_{o;1,k;t}^{(i)}, \ldots oW_{o;p_i,k;t}^{(i)},\\
&oH_{f;1,k;t}^{(i)}, \ldots, oH_{f;p_{i+1}k;t}^{(i)}, oH_{i;1,k;t}^{(i)}, \ldots, oH_{i;p_{i+1}k;t}^{(i)},\\
&oH_{c;1,k;t}^{(i)}, \ldots, oH_{c;p_{i+1}k;t}^{(i)}, oH_{o;1,k;t}^{(i)}, \ldots, oH_{o;p_{i+1}k;t}^{(i)}), k = 1,\ldots,p_{i+1}.
\end{aligned} \tag{A.14}$$

According to LSTM formulation in Eq. (A.11) and (A.12), we can aggregate all these intermediate results with the aggregating function $f_{2;c}$ for the cell state and $f_{2;x}$ for the hidden state as

$$\begin{aligned}
c_{k;t}^{(i)} = f_{2;c}(\boldsymbol{o}_{k;t}^{(i)}) = \sigma_g\Big(\sum_{j=1}^{p_i} oW_{f;j,k;t}^{(i)} + \sum_{l=1}^{p_{i+1}} oH_{f;l,k;t}^{(i)}\Big) \cdot c_{k;t-1}^{(i)}\\
+\sigma_g\Big(\sum_{j=1}^{p_i} oW_{i;j,k;t}^{(i)} + \sum_{l=1}^{p_{i+1}} oH_{i;l,k;t}^{(i)}\Big) \cdot \sigma_c\Big(\sum_{j=1}^{p_i} oW_{c;j,k;t}^{(i)} + \sum_{l=1}^{p_{i+1}} oH_{c;l,k;t}^{(i)}\Big), k = 1,\ldots,p_{i+1},
\end{aligned} \tag{A.15}$$

$$x_{k;t}^{(i+1)} = f_{2;x}(\boldsymbol{o}_{k;t}^{(i)}) = \Big(\sum_{j=1}^{p_i} oW_{o;j,k;t}^{(i)} + \sum_{l=1}^{p_{i+1}} oH_{o;l,k;t}^{(i)}\Big) \cdot (f_{2,c}(\boldsymbol{o}_{k;t}^{(i)}) + 1), k = 1,\ldots,p_{i+1}. \tag{A.16}$$

With these two aggregating functions in Phase-2 learning, we generate the new output $\boldsymbol{x}_t^{(i+1)}$ and cell state $\boldsymbol{c}_t^{(i)}$ at time t through the following equations:

$$\boldsymbol{c}_t^{(i)} = (f_{2;c}(\boldsymbol{o}_{1;t}^{(i)}), f_{2;c}(\boldsymbol{o}_{2;t}^{(i)}), \ldots, f_{2;c}(\boldsymbol{o}_{p_{i+1};t}^{(i)})), \tag{A.17}$$

$$\boldsymbol{x}_t^{(i+1)} = (f_{2;x}(\boldsymbol{o}_{1;t}^{(i)}), f_{2;x}(\boldsymbol{o}_{2;t}^{(i)}), \ldots, f_{2;x}(\boldsymbol{o}_{p_{i+1};t}^{(i)})). \tag{A.18}$$

Appendix B: Proof of Propositions

Proof of Proposition 1: First, we consider the feature extraction for the *wider* layer i'. We apply the new student network parameters of $\boldsymbol{W}^{(i')}$ and $\boldsymbol{H}^{(i')}$ in Proposition 1 to Eq. (A.7) and (A.8), and the extracted features at time t are given by:

$$oW^{(i')}_{j,k;t} = oW^{(i)}_{j,g(k);t}, j = 1, 2, \dots, p_i, k = 1, \dots, p'_{i+1}, \tag{B.1}$$

$$oH^{(i')}_{l,k;t} = f_w(l) oH^{(i)}_{g(l),g(k);t}, l = 1, 2, \dots, p'_{i+1}, k = 1, \dots, p'_{i+1}. \tag{B.2}$$

Following Eq. (A.9), the feature extraction results of layer i' at time t, $\boldsymbol{o}^{(i')}_{k;t}$, are given by:

$$\boldsymbol{o}^{(i')}_{k;t} = (oW^{(i')}_{1,k;t}, \dots oW^{(i')}_{p_i,k;t}, oH^{(i')}_{1,k;t}, \dots, oH^{(i')}_{p'_{i+1},k;t}), k = 1, \dots, p'_{i+1}. \tag{B.3}$$

Then, by Eq. (A.10) and the aggregating function $f_2(\boldsymbol{o}^{(i')}_{k;t}) = \sum_{j=1}^{p_i} oW^{(i')}_{j,g(k);t} + \sum_{l=1}^{p'_{i+1}} oH^{(i')}_{l,k;t}$, $k = 1, \dots, p'_{i+1}$, the new output $x^{(i'+1)}_{k;t}$ are given by:

$$\begin{aligned} x^{(i'+1)}_{k;t} &= f_a(f_2(\boldsymbol{o}^{(i')}_{k;t})) \\ &= f_a(\sum_{j=1}^{p_i} oW^{(i')}_{j,g(k);t} + \sum_{l=1}^{p'_{i+1}} oH^{(i')}_{l,k;t}) \\ &= f_a(\sum_{j=1}^{p_i} oW^{(i)}_{j,g(k);t} + \sum_{l=1}^{p'_{i+1}} f_w(l) oH^{(i)}_{g(l),g(k);t}) \\ &= f_a(\sum_{j=1}^{p_i} oW^{(i)}_{j,g(k);t} + \sum_{l=1}^{p_{i+1}} oH^{(i)}_{l,g(k);t}) \\ &= x^{(i+1)}_{g(k);t} \quad , \quad k = 1, \dots, p'_{i+1}, \end{aligned} \tag{B.4}$$

where the third equality holds by Eq. (B.1) and (B.2), the fourth equality holds by Eq. (3), and the last equality holds by the definition of the aggregating function and Eq. (A.10). Hence, the output feature vector $\boldsymbol{x}^{(i'+1)}_t$ of layer i' also serves as the input features of layer $i'+1$. Following Eq. (A.10), and $\boldsymbol{x}^{(i'+1)}_t$ can be identified as

$$\boldsymbol{x}^{(i'+1)}_t = (x^{(i+1)}_{g(1);t}, x^{(i+1)}_{g(2);t}, \dots, x^{(i+1)}_{g(p'_{i+1});t}). \tag{B.5}$$

Note that $\boldsymbol{x}^{(i'+1)}_t$ is essentially a random mapping defined by g. Similar to the RNN reformulation in wider layer i' as shown above, we leverage the new student network parameters of $\boldsymbol{W}^{(i'+1)}$ and $\boldsymbol{H}^{(i'+1)}$ in Proposition 1 to Eq. (A.7) and (A.8), and acquire the extracted features for the next layer $i'+1$ as follows:

$$oW^{(i'+1)}_{k,h;t} = f_w(k) oW^{(i+1)}_{g(k),h;t}, k = 1, 2, \dots, p'_{i+1}, h = 1, \dots, p_{i+2}, \tag{B.6}$$

$$oH^{(i'+1)}_{r,h;t} = oH^{(i+1)}_{r,h;t}, r = 1, 2, \dots, p_{i+2}, h = 1, \dots, p_{i+2}. \tag{B.7}$$

As observed in Eq. (B.7), the feature extraction for outputs of layer $i'+1$ is the same as that of layer $i+1$ since the dimension is unchanged. Following Eq. (A.9), the feature extraction results of layer $i'+1$ at time t, $\boldsymbol{o}^{(i'+1)}_{h;t}$, are given by

$$\boldsymbol{o}^{(i'+1)}_{h;t} = (oW^{(i'+1)}_{1,h;t}, \dots oW^{(i'+1)}_{p'_{i+1},h;t}, oH^{(i'+1)}_{1,h;t}, \dots, oH^{(i'+1)}_{p_{i+2},h;t}), h = 1, \dots, p_{i+2}. \tag{B.8}$$

Next, we sum up the extracted features with the aggregating function $f_2(\boldsymbol{o}_{h;t}^{(i'+1)}) = \sum_{k=1}^{p'_{i+1}} oW_{k,h;t}^{(i'+1)} + \sum_{r=1}^{p_{i+2}} oH_{r,h;t}^{(i'+1)}$, for $h = 1,\dots,p_{i+2}$, to generate the new output $x_{h;t}^{(i'+2)}$, $h = 1,\dots,p_{i+2}$. Similar to the analysis of Eq. (B.4), by incorporating Eq. (9-10) and Eq. (16-18), we have the following output:

$$\begin{aligned} x_{h;t}^{(i'+2)} =& \ f_a(f_2(\boldsymbol{o}_{h;t}^{(i'+1)})) \\ &= f_a(\sum_{k=1}^{p'_{i+1}} oW_{k,h;t}^{(i'+1)} + \sum_{r=1}^{p_{i+2}} oH_{r,h;t}^{(i'+1)}) \\ &= f_a(\sum_{k=1}^{p'_{i+1}} f_w(k) oW_{g(k),h;t}^{(i+1)} + \sum_{r=1}^{p_{i+2}} oH_{r,h;t}^{(i+1)}) \\ &= f_a(\sum_{k=1}^{p_{i+1}} oW_{k,h;t}^{(i+1)} + \sum_{r=1}^{p_{i+2}} oH_{r,h;t}^{(i+1)}) \\ &= x_{h;t}^{(i+2)}, h = 1,\dots,p_{i+2}. \end{aligned} \tag{B.9}$$

Therefore, we have the output features of layer $i'+1$

$$\boldsymbol{x}_t^{(i'+2)} = (x_{1,t}^{(i+2)}, x_{2,t}^{(i+2)}, \dots, x_{p_{i+2},t}^{(i+2)}). \tag{B.10}$$

From Eq. (B.10), the output feature vector $\boldsymbol{x}_t^{(i'+2)}$ of layer $i'+1$ is identical to that of layer $i+1$, i.e., $\boldsymbol{x}_t^{(i'+2)} = \boldsymbol{x}_t^{(i+2)}$, in line with the function-preserving transformation in Eq. (1), which indicates that the wider transformation for layer i is a function-preserving transformation. Hence, the proof of Proposition 1 is completed. □

Proof of Proposition 2: First, we consider the first stage processing for the *wider* layer i'. We apply the new student network parameters of $\boldsymbol{W}_f^{(i')} \in \mathbb{R}^{p_i \times p'_{i+1}}, \boldsymbol{W}_i^{(i')} \in \mathbb{R}^{p_i \times p'_{i+1}}$, $\boldsymbol{W}_o^{(i')} \in \mathbb{R}^{p_i \times p'_{i+1}}$, $\boldsymbol{W}_c^{(i')} \in \mathbb{R}^{p_i \times p'_{i+1}}$, $\boldsymbol{H}_f^{(i')} \in \mathbb{R}^{p'_{i+1} \times p'_{i+1}}$, $\boldsymbol{H}_i^{(i')} \in \mathbb{R}^{p'_{i+1} \times p'_{i+1}}$, $\boldsymbol{H}_o^{(i')} \in \mathbb{R}^{p'_{i+1} \times p'_{i+1}}$ and $\boldsymbol{H}_c^{(i')} \in \mathbb{R}^{p'_{i+1} \times p'_{i+1}}$ in Proposition 2 to Eq. (A.13), and the extracted features at time t are given by:

$$\begin{aligned} oW_{f;j,k;t}^{(i')} &= oW_{f;j,g(k);t}^{(i)}, j = 1,2,\dots,p_i, k = 1,\dots,p'_{i+1}, \\ oW_{i;j,k;t}^{(i')} &= oW_{i;j,g(k);t}^{(i)}, j = 1,2,\dots,p_i, k = 1,\dots,p'_{i+1}, \\ oW_{f;j,k;t}^{(o')} &= oW_{o;j,g(k);t}^{(i)}, j = 1,2,\dots,p_i, k = 1,\dots,p'_{i+1}, \\ oW_{c;j,k;t}^{(c')} &= oW_{c;j,g(k);t}^{(i)}, j = 1,2,\dots,p_i, k = 1,\dots,p'_{i+1}, \\ oH_{f;l,k;t}^{(i')} &= f_w(l) oH_{f;g(l),g(k);t}^{(i)}, l = 1,2,\dots,p'_{i+1}, k = 1,\dots,p'_{i+1}, \\ oH_{i;l,k;t}^{(i')} &= f_w(l) oH_{i;g(l),g(k);t}^{(i)}, l = 1,2,\dots,p'_{i+1}, k = 1,\dots,p'_{i+1}, \\ oH_{o;l,k;t}^{(i')} &= f_w(l) oH_{o;g(l),g(k);t}^{(i)}, l = 1,2,\dots,p'_{i+1}, k = 1,\dots,p'_{i+1}, \\ oH_{c;l,k;t}^{(i')} &= f_w(l) oH_{c;g(l),g(k);t}^{(i)}, l = 1,2,\dots,p'_{i+1}, k = 1,\dots,p'_{i+1}. \end{aligned} \tag{B.11}$$

Following Eq. (A.14), the feature extraction results of layer i' at time t, $\boldsymbol{o}_{k;t}^{(i')}$, are given by:

$$\begin{aligned} \boldsymbol{o}_{k;t}^{(i')} = (& oW_{f;1,k;t}^{(i')}, \dots oW_{f;p_i,k;t}^{(i')}, oW_{i;1,k;t}^{(i')}, \dots oW_{i;p_i,k;t}^{(i')}, \\ & oW_{c;1,k;t}^{(i')}, \dots oW_{c;p_i,k;t}^{(i')}, oW_{o;1,k;t}^{(i')}, \dots oW_{o;p_i,k;t}^{(i')}, \\ & oH_{f;1,k;t}^{(i')}, \dots, oH_{f;p_{i+1}k;t}^{(i')}, oH_{i;1,k;t}^{(i')}, \dots, oH_{i;p_{i+1},k;t}^{(i')}, \\ & oH_{c;1,k;t}^{(i')}, \dots, oH_{c;p_{i+1},k;t}^{(i')}, oH_{o;1,k;t}^{(i')}, \dots, oH_{o;p_{i+1},k;t}^{(i')},), k = 1,\dots,p'_{i+1}. \end{aligned} \tag{B.12}$$

Then, by Eq. (A.15) and Eq. (A.16) the new cell state $c_{k;t}^{(i')}$ is given by:

$$\begin{aligned}
c_{k;t}^{(i')} &= f_a(f_{2;c}(\boldsymbol{o}_{k;t}^{(i')})) \\
&= f_a(\sigma_g(\sum_{j=1}^{p_i} oW_{f;j,k;t}^{(i')} + \sum_{l=1}^{p'_{i+1}} oH_{f;l,k;t}^{(i')}) \cdot c_{k;t-1}^{(i')} \\
&\quad + \sigma_g(\sum_{j=1}^{p_i} oW_{i;j,k;t}^{(i')} + \sum_{l=1}^{p'_{i+1}} oH_{i;l,k;t}^{(i')}) \cdot \sigma_c(\sum_{j=1}^{p_i} oW_{c;j,k;t}^{(i')} + \sum_{l=1}^{p'_{i+1}} oH_{c;l,k;t}^{(i')})) \\
&= f_a(\sigma_g(\sum_{j=1}^{p_i} oW_{f;j,g(k);t}^{(i)} + \sum_{l=1}^{p'_{i+1}} f_w(l) oH_{f;g(l),g(k);t}^{(i)}) \cdot c_{g(k);t-1}^{(i)} \\
&\quad + \sigma_g(\sum_{j=1}^{p_i} oW_{i;j,g(k);t}^{(i)} + \sum_{l=1}^{p'_{i+1}} f_w(l) oH_{i;g(l),g(k);t}^{(i)}) \cdot \sigma_c(\sum_{j=1}^{p_i} oW_{c;j,g(k);t}^{(i)} + \sum_{l=1}^{p'_{i+1}} f_w(l) oH_{c;g(l),g(k);t}^{(i)})) \\
&= f_a(\sigma_g(\sum_{j=1}^{p_i} oW_{f;j,g(k);t}^{(i)} + \sum_{l=1}^{p_{i+1}} oH_{f;l,g(k);t}^{(i)}) \cdot c_{g(k);t-1}^{(i)} \\
&\quad + \sigma_g(\sum_{j=1}^{p_i} oW_{i;j,g(k);t}^{(i)} + \sum_{l=1}^{p_{i+1}} oH_{i;l,g(k);t}^{(i)}) \cdot \sigma_c(\sum_{j=1}^{p_i} oW_{c;j,g(k);t}^{(i)} + \sum_{l=1}^{p_{i+1}} oH_{c;l,g(k);t}^{(i)})) \\
&= c_{g(k);t}^{(i)} \quad , \quad k = 1, \ldots, p'_{i+1},
\end{aligned} \tag{B.13}$$

where the third equality holds by Eq. (B.11), the fourth equality holds by replication factor in Eq. (3), and the last equality holds by the definition of the aggregating function of the cell state in Eq. (A.15). Moreover, the second and third equality implies $c_{k;t-1}^{(i')} = c_{g(k);t-1}^{(i)}$, which can be proved by the following mathematical induction. Initially, we have $c_{k,0}^{(i)} = 0, k = 1, \ldots, p_{i+1}$ and $c_{k;0}^{(i')} = 0, k = 1, \ldots, p'_{i+1}$. When $t = 1$, we have $c_{k;t-1}^{(i')} = c_{g(k);t-1}^{(i)} = 0$. Suppose we have $c_{k;t-1}^{(i')} = c_{g(k);t-1}^{(i)}$, then we have $c_{k;t}^{(i')} = c_{g(k);t}^{(i)}$ according to Eq. (B.13). Hence, we can obtain a conclusion that $c_{k;t-1}^{(i')} = c_{g(k);t-1}^{(i)}$ holds for any t. The new output $x_{k;t}^{(i'+1)}$ is subsequently given by:

$$\begin{aligned}
x_{k;t}^{(i'+1)} &= f_a(f_{2;x}(\boldsymbol{o}_{k;t}^{(i')})) \\
&= f_a((\sum_{j=1}^{p_i} oW_{o;j,k;t}^{(i')} + \sum_{l=1}^{p'_{i+1}} oH_{o;l,k;t}^{(i')}) \cdot (c_{k;t}^{(i')} + 1)) \\
&= f_a((\sum_{j=1}^{p_i} oW_{o;j,g(k);t}^{(i)} + \sum_{l=1}^{p'_{i+1}} f_w(l) oH_{o;g(l),g(k);t}^{(i)}) \cdot (c_{g(k);t}^{(i)} + 1)) \\
&= f_a((\sum_{j=1}^{p_i} oW_{o;j,g(k);t}^{(i)} + \sum_{l=1}^{p_{i+1}} oH_{o;l,g(k);t}^{(i)}) \cdot (c_{g(k);t}^{(i)} + 1)) \\
&= x_{g(k);t}^{(i+1)} \quad , \quad k = 1, \ldots, p'_{i+1},
\end{aligned} \tag{B.14}$$

where the third equality holds by Eq. (B.11), the fourth equality holds by replication factor in Eq. (3), and the last equality holds by the aggregating function of the hidden state in Eq. (A.16). Hence, the output feature vector $\boldsymbol{x}_t^{(i'+1)}$ of layer i' are the input feature vector of layer $i'+1$. Following Eq. (A.18), $\boldsymbol{x}_t^{(i'+1)}$ can be represented as

$$\boldsymbol{x}_t^{(i'+1)} = (x_{g(1);t}^{(i+1)}, x_{g(2);t}^{(i+1)}, \ldots, x_{g(p'_{i+1});t}^{(i+1)}). \tag{B.15}$$

As the same as that in the wider transformer of RNN, $\boldsymbol{x}_t^{(i'+1)}$ is also a random mapping defined by g. Similar to the LSTM reformulation in wider layer i' as shown above, we apply the new student network parameters of $\boldsymbol{W}_f^{(i'+1)} \in \mathbb{R}^{p'_{i+1}\times p_{i+2}}$, $\boldsymbol{W}_i^{(i'+1)} \in \mathbb{R}^{p'_{i+1}\times p_{i+2}}$, $\boldsymbol{W}_o^{(i'+1)} \in \mathbb{R}^{p'_{i+1}\times p_{i+2}}$, $\boldsymbol{W}_c^{(i'+1)} \in \mathbb{R}^{p'_{i+1}\times p_{i+2}}$, $\boldsymbol{H}_f^{(i'+1)} \in \mathbb{R}^{p_{i+2}\times p_{i+2}}$, $\boldsymbol{H}_i^{(i'+1)} \in \mathbb{R}^{p_{i+2}\times p_{i+2}}$, $\boldsymbol{H}_o^{(i'+1)} \in \mathbb{R}^{p_{i+2}\times p_{i+2}}$, and $\boldsymbol{H}_c^{(i'+1)} \in \mathbb{R}^{p_{i+2}\times p_{i+2}}$ in Proposition 2 to Eq. (A.13), and acquire the extracted features for the next layer $i'+1$ as follows:

$$oW^{(i'+1)}_{\alpha;k,h;t} = f_w(k)oW^{(i+1)}_{\alpha;g(k),h;t}, \alpha \in \{f,i,o,c\}, k=1,2,\ldots,p'_{i+1}, h=1,\ldots,p_{i+2}, \tag{B.16}$$

$$oH^{(i'+1)}_{\alpha;r,h;t} = oH^{(i+1)}_{\alpha;r,h;t}, \alpha \in \{f,i,o,c\}, r=1,2,\ldots,p_{i+2}, h=1,\ldots,p_{i+2}. \tag{B.17}$$

As observed in Eq. (B.17), the feature extraction for outputs of layer $i'+1$ is the same as that of layer $i+1$ since the dimension is unchanged. Following Eq. (A.14), the feature extraction results of layer $i'+1$ at time t, $\boldsymbol{o}^{(i'+1)}_{h;t}$, are given by

$$\begin{aligned}\boldsymbol{o}^{(i'+1)}_{h;t} = (&oW^{(i'+1)}_{f;1,h;t}, \ldots oW^{(i'+1)}_{f;p'_{i+1},h;t}, oW^{(i'+1)}_{i;1,h;t}, \ldots oW^{(i'+1)}_{i;p'_{i+1},h;t},\\ &oW^{(i'+1)}_{c;1,h;t}, \ldots oW^{(i'+1)}_{c;p'_{i+1},h;t}, oW^{(i'+1)}_{o;1,h;t}, \ldots oW^{(i'+1)}_{o;p'_{i+1},h;t},\\ &oH^{(i'+1)}_{f;1,h;t}, \ldots, oH^{(i'+1)}_{f;p_{i+2},h;t}, oH^{(i'+1)}_{i;1,h;t}, \ldots, oH^{(i'+1)}_{i;p_{i+2},h;t},\\ &oH^{(i'+1)}_{c;1,h;t}, \ldots, oH^{(i'+1)}_{c;p_{i+2},h;t}, oH^{(i'+1)}_{o;1,h;t}, \ldots, oH^{(i'+1)}_{o;p_{i+2},h;t},), h=1,\ldots,p_{i+2}.\end{aligned} \tag{B.18}$$

Next, we aggregate the extracted features with the aggregating functions, Eq. (A.15) and (A.16), to generate the new cell state $c^{(i'+1)}_{h;t}$ and new output $x^{(i'+2)}_{h;t}$, $h=1,\ldots,p_{i+2}$. Similar to the analysis in Eq. (B.13) and Eq. (B.14), by incorporating Eq. (A.15-A.18) and Eq. (2-3), we have the following cell state:

$$\begin{aligned} c^{(i'+1)}_{h;t} &= f_a(f_{2;c}(\boldsymbol{o}^{(i'+1)}_{h;t}))\\ &= f_a(\sigma_g(\sum_{k=1}^{p'_{i+1}} oW^{(i'+1)}_{f;k,h;t} + \sum_{l=1}^{p_{i+2}} oH^{(i'+1)}_{f;r,h;t}) \cdot c^{(i'+1)}_{h;t-1}\\ &\quad + \sigma_g(\sum_{k=1}^{p'_{i+1}} oW^{(i'+1)}_{i;k,h;t} + \sum_{r=1}^{p_{i+2}} oH^{(i'+1)}_{i;r,h;t}) \cdot \sigma_c(\sum_{k=1}^{p'_{i+1}} oW^{(i'+1)}_{c;k,h;t} + \sum_{r=1}^{p_{i+2}} oH^{(i'+1)}_{c;r,h;t}))\\ &= f_a(\sigma_g(\sum_{k=1}^{p'_{i+1}} f_w(k)oW^{(i+1)}_{f;g(k),h;t} + \sum_{l=1}^{p_{i+2}} oH^{(i+1)}_{f;r,h;t}) \cdot c^{(i+1)}_{h;t-1}\\ &\quad + \sigma_g(\sum_{k=1}^{p'_{i+1}} f_w(k)oW^{(i+1)}_{i;g(k),h;t} + \sum_{r=1}^{p_{i+2}} oH^{(i+1)}_{i;r,h;t}) \cdot \sigma_c(\sum_{k=1}^{p'_{i+1}} f_w(k)oW^{(i+1)}_{c;g(k),h;t} + \sum_{r=1}^{p_{i+2}} oH^{(i+1)}_{c;r,h;t}))\\ &= f_a(\sigma_g(\sum_{k=1}^{p_{i+1}} oW^{(i+1)}_{f;k,h;t} + \sum_{l=1}^{p_{i+2}} oH^{(i+1)}_{f;r,h;t}) \cdot c^{(i+1)}_{h;t-1}\\ &\quad + \sigma_g(\sum_{k=1}^{p_{i+1}} oW^{(i+1)}_{i;k,h;t} + \sum_{r=1}^{p_{i+2}} oH^{(i+1)}_{i;r,h;t}) \cdot \sigma_c(\sum_{k=1}^{p_{i+1}} oW^{(i+1)}_{c;k,h;t} + \sum_{r=1}^{p_{i+2}} oH^{(i+1)}_{c;r,h;t}))\\ &= c^{(i+1)}_{h;t}, h=1,\ldots,p_{i+2}, \end{aligned} \tag{B.19}$$

and hidden state:

$$\begin{aligned} x_{h;t}^{(i'+2)} &= f_a(f_{2;x}(\boldsymbol{o}_{h;t}^{(i'+1)})) \\ &= f_a((\sum_{k=1}^{p'_{i+1}} oW_{o;k,h;t}^{(i'+1)} + \sum_{r=1}^{p_{i+2}} oH_{o;r,h;t}^{(i'+1)}) \cdot (c_{h;t}^{(i'+1)}+1)) \\ &= f_a((\sum_{k=1}^{p'_{i+1}} f_w(k) oW_{o;g(k),h;t}^{(i+1)} + \sum_{r=1}^{p_{i+2}} oH_{o;r,h;t}^{(i+1)}) \cdot (c_{h;t}^{(i+1)}+1)) \\ &= f_a((\sum_{k=1}^{p_{i+1}} oW_{o;k,h;t}^{(i)} + \sum_{r=1}^{p_{i+2}} oH_{o;r,h;t}^{(i)}) \cdot (c_{h;t}^{(i+1)}+1)) \\ &= x_{h;t}^{(i+2)}, h=1,\ldots,p_{i+2}. \end{aligned} \tag{B.20}$$

Therefore, we have the output feature vector of layer $i'+1$

$$\boldsymbol{x}_t^{(i'+2)} = (x_{1,t}^{(i+2)}, x_{2,t}^{(i+2)}, \ldots, x_{p_{i+2},t}^{(i+2)}). \tag{B.21}$$

From Eq. (B.21), the output feature vector $\boldsymbol{x}_t^{(i'+2)}$ of layer $i'+1$ is identical to that of layer $i+1$, i.e., $\boldsymbol{x}_t^{(i'+2)} = \boldsymbol{x}_t^{(i+2)}$, which indicates that the wider transformation for layer i is a function-preserving transformation. Hence, the proof of Proposition 2 is completed. □

Proof of Proposition 3: First, we consider the feature extraction for the deeper layer i''. We apply the new network parameters of $\boldsymbol{W}^{(i'')}$ and $\boldsymbol{H}^{(i'')}$ in Proposition 3 to Eq. (A.7) and (A.8), and we get the extracted features for inputs at time t and outputs at time $t+1$ as follows:

$$oW_{l,k;t}^{(i'')} = f_1(x_{l;t}^{(i'')}, W_{l,k}^{(i'')}) = \begin{cases} x_{l;t}^{(i'')} & l=k \\ 0 & l\neq k \end{cases}, \qquad l=1,2,\ldots,p_{i+1}, k=1,\ldots,p_{i+1}, \tag{B.22}$$

$$oH_{l,k;t}^{(i'')} = f_1(x_{l;t-1}^{(i''+1)}, H_{l,k}^{(i'')}) = 0, \qquad l=1,2,\ldots,p_{i+1}, k=1,\ldots,p_{i+1}. \tag{B.23}$$

Based on Eq. (A.9), the feature extraction results ($\boldsymbol{o}_{k;t}^{(i'')}$) of layer i'' at time t, are described as

$$\boldsymbol{o}_{k;t}^{(i'')} = (0,\ldots x_{k;t}^{(i'')},\ldots,0,0,\ldots,0), \qquad k=1,\ldots,p_{i+1}. \tag{B.24}$$

By evaluating the aggregating function $f_2(\boldsymbol{o}_{k;t}^{(i'')}) = \sum_{l=1}^{p_{i+1}} oW_{l,k;t}^{(i'')} + \sum_{l=1}^{p_{i+1}} oH_{l,k;t}^{(i'')} = x_{k;t}^{(i'')}$, for $k=1,\ldots,p_{i+1}$, the new output $x_{k;t}^{(i''+1)}$ is then obtained:

$$x_{k;t}^{(i''+1)} = f_a(f_2(\boldsymbol{o}_{k;t}^{(i'')})) = f_a(x_{k;t}^{(i'')}) = x_{k;t}^{(i'')}, k=1,\ldots,p_{i+1}, \tag{B.25}$$

Therefore, we have:

$$\boldsymbol{x}_t^{(i''+1)} = \boldsymbol{x}_t^{(i'')}. \tag{B.26}$$

As observed in Eq. (B.26), the input and output features are identical, verifying the function-preserving transformation for layer i in Eq. (1). Hence, the proof of Proposition 3 is completed. □

Proof of Proposition 4: First, we take the feature extraction for the deeper layer i'' into account. We leverage the new network parameters of $\boldsymbol{W}_f^{(i)} \in \mathbb{R}^{p_{i+1}\times p_{i+1}}, \boldsymbol{W}_i^{(i)} \in \mathbb{R}^{p_{i+1}\times p_{i+1}}$, $\boldsymbol{W}_o^{(i)} \in \mathbb{R}^{p_{i+1}\times p_{i+1}}$, $\boldsymbol{W}_c^{(i)} \in \mathbb{R}^{p_{i+1}\times p_{i+1}}$, $\boldsymbol{H}_f^{(i)} \in \mathbb{R}^{p_{i+1}\times p_{i+1}}$, $\boldsymbol{H}_i^{(i)} \in \mathbb{R}^{p_{i+1}\times p_{i+1}}$, $\boldsymbol{H}_o^{(i)} \in \mathbb{R}^{p_{i+1}\times p_{i+1}}$ and $\boldsymbol{H}_c^{(i)} \in \mathbb{R}^{p_{i+1}\times p_{i+1}}$ in Proposition 4 to Eq. (A.13) , and we obtain the extracted features for inputs at time t and outputs at time $t+1$ as follows:

$$oW_{o;l,k;t}^{(i'')} = \begin{cases} x_{l;t}^{(i'')} & l=k \\ 0 & l\neq k \end{cases}, \quad l=1,2,\ldots,p_{i+1}, k=1,\ldots,p_{i+1}, \tag{B.27}$$

$$oW_{\alpha;l,k;t}^{(i'')} = 0, \qquad \alpha\in\{f,i,c\}, l=1,2,\ldots,p_{i+1}, k=1,\ldots,p_{i+1}, \tag{B.28}$$

$$oH_{\alpha;l,k;t}^{(i'')} = 0, \qquad \alpha\in\{f,i,o,c\}, l=1,2,\ldots,p_{i+1}, k=1,\ldots,p_{i+1}. \tag{B.29}$$

By evaluating the aggregating functions, $f_{2;c}$ and $f_{2;x}$, we can obtain the new cell state:

$$\begin{aligned} c_{k;t}^{(i'')} &= f_{2;c}(\boldsymbol{o}_{k;t}^{(i)}) = \sigma_g(\sum_{j=1}^{p_{i+1}} oW_{f;j,k;t}^{(i)} + \sum_{l=1}^{p_{i+1}} oH_{f;l,k;t}^{(i)}) \cdot c_{k;t-1}^{(i)} \\ &+\sigma_g(\sum_{j=1}^{p_{i+1}} oW_{i;j,k;t}^{(i)} + \sum_{l=1}^{p_{i+1}} oH_{i;l,k;t}^{(i)}) \cdot \sigma_c(\sum_{j=1}^{p_{i+1}} oW_{c;j,k;t}^{(i)} + \sum_{l=1}^{p_{i+1}} oH_{c;l,k;t}^{(i)}) \\ &= \mathbf{0}, k = 1, \ldots, p_{i+1}. \end{aligned} \tag{B.30}$$

And the new hidden state $x_{k;t}^{(i''+1)}$ is:

$$\begin{aligned} x_{k;t}^{(i''+1)} &= f_a(f_{2;x}(\boldsymbol{o}_{k;t}^{(i'')})) \\ &= (\sum_{j=1}^{p_{i+1}} oW_{o;j,k;t}^{(i'')} + \sum_{l=1}^{p_{i+1}} oH_{o;l,k;t}^{(i'')}) \cdot (f_{2,c}(\boldsymbol{o}_{k;t}^{(i'')}) + 1) \\ &= x_{k;t}^{(i'')}, k = 1, \ldots, p_{i+1}. \end{aligned} \tag{B.31}$$

Therefore, we have:

$$\boldsymbol{x}_t^{(i''+1)} = \boldsymbol{x}_t^{(i'')}. \tag{B.32}$$

As seen in Eq. (B.32), the input and output features are the same, which verifies the function-preserving transformation for layer i in Eq. (1). Hence, the proof of Proposition 4 is completed. □

Appendix C: Pruning Formulations of FCN, RNN, CNN and LSTM

Consider a specific case of pruning a single FCN layer i with p_i input features and p_{i+1} output features. The FCN layer $i+1$ has p_{i+2} output features, then the parameters of the layers i and $i+1$ are $\boldsymbol{W}^{(i)} \in \mathbb{R}^{p_i \times p_{i+1}}$ and $\boldsymbol{W}^{(i+1)} \in \mathbb{R}^{p_{i+1} \times p_{i+2}}$, respectively. If we prune layer i to layer i' with p'_{i+1} output features ($p'_{i+1} < p_{i+1}$) according to the importance score $\boldsymbol{\lambda}^{(i+1)}$ where $\lambda_k^{(i+1)} = \sum \Lambda_{l,k}^{(i)} (l = 1, \cdots, p_i)$ and $\boldsymbol{\Lambda}^{(i)}$ is the importance score of $\boldsymbol{W}^{(i)}$, the original network parameters of layers i and $i+1$ are replaced by $\boldsymbol{W}^{(i')} \in \mathbb{R}^{p_i \times p'_{i+1}}$ and $\boldsymbol{W}^{(i'+1)} \in \mathbb{R}^{p'_{i+1} \times p_{i+2}}$, separately. The mask $\boldsymbol{m}^{(i+1)} = Top_v(\boldsymbol{\lambda}^{(i+1)})$ is a vector with p_{i+1} dimensions, and we can obtain the mapping function g satisfying the following conditions:

$$m_{g(k)}^{(i+1)} = 1, \ k = 1, 2, \ldots, p'_{i+1}, \tag{C.1}$$

$$|\{l | g(k) = g(l)\}| = 1, l = 1, 2, \ldots, p'_{i+1}, \ k = 1, 2, \ldots, p'_{i+1}. \tag{C.2}$$

Eq. (C.1) requires that g choose the neurons that are in the highest $v\%$ and Eq. (C.2) guarantees that each chosen neuron is distinct, which means that no neuron is overlooked and there are no duplicates. After pruning, we can obtain the updated parameters for this FCN layer as:

$$\begin{aligned} W_{j,k}^{(i')} &= W_{j,g(k)}^{(i)}, & j &= 1, 2, \ldots, p_i, k = 1, \ldots, p'_{i+1}, \\ W_{k,h}^{(i'+1)} &= W_{g(k),h}^{(i+1)}, & k &= 1, 2, \ldots, p'_{i+1}, h = 1, \ldots, p_{i+2}. \end{aligned}$$

Moreover, we consider pruning a single RNN layer i with p_i input features and p_{i+1} output features. The RNN layer $i+1$ has p_{i+2} output features; then the parameters of the layers i and $i+1$ are $\boldsymbol{W}^{(i)} \in \mathbb{R}^{p_i \times p_{i+1}}$, $\boldsymbol{H}^{(i)} \in \mathbb{R}^{p_{i+1} \times p_{i+1}}$ and $\boldsymbol{W}^{(i+1)} \in \mathbb{R}^{p_{i+1} \times p_{i+2}}$, $\boldsymbol{H}^{(i+1)} \in \mathbb{R}^{p_{i+2} \times p_{i+2}}$, respectively. If we prune layer i to layer i' with p'_{i+1} output features ($p'_{i+1} < p_{i+1}$) according to the importance score $\boldsymbol{\lambda}^{(i+1)}$, the original network parameters of layers i and $i+1$ are replaced by $\boldsymbol{W}^{(i')} \in \mathbb{R}^{p_i \times p'_{i+1}}$, $\boldsymbol{H}^{(i')} \in \mathbb{R}^{p'_{i+1} \times p'_{i+1}}$ and $\boldsymbol{W}^{(i'+1)} \in \mathbb{R}^{p'_{i+1} \times p_{i+2}}$, $\boldsymbol{H}^{(i'+1)} \in \mathbb{R}^{p_{i+2} \times p_{i+2}}$, separately. Here, $\lambda_k^{(i+1)} = \sum_l \Lambda w_{l,k}^{(i)} + \sum_h \Lambda h_{h,k}^{(i)} (l = 1, \cdots, p_i; h = 1, \cdots, p_{i+1})$ and $\boldsymbol{\Lambda w}^{(i)}$ is the importance score of $\boldsymbol{W}^{(i)}$, $\boldsymbol{\Lambda h}^{(i)}$ is the importance score of $\boldsymbol{H}^{(i)}$. The mask $\boldsymbol{m}^{(i+1)} = Top_v(\boldsymbol{\lambda}^{(i+1)})$ is a vector with p_{i+1} dimensions. After pruning, we can achieve the parameters for this pruned RNN layer as:

$$\begin{aligned} W_{j,k}^{(i')} &= W_{j,g(k)}^{(i)}, & j &= 1, 2, \ldots, p_i, k = 1, \ldots, p'_{i+1}, \\ H_{l,k}^{(i')} &= H_{g(l),g(k)}^{(i)}, & l &= 1, 2, \ldots, p'_{i+1}, k = 1, \ldots, p'_{i+1}, \\ W_{k,h}^{(i'+1)} &= W_{g(k),h}^{(i+1)}, & k &= 1, 2, \ldots, p'_{i+1}, h = 1, \ldots, p_{i+2}, \\ H_{r,h}^{(i'+1)} &= H_{r,h}^{(i+1)}, & r &= 1, 2, \ldots, p_{i+2}, h = 1, \ldots, p_{i+2}. \end{aligned}$$

Furthermore, we consider pruning a single one-dimensional CNN layer i with p_i input channels and p_{i+1} output channels. The CNN layer $i+1$ has p_{i+2} output channels. The parameters of the layers i and $i+1$ are then $\boldsymbol{W}^{(i)} \in \mathbb{R}^{p_i \times p_{i+1} \times d_k}$ and $\boldsymbol{W}^{(i+1)} \in \mathbb{R}^{p_{i+1} \times p_{i+2} \times d_k}$, respectively, where d_k is the kernel size of the one-dimensional CNN layer. If we prune layer i to layer i' with p'_{i+1} output channels ($p'_{i+1} < p_{i+1}$) according to the importance score $\boldsymbol{\lambda}^{(i+1)}$, the original network parameters of layers i and $i+1$ are replaced by $\boldsymbol{W}^{(i')} \in \mathbb{R}^{p_i \times p'_{i+1} \times d_k}$ and $\boldsymbol{W}^{(i'+1)} \in \mathbb{R}^{p'_{i+1} \times p_{i+2} \times d_k}$, separately. Therefore, we can obtain the prune mask $\boldsymbol{m}^{(i+1)} =$

$Top_v(\boldsymbol{\lambda}^{(i+1)})$ the mapping function g that satisfies the conditions in Eq. (C.1) and (C.2). After pruning, we can obtain the updated parameters for this CNN layer as:

$$\begin{aligned} W^{(i')}_{j,k,l} &= W^{(i)}_{j,g(k),l}, && j=1,2,\ldots,p_i, k=1,\ldots,p'_{i+1}, l=1,\ldots,d_k, \\ W^{(i'+1)}_{k,h,l} &= W^{(i+1)}_{g(k),h,l}, && k=1,2,\ldots,p'_{i+1}, h=1,\ldots,p_{i+2}, l=1,\ldots,d_k. \end{aligned}$$

Lastly, we consider pruning LSTM neurons. Assume LSTM layer i has p_i input and p_{i+1} output features, and LSTM layer $i+1$ has p_{i+1} input and p_{i+2} output features. Hence, the parameters of layer i are $\boldsymbol{W}^{(i)}_f \in \mathbb{R}^{p_i \times p_{i+1}}, \boldsymbol{W}^{(i)}_i \in \mathbb{R}^{p_i \times p_{i+1}}$, $\boldsymbol{W}^{(i)}_o \in \mathbb{R}^{p_i \times p_{i+1}}$, $\boldsymbol{W}^{(i)}_c \in \mathbb{R}^{p_i \times p_{i+1}}$, $\boldsymbol{H}^{(i)}_f \in \mathbb{R}^{p_{i+1} \times p_{i+1}}$, $\boldsymbol{H}^{(i)}_i \in \mathbb{R}^{p_{i+1} \times p_{i+1}}$, $\boldsymbol{H}^{(i)}_o \in \mathbb{R}^{p_{i+1} \times p_{i+1}}$, and $\boldsymbol{H}^{(i)}_c \in \mathbb{R}^{p_{i+1} \times p_{i+1}}$; moreover, the parameters of layer $i+1$ are $\boldsymbol{W}^{(i+1)}_f \in \mathbb{R}^{p_{i+1} \times p_{i+2}}, \boldsymbol{W}^{(i+1)}_i \in \mathbb{R}^{p_{i+1} \times p_{i+2}}$, $\boldsymbol{W}^{(i+1)}_o \in \mathbb{R}^{p_{i+1} \times p_{i+2}}$, $\boldsymbol{W}^{(i+1)}_c \in \mathbb{R}^{p_{i+1} \times p_{i+2}}$, $\boldsymbol{H}^{(i+1)}_f \in \mathbb{R}^{p_{i+2} \times p_{i+2}}$, $\boldsymbol{H}^{(i+1)}_i \in \mathbb{R}^{p_{i+2} \times p_{i+2}}$, $\boldsymbol{H}^{(i+1)}_o \in \mathbb{R}^{p_{i+2} \times p_{i+2}}$, and $\boldsymbol{H}^{(i+1)}_c \in \mathbb{R}^{p_{i+2} \times p_{i+2}}$. If we prune layer i to layer i' with p'_{i+1} output features ($p'_{i+1} < p_{i+1}$), the original network parameters of layers i are replaced by $\boldsymbol{W}^{(i')}_f \in \mathbb{R}^{p_i \times p'_{i+1}}, \boldsymbol{W}^{(i')}_i \in \mathbb{R}^{p_i \times p'_{i+1}}$, $\boldsymbol{W}^{(i')}_o \in \mathbb{R}^{p_i \times p'_{i+1}}$, $\boldsymbol{W}^{(i')}_c \in \mathbb{R}^{p_i \times p'_{i+1}}$, $\boldsymbol{H}^{(i')}_f \in \mathbb{R}^{p'_{i+1} \times p'_{i+1}}$, $\boldsymbol{H}^{(i')}_i \in \mathbb{R}^{p'_{i+1} \times p'_{i+1}}$, $\boldsymbol{H}^{(i')}_o \in \mathbb{R}^{p'_{i+1} \times p'_{i+1}}$, and $\boldsymbol{H}^{(i')}_c \in \mathbb{R}^{p'_{i+1} \times p'_{i+1}}$. In addition, the parameters of layer $i+1$ are replaced by $\boldsymbol{W}^{(i'+1)}_f \in \mathbb{R}^{p'_{i+1} \times p_{i+2}}, \boldsymbol{W}^{(i'+1)}_i \in \mathbb{R}^{p'_{i+1} \times p_{i+2}}$, $\boldsymbol{W}^{(i'+1)}_o \in \mathbb{R}^{p'_{i+1} \times p_{i+2}}$, $\boldsymbol{W}^{(i'+1)}_c \in \mathbb{R}^{p'_{i+1} \times p_{i+2}}$, $\boldsymbol{H}^{(i'+1)}_f \in \mathbb{R}^{p_{i+2} \times p_{i+2}}$, $\boldsymbol{H}^{(i'+1)}_i \in \mathbb{R}^{p_{i+2} \times p_{i+2}}$, $\boldsymbol{H}^{(i'+1)}_o \in \mathbb{R}^{p_{i+2} \times p_{i+2}}$, and $\boldsymbol{H}^{(i'+1)}_c \in \mathbb{R}^{p_{i+2} \times p_{i+2}}$. After calculating the importance score $\boldsymbol{\lambda}^{(i+1)}$ by automatic differentiation technique (Baydin et al. 2018), we can obtain the prune mask $\boldsymbol{m}^{(i+1)} = Top_v(\boldsymbol{\lambda}^{(i+1)})$ and the mapping function g. Consequently, we can achieve the updated parameters for this LSTM layer as:

$$\begin{aligned}
W^{(i')}_{f;j,k} &= W^{(i)}_{f;j,g(k)}, && j=1,2,\ldots,p_i, k=1,\ldots,p'_{i+1}, \\
W^{(i')}_{i;j,k} &= W^{(i)}_{i;j,g(k)}, && j=1,2,\ldots,p_i, k=1,\ldots,p'_{i+1}, \\
W^{(i')}_{o;j,k} &= W^{(i)}_{o;j,g(k)}, && j=1,2,\ldots,p_i, k=1,\ldots,p'_{i+1}, \\
W^{(i')}_{c;j,k} &= W^{(i)}_{c;j,g(k)}, && j=1,2,\ldots,p_i, k=1,\ldots,p'_{i+1}, \\
H^{(i')}_{f;l,k} &= f_w(l)H^{(i)}_{f;g(l),g(k)}, && l=1,2,\ldots,p'_{i+1}, k=1,\ldots,p'_{i+1}, \\
H^{(i')}_{i;l,k} &= f_w(l)H^{(i)}_{i;g(l),g(k)}, && l=1,2,\ldots,p'_{i+1}, k=1,\ldots,p'_{i+1}, \\
H^{(i')}_{o;l,k} &= f_w(l)H^{(i)}_{o;g(l),g(k)}, && l=1,2,\ldots,p'_{i+1}, k=1,\ldots,p'_{i+1}, \\
H^{(i')}_{c;l,k} &= f_w(l)H^{(i)}_{c;g(l),g(k)}, && l=1,2,\ldots,p'_{i+1}, k=1,\ldots,p'_{i+1}, \\
W^{(i'+1)}_{f;k,h} &= f_w(k)W^{(i+1)}_{f;g(k),h}, && k=1,2,\ldots,p'_{i+1}, h=1,\ldots,p_{i+2}, \\
W^{(i'+1)}_{i;k,h} &= f_w(k)W^{(i+1)}_{i;g(k),h}, && k=1,2,\ldots,p'_{i+1}, h=1,\ldots,p_{i+2}, \\
W^{(i'+1)}_{o;k,h} &= f_w(k)W^{(i+1)}_{o;g(k),h}, && k=1,2,\ldots,p'_{i+1}, h=1,\ldots,p_{i+2}, \\
W^{(i'+1)}_{c;k,h} &= f_w(k)W^{(i+1)}_{c;g(k),h}, && k=1,2,\ldots,p'_{i+1}, h=1,\ldots,p_{i+2}, \\
H^{(i'+1)}_{f;r,h} &= H^{(i+1)}_{f;r,h}, && r=1,2,\ldots,p_{i+2}, h=1,\ldots,p_{i+2}, \\
H^{(i'+1)}_{i;r,h} &= H^{(i+1)}_{i;r,h}, && r=1,2,\ldots,p_{i+2}, h=1,\ldots,p_{i+2}, \\
H^{(i'+1)}_{o;r,h} &= H^{(i+1)}_{o;r,h}, && r=1,2,\ldots,p_{i+2}, h=1,\ldots,p_{i+2}, \\
H^{(i'+1)}_{c;r,h} &= H^{(i+1)}_{c;r,h}, && r=1,2,\ldots,p_{i+2}, h=1,\ldots,p_{i+2}.
\end{aligned}$$

Appendix D: Operation of ACER Algorithm

Compared to the REINFORCE algorithm used in Cai et al. (2018), we make use of the ACER algorithm (Wang et al. 2017) to train the three RL agents. As presented in Section 3.2, the goal of the RL agents in this study is to maximize the discounted reward over transformation steps. For an agent following policy π, we use the standard definitions of the state-action and state only value function:

$$Q^{\pi}(s_u, a_u) = E_{s_{u+1:\infty}, a_{u+1:\infty}}(R_u | s_u, a_u), \tag{D.1}$$

$$V^{\pi}(s_u) = E_{a_u}(Q^{\pi}(s_u, a_u) | s_u), \tag{D.2}$$

where the expectations are with respect to s_u and a_u generated by the policy π, and $s_{u+1:\infty}$ indicates a state trajectory starting at time $u+1$. In ACER, policy π is the target policy. ACER also uses a behavior policy μ. Generally, the actions made by behavior policy is used to update the target policy. ACER estimates $Q^{\pi}(s_u, a_u)$ using Retrace (Munos et al. 2016). Given a trajectory generated under the behavior policy μ, the Retrace estimator can be written in a recursive manner:

$$Q^{ret}(s_u, a_u) = r_u + \gamma \bar{\rho}_{u+1} \left[Q^{ret}(s_{u+1}, a_{u+1}) - Q(s_{u+1}, a_{u+1}) \right] + \gamma V(s_{u+1}), \tag{D.3}$$

where $\rho_u = \frac{\pi(a_u|s_u)}{\mu(a_u|s_u)}$ denotes the importance weight, $\bar{\rho}_u = \min(c, \rho_u)$ is the truncated importance weight, c is a constant, Q is the current value estimate of Q^{π}, and $V(s) = E_{a\sim\pi}(Q(s,a))$. To compute Q, we adopt the proposed actor networks in Section 3.3 with "two heads" that output the estimate $Q_{\theta}(s_u, a_u)$ as well as the policy $\pi_{\theta}(a_u|s_u)$. The estimate $V_{\theta}(s_u)$ can be obtained by taking the expectation of $Q_{\theta}(s_u, a_u)$ under π_{θ}.

Given the sampling trajectories $\{s_0, a_0, r_0, \mu(\cdot|s_0), \ldots, s_u, a_u, r_u, \mu(\cdot|s_u)\}$ generated from the behavior policy ($\mu(\cdot|s_i)$ are the policy vectors, $i \in 1, 2, \ldots, u$), the actor network can be updated using the off-policy ACER gradient:

$$\begin{aligned} \hat{g_u}^{acer} =& \bar{\rho}_u \nabla_{\theta} \log \pi_{\theta}(a_u|s_u)[Q^{ret}(s_u, a_u) - V_{\theta}(s_u)] \\ &+ E_{a\sim\pi}\left([\frac{\rho_u(a) - c}{\rho_u(a)}]_+ \nabla_{\theta} \log \pi_{\theta}(a|s_u)[Q_{\theta}(s_u, a) - V_{\theta}(s_u)] \right), \end{aligned} \tag{D.4}$$

where $\hat{g_u}^{acer}$ is the ACER gradient, $[x]_+ = x$ if $x > 0$ and $[x]_+ = 0$ otherwise.

The critic network uses a mean squared error loss and updates its parameters by the standard gradient:

$$\hat{g_u}^{critic} = (Q^{ret}(s_u, a_u) - Q_{\theta}(s_u, a_u)) \nabla_{\theta} Q_{\theta}(s_u, a_u). \tag{D.5}$$

For more details about ACER algorithm, please refer to Wang et al. (2017).

Appendix E: Operation of RL in IAAS Framework

The heuristic screening algorithm in Algorithm 1 involves three other algorithms related to RL. In the following, we first present the Algorithm 1 to show how RL is involved in IAAS and then introduce the operation of RL in the other three algorithms. In particular, Algorithm 2 details the process of sampling a new network architecture by selector actor, wider actor, and deeper actor. It also adds a step information to the trajectory. Note that the reward signal is initiated to zero since we can only evaluate the reward signal after some training in the new network structure. Subsequently, the succeeding Algorithm 3 supplements the vacancy left in the Algorithm 2. After one episode of network search, we update the three RL agents using Algorithm 4. We construct a trajectory training set T_{train} with maximum capacity N_{Tmax} ($N_{Tmax} = 100$ in our implementation). After that, we update the agents by the training set using the ACER algorithm (Wang et al. 2017) described in Appendix D.

Algorithm 1: Heuristic Screening Algorithm

```
Initialization: Network pool set $Q$; Net pool capacity $C_Q$; Number of networks in net pool $S_Q$; Number of
    random generated structures $M$; Maximum number of search episodes $N_{max}$; Historical RL trajectories $T$; RL
    trajectories of networks in pool $T_p$.
 1: Initialize network pool set $Q$;
 2: Train and test networks in $Q$;
 3: while stopping criteria are not met (total episodes$< N_{max}$ in our case) do
 4:     for each network $q_i \in Q$ do
 5:         $q_i{}' =$ SAMPLE_NEW_NETWORK($q_i$,$T$,$T_p$);
 6:         Add $q_i{}'$ to $Q$;
 7:     end for
 8:     Collect the maximum layer number $\widetilde{A}$ and maximum unit number in one layer $\widetilde{B}$ from $Q$;
 9:     Generate $M$ network(s) constrained by $\widetilde{A}$ and $\widetilde{B}$, then add to $Q$;
10:     Train networks in $Q$;
11:     for each network $q_i \in Q$ do
12:         $\widetilde{M_i} =$ RMSE evaluation result of $q_i$
13:         RECORD_REWARD($T_p, \widetilde{M_i}, q_i$)
14:     end for
15:     Sort networks in $Q$ by their performance;
16:     if $S_Q > C_Q$ then
17:         Update $Q$ by dropping the last $S_Q - C_Q$ networks ;
18:         Move trajectories of dropped networks from $T_p$ to $T$
19:     end if
20:     UPDATE_RL($T$,$T_p$);
21: end while
22: Identify the best network $q^*$ in $Q$;
Output: Search result best network $q^*$.
```

Algorithm 2: SAMPLE_NEW_NETWORKS

```
Initialization: Original network structure $q_i$; RL trajectories of networks in the netpool $T_p$; Selector Actor $A_s$;
    Wider Actor $A_w$; Deeper Actor $A_d$.
 1: Get structure state $s_{q_i}$ from $q_i$;
 2: Get decision polices $\mu_{q_i,s}(\cdot|s_{q_i}) = A_s(s_{q_i}), \mu_{q_i,w}(\cdot|s_{q_i}) = A_w(s_{q_i}), \mu_{q_i,d}(\cdot|s_{q_i}) = A_d(s_{q_i})$
 3: Sample actions $a_{q_i,s}$ from polices $\mu_{q_i,s}(\cdot|s_{q_i})$
 4: if $a_{q_i,s}$ is unchanged then
 5:     $q_i \rightarrow {q_i}'$ ;
 6: else if $a_{q_i,s}$ is prune then
 7:     Pruning $q_i$ with the movement pruning algorithm $\rightarrow {q_i}'$;
 8: else if $a_{q_i,s}$ is wider then
 9:     Sample actions $a_{q_i,w}$ from polices $\mu_{q_i,w}(\cdot|s_{q_i})$
10:     Widening $q_i$ according to $a_{q_i,w} \rightarrow {q_i}'$;
11: else
12:     Sample actions $a_{q_i,d}$ from polices $\mu_{q_i,d}(\cdot|s_{q_i})$
13:     Deepening $q_i$ according to $a_{q_i,d} \rightarrow {q_i}'$;
14: end if
15: Set reward $r = 0$ as a placeholder for future update;
16: Record actions $a_{q_i} = (a_{q_i,s}, a_{q_i,w}, a_{q_i,d})$;
17: Record behavior polices $\mu_{q_i}(\cdot|s_{q_i}) = (\mu_{q_i,s}(\cdot|s_{q_i}), \mu_{q_i,w}(\cdot|s_{q_i}), \mu_{q_i,d}(\cdot|s_{q_i}))$;
18: Add new RL record$\{s_{q_i}, a_{q_i}, r, \mu_{q_i}(\cdot|s_{q_i})\}$ to $T_p$
Output: Transformation result ${q_i}'$.
```

Algorithm 3: RECORD_REWARD

```
Initialization: Original network structure $q_i$; RMSE evaluation result $\widetilde{M_i}$; RL trajectories of networks in netpool
    $T_p$;.
 1: Find unfinished record $\{s_{q_i}, a_{q_i}, r, \mu_{q_i}(\cdot|s_{q_i})\}$ of $q_i$ from $T_p$;
 2: $r_{q_i} = \frac{1}{\widetilde{M_i}}$
 3: Update record in $T_p$ by $\{s_{q_i}, a_{q_i}, r_{q_i}, \mu_{q_i}(\cdot|s_{q_i})\}$
```

Algorithm 4: RL_UPDATE

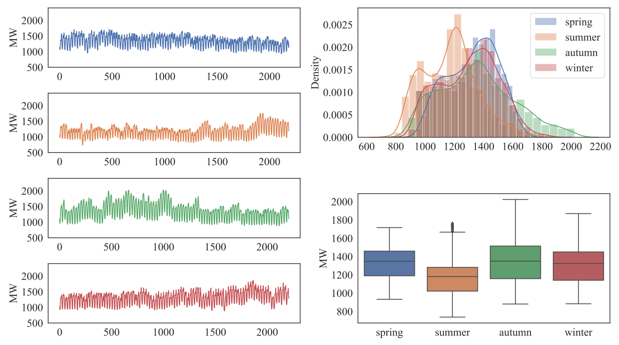
```
Initialization: Historical RL trajectories $T$; RL trajectories of networks in the net pool $T_p$; Selector Actor $A_s$;
    Wider Actor $A_w$; Deeper Actor $A_d$; Parameters $\theta_s, \theta_w, \theta_d$ of $A_s, A_w$ and $A_d$; Max training trajectory number
    $N_{Tmax}$.
 1: $\theta = \{\theta_s, \theta_w, \theta_d\}$
 2: $T_{train} = T_p$
 3: if $size(T_{train}) < N_{Tmax}$ then
 4:     Sample max$(N_{Tmax} - size(T_{train}), size(T))$ trajectories and add it to $T_{train}$
 5: end if
 6: for Trajectory $t \in T_{train}$ do
 7:     Calculate ACER gradient $\hat{g_t}^{acer}$ and Critic gradient $\hat{g_u}^{critic}$ using Eq. (D.4) and Eq. (D.5);
 8:     Update $\theta$ by ACER gradient $\hat{g_t}^{acer}$ and Critic gradient $\hat{g_u}^{critic}$;
 9: end for
```

Appendix F: Detailed Data Description

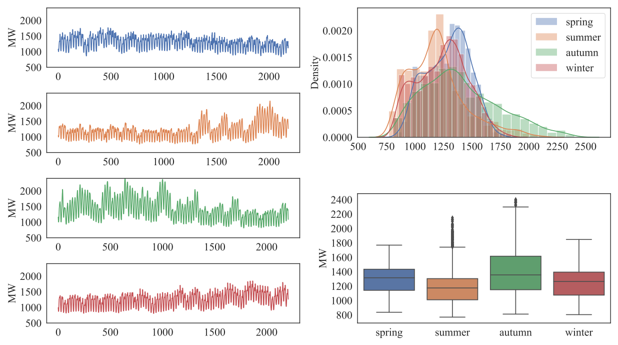
In each electricity load subset, the time step of load data is set to an hour, thereby leading to 24 data points per day, and thus approximately 2160 data points per season. Following Jalali et al. (2021), we split each independent subset into a training dataset with 75% data points and a test dataset with 25% data points. Moreover, the data in the training dataset is chronically earlier than that in the test dataset. When forecasting the data points in the load demand subsets, apart from using historical load demand data, we incorporate their corresponding time information as inputs, namely $\widetilde{h}^{th}$ hour of a day and $\widetilde{w}^{th}$ day of a week, where $\widetilde{h}$ and $\widetilde{w}$ are restricted to ranges of $[1,24]$ and $[1,7]$, respectively. For instance, if the load demand is 50 MW at 8:00 am on a Sunday, we describe such an input vector as $(50,8,7)'$. To improve forecasting accuracy, we use one-week data with the size of 168×3 as one input for the forecasting model to predict the 24h-ahead load demand point.

In each wind power subset, the time step of wind power data is set as an hour. Similar to the settings in Zhang et al. (2020), the last five-day data of each season are treated as the test dataset, and the remaining data of that season are the training dataset. As for the forecast of data points in the wind power subsets, numerical weather prediction (NWP) data are widely utilized following the day-ahead wind power literature (see Zhang et al. 2020, Chen et al. 2023). We can notice that the NWP data contains five dimensions, namely, humidity, wind speed, wind direction, temperature and air pressure, which are weather simulation data generated from NWP computer models of the atmosphere and oceans (Rabier 2005). Consistent with Zhang et al. (2020), we use the sine and cosine values of wind direction to signify the real wind direction. Moreover, we use the historical wind speed, wind power data, and hourly timestamp information in a day as additional input variables. Therefore, we have an input vector with nine dimensions. We use three-day data with a size of 72×9 as one input for the forecasting model to predict the 24h-ahead wind power point. Unlike the load input data, NWP data at the forecasting point is known in advance. For instance, if we are at 8 am on Jun 9 and forecast the wind power generation at 8 am on Jun 10, we can use the NWP data from 9 am on Jun 8 to 8 am on Jun 10 as input.

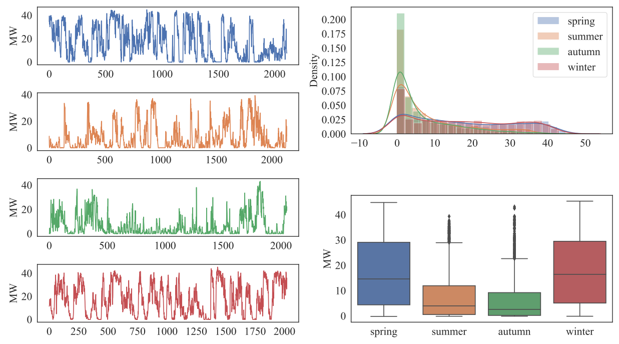
We plot the data value, empirical density, and boxplot by seasons in each sub-figure of Fig. A.3. We can observe that the load data is quite different from wind power data; however, both the load data and wind power data have seasonal patterns. For instance, the ME-spring load data has similar patterns to the NH-spring load data, and WF1-winter wind power generation data is also akin to WF2-winter wind power generation data. However, despite these similarities, they also have some differences from the further observations. We select these datasets for the numerical experiments for the following reasons. First, two types of electricity data, i.e., load and wind power data, are utilized to demonstrate the generalizations of the proposed IAAS framework. Second, the two identical types of data within some similarities are leveraged to display the stability of this framework.



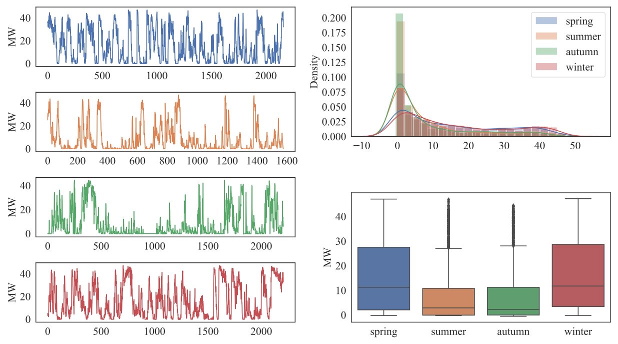
(a) ME load dataset



(b) NH load dataset



(c) WF1 wind power dataset



(d) WF2 wind power dataset

Figure A.3 Data description of four cases

Appendix G: Experimental Settings

Table A.1 Experimental settings for the baselines and IAAS framework

Model	Hyperparameter	Value
IAAS	Activation function	ReLU
	Batch Size	256
	Optimizer	Adam
	Epoch number of each episode	50
	Max. search episodes	200
SNAS	Activation function	ReLU
	Batch size	[64-1024], step size 64
	Hidden channels	32
	Layer per cell	[1,2,3,4]
	Search epoch	[50-100] step size 10
	Search iteration per epoch	[64-256] step size 64
	Training epoch	500
DAIN	Activation function	ReLU
	Mean learning rate	[1e-5,1e-4,1e-3,1e-2]
	Gate learning rate	[1e-5,1e-4,1e-3,1e-2]
	Scale learning rate	[1e-5,1e-4,1e-3,1e-2]
CNN+LSTM CNN LSTM ResNet ResNetPlus	Size of each layer	[4-128], step size 4
	Number of layers	[2-10], step size 1
	Activation function	ReLU, SELU, LeakyReLU
	Batch size	[64-1024], step size 64
	Learning rate	[0.001-0.1] , step size 0.001
	Dropout rate	[0.2-0.8] , step size 0.1
SVR	Kernel	RBF, Sigmoid , Poly
	Polynomial order	[1-12] , step size 1
	Regularization	1,10,100,1000
	epsilon	[0.1-1] , step size 0.1
RF	Split measure function	squared, absolute, Poisson
	Number of trees	[16-128] , step size 4
RR	penalty alpha	[0-4] , step size 0.1

The experimental settings, including all optimized hyperparameters for the baseline models (or methods) and our proposed IAAS framework, are listed in Table A.1. It should be mentioned that for the CNN, LSTM, and CNN+LSTM, we use the grid search algorithm to optimize the number of layers, size of each layer, activation functions, batch size, learning rate (LR), and dropout rate. Note that we make use of three activation functions, namely ReLU, scaled exponential linear units (SELU), and LeakyReLU, as the optimization candidates. For ResNet and ResNetPlus, we first examine their performance based on the reference article. Then, to obtain their best performances, we also utilize the grid search algorithm to optimize their hyperparameters, such as the number of layers and LR. For DAIN, we use the multilayer perceptron (MLP) to build the forecasting model since MLP has achieved better forecasting accuracy than RNN in Passalis et al. (2019). We follow the hyperparameter settings from Passalis et al. (2019) to conduct the experiments. And then, we make use of the trial-and-error simulation method to optimize the mean LR, gate LR, and scale LR to achieve a better result. As to the SNAS, we follow Chen et al. (2021) to set the hyperparameters since it is a NAS method to develop the time-series model, and these hyperparameter settings have helped SNAS obtain a better time-series forecasting accuracy on various datasets. Moreover, we also utilize the trial-and-error simulation method to optimize the hyperparameters, such as batch size, the number of layers per cell, the number of search iterations per epoch, and the number of search epochs. Note that since both Passalis et al. (2019) and Chen et al. (2021) only leveraged the ReLU activation function, we follow this setting in these two baseline models. In terms of SVR, RF, and RR, we use the grid search algorithm to optimize the hyperparameters as shown in Table A.1 as well.

Appendix H: Sensitivity Analysis Regarding the Number of Search Episodes

Considering the importance of the number of search episodes, we perform a sensitivity analysis to investigate its effect on model performance in this section. Specifically, we use five different numbers of episodes, namely 40, 80, 120, 160, 200, 240 and 280 for the experiments with the usage of ME load and WF1 datasets. We present the results in Table A.2 and bold the first best value from left to right in each row. Observing all eight cases, the RMSE values with more episodes are never larger than those with fewer episodes. This is because of the heuristic screening algorithm, which always preserves the network structures with good performance and kick off the bad ones in the net pool. The RMSLE values in all four load cases have the same changing pattern as their corresponding RMSE values. As to the MAE values, they have some differences. For example, in the case of ME-spring, when using 40 episodes and 160 episodes, the MAE values are 48.366 and 48.879, respectively. This means that the increase of the number of search episodes leads to the increase of the MAE value. The reason for this is that we use the RMSE as the loss function, which directly reflects the change in the RMSE forecasting accuracy but not the MAE. Note that the calculation of MAE is different from that of RMSE; hence, we regard this phenomenon as normal.

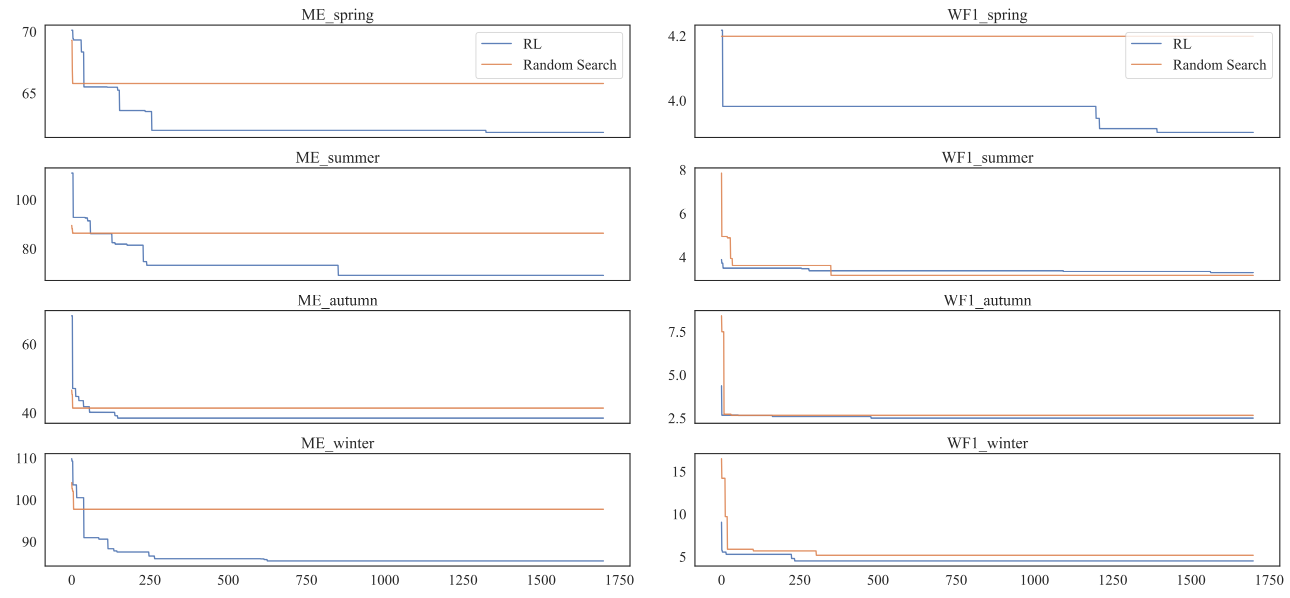
Table A.2 Sensitivity analysis regarding the number of search episodes

Case	Metrics	40 episodes	80 episodes	120 episodes	160 episodes	200 episodes	240 episodes	280 episodes
ME-spring	RMSE	61.978	61.978	61.978	**61.807**	61.807	61.807	61.807
	MAE	**48.366**	48.366	48.366	48.879	48.879	48.879	48.879
	RMSLE	0.05	0.05	0.05	**0.049**	0.049	0.049	0.049
ME-summer	RMSE	73.258	73.258	**69.154**	69.154	69.154	69.154	69.154
	MAE	**57.29**	57.29	58.187	58.187	58.187	58.187	58.187
	RMSLE	0.054	0.054	**0.052**	0.052	0.052	0.052	0.052
ME-autumn	RMSE	**38.483**	38.483	38.483	38.483	38.483	38.483	38.483
	MAE	**30.526**	30.526	30.526	30.526	30.526	30.526	30.526
	RMSLE	**0.031**	0.031	0.031	0.031	0.031	0.031	0.031
ME-winter	RMSE	85.92	**85.386**	85.386	85.386	85.386	85.386	85.386
	MAE	**66.123**	68.773	68.773	68.773	68.773	68.773	68.773
	RMSLE	0.064	**0.062**	0.062	0.062	0.062	0.062	0.062
WF1-spring	RMSE	3.982	3.982	3.982	**3.901**	3.901	3.901	3.901
	MAE	3.054	3.054	3.054	**2.996**	2.996	2.996	2.996
WF1-summer	RMSE	3.375	3.375	3.375	3.348	**3.29**	3.29	3.29
	MAE	**2.31**	2.31	2.31	2.474	2.614	2.614	2.614
WF1-autumn	RMSE	2.587	2.498	2.498	2.498	**2.237**	2.237	2.237
	MAE	1.769	1.91	1.91	1.91	**1.633**	1.633	1.633
WF1-winter	RMSE	**4.5**	4.5	4.5	4.5	4.5	4.5	4.5
	MAE	**3.427**	3.427	3.427	3.427	3.427	3.427	3.427

All in all, we have the following five findings: (i) when using 200 episodes, we can obtain the best RMSE accuracy in all cases and best RMSLE accuracy in all load cases; (ii) in three out of eight cases, we obtain the best MAE accuracy when using 200 episodes; (iii) the forecasting accuracies using 240 episodes and 280 episodes are the same as those using 200 episodes; (iv) in most cases, the forecasting accuracies between using 200 episodes and others do not show large differences; (v) in the case of WF1-autumn, the best forecasting accuracy is from using 200 episodes and is better than that from using 160 episodes with improvement of 10.4% (RMSE) and 14.5% (MAE). Therefore, we can conclude that the determination of using 200 episodes in this study is reasonable. Others may consider using 40 episodes if they pay more attention to the computational time.

Appendix I: Efficiency Performance Examination of the Selected RL Algorithm

In this study, we utilize RL to implement the intelligent control for the network transformation. To demonstrate the efficiency of using RL in the IAAS framework, we follow Cai et al. (2018) and make use of the random search algorithm as a benchmark. We randomly select the ME load dataset and WF1 dataset from the four datasets to demonstrate the forecasting performance of the selected RL and random search algorithms. For the random search operation, we use the same pool size and randomly generate network structures at each search episode instead of using RL-guided network transformation operation. To manage the networks in the pool, we utilize the proposed heuristic screening algorithm so that we can iterativley search a neural architecture with good performance after completing total search episodes. In this experiment, we set each algorithm to iteratively select 600 network structures. Fig. A.4 shows the change of the best forecasting performance along with the increase number of the searched network structures. As seen, after searching a certain number of network structures, both of these two algorithms can achieve the best network structure among the 600 ones in each case. However, the RL algorithm can efficiently find the network structure with better performance in seven out of eight cases. Although the network performance from RL is lower than that from random search in WF1_autumn case, their performances are very close as shown in Fig. A.4(b).

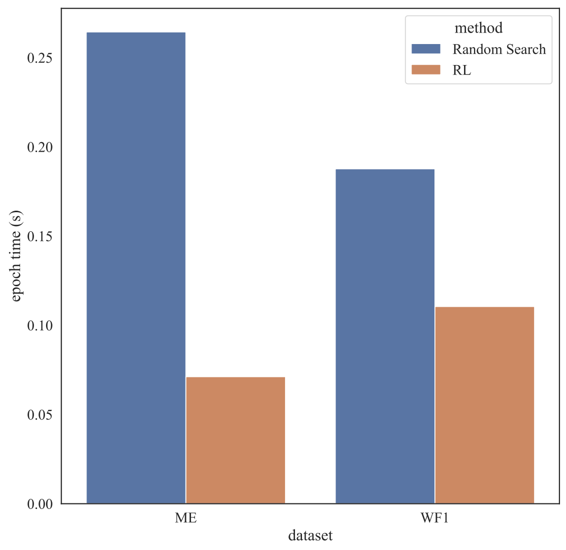


(a) Load case (b) Wind power case

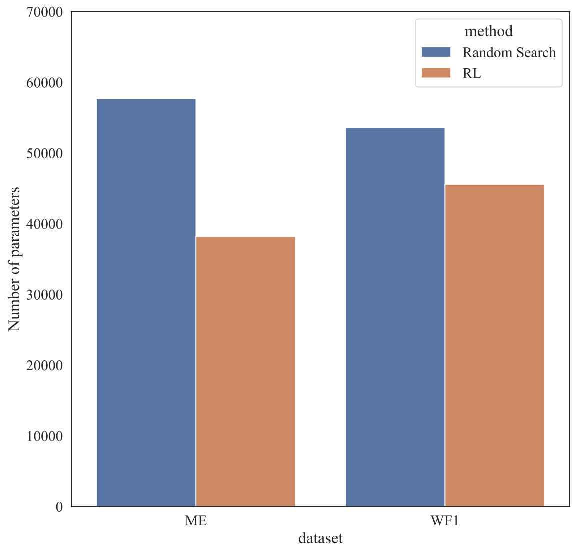
Figure A.4 Top RMSE forecasting accuracy in the search process.

Moreover, we use the average training time of the generated networks over episodes, and the average number of parameters over the generated networks to showcase the performance of these two algorithms. As seen in Fig. A.5(a), using RL-guided network transformation reduces the average training time when comparing with random search. As seen in Fig. A.5(b), RL tends to generate network with fewer parameters.

From these three perspectives, we can conclude the selected RL is more suitable to the IAAS framework than the random search algorithm.



(a)



(b)

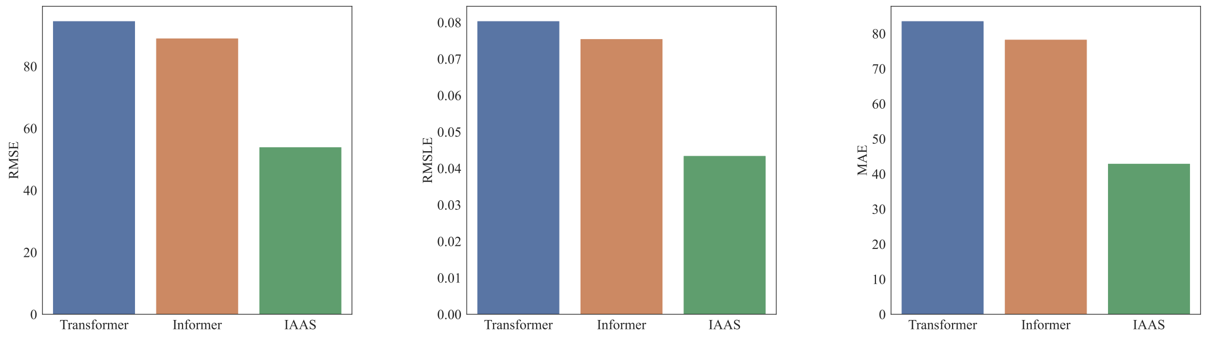
Figure A.5 Performance comparison between the selected RL and random search algorithm: (a) average training time per epoch and (b) average parameter number of each searched model.

Appendix J: Seq2Seq Forecasting Accuracy Comparison

In Section 5.1, we segment the dataset quarterly, and use the small portion of the data as the testing dataset. However, in practice, the electricity company always make the forecasting for the next several days at each time step. Therefore, the rolling-window technique is usually adopted for such a Seq2Seq forecasting task. Considering the excellent performance of transformer models in natural language processing area, we conduct a further experiment between the IAAS and the transformer models under the Seq2Seq setting. Specifically, we select two well-known transformer models, Transformer (Vaswani et al. 2017) and Informer (Zhou et al. 2021) as benchmarks. Following the above experiments, we also use RMSE, MAE and RMSLE as error measurements for load forecasting accuracy evaluation, and use RMSE and MAE for wind power forecasting accuracy evaluation. In this experiment, we randomly select ME load dataset and WF1 dataset from Section 4, and conduct one-day ahead forecasting and one-week ahead forecasting, respectively.

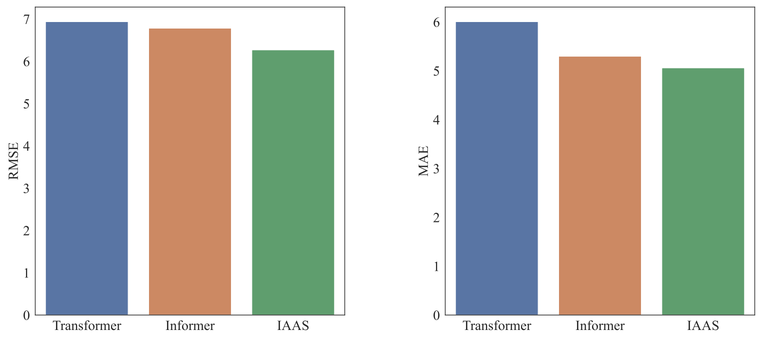
Intuitively, the one-day ahead forecasting experiment is designed to use the hourly historical data in past days to predict the data in the future one day (24 time steps). The input sequence sizes for the load and wind data are 168×3 (seven days) and 72×9 (three days), respectively. The output sequence sizes for these two cases are both 24×1, which means we forecast the load or wind power in the next day. For example, on Jan 1, we use the IAAS framework to search a Seq2Seq model to forecast the load or wind power at Jan 2. After that, we use this Seq2Seq model at Jan 2 to forecast them on Jan 3 with the past 7 days' real data including that of Jan 1 as input. Following this way, we will use this model to forecast the load or wind power at Jan 4-6. In total, we make use of this Seq2Seq model five times for one-day ahead forecasting. Considering the correlation of weather in consecutive days, we update this model at Jan 6. We dynamically update the Seq2Seq model every five days, aiming to provide a quality forecasting.

We use three-month data for training the Seq2Seq model and the rest nine-month data for testing. The rolling window scheme is used to dynamically update the training and testing data for the one-day ahead forecasting case. To reuse the knowledge learned in the previous window, we do not train the Transformer and Informer models from the scratch. Instead, we train them based on their optimized parameters from last window. For IAAS, we keep the initial settings same for the RL agents and net pool to search the neural structures for each rolling-window process. We design the Transformer and Informer structures through the trial-and-error simulation method, and the best structures of these two models both contain two encoder layers and two decoder layers with 512 model dimensions and 8 heads. Following the hyperparameter settings from Informer, which is designed for the time-series forecasting task, we train these two models for 1000 epochs using Adam optimizer, ReLU activation function, and RMSE loss function. Moreover, we use the grid search algorithm to determine the batch size from [128, 256, 512] and the weight decay rate from [0.1, 0.01, 0.001, 0.0001, 0.00001]. The overall testing forecasting accuracy results are presented in Figs. A.6 and A.7. As seen, in the load forecasting case, the IAAS model performs much better than Transformer and Informer regarding the three evaluation metrics; in the wind power forecasting case, the IAAS model has slightly better performance than Transformer and Informer models. All in all, the IAAS framework is likely to generate better electricity time-series forecasting models than the existing transformer models under the Seq2Seq setting. In other words, the IAAS framework is better for the practical applications in power system.



(a) RMSE (b) RMSLE (C) MAE

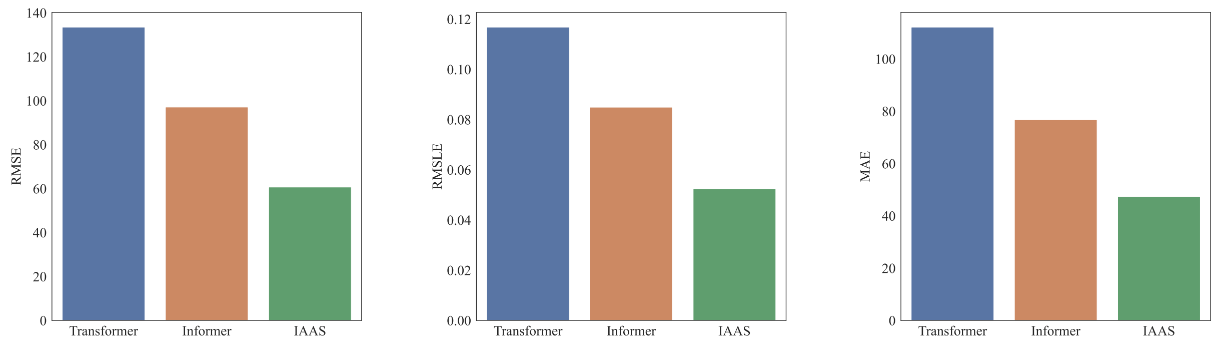
Figure A.6 One-day ahead load forecasting accuracy results under the Seq2Seq setting using Transformer, Informer, and IAAS.



(a) RMSE (b) MAE

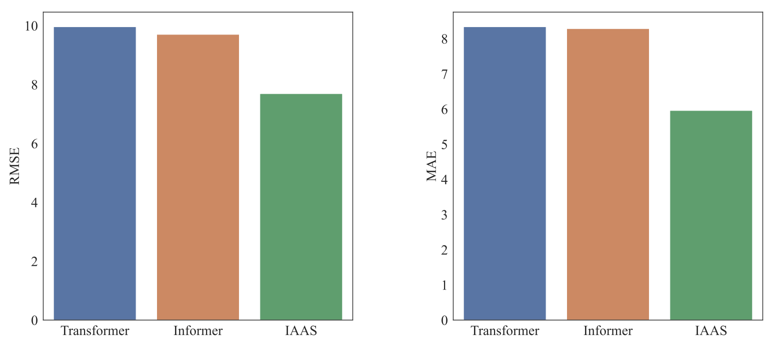
Figure A.7 One-day ahead wind power forecasting accuracy results using Transformer, Informer, and IAAS.

The one-week ahead forecasting experiment is designed to use hourly historical data in the past 14 days (336 steps) to predict the data in the future seven days (168 steps). Hence, the input sequence sizes for the load and wind data are 336×3 and 336×9, respectively. The output sequence sizes for these two cases are both 168×1. Similar to one-day ahead forecasting experiment, we also use 3-month data for training but use the following 7 days for testing. We move the rolling window every seven days for the next round of training and testing. Therefore, for this case, we only obtain one forecasting sequence with the size of 168×1 for the corresponding week. In other words, for one-week ahead load forecasting, the Seq2Seq setting is 336×3 to 168×1, which is a many-to-many setting. Similarly, for one-week ahead wind power forecasting, the Seq2Seq setting is 336×9 to 168×1. This means that we predict the next one-week load or wind power using the previous two-week real data. For example, on Jan 1, we forecast the load or wind power in the following week (i.e., from Jan 2 to Jan 8). Hence, we will use the IAAS framework to search a Seq2Seq neural architecture for the load or wind power forecasting. This model will be only used once. On Jan 8, we update this Seq2Seq model by using IAAS with another search to forecast the load or wind power in the following week (i.e., from Jan 9 to Jan 15). We put the experiment results in Figs. A.8 and A.9. As seen, IAAS still achieves better accuracy in the week-ahead forecasting case than the other two transformer models under the seq2seq setting. All in all, the IAAS framework is likely to generate better electricity time-series forecasting models than the existing transformer models under the seq2seq setting. In other words, the IAAS framework is better for the practical applications in power systems.



(a) RMSE (b) RMSLE (C) MAE

Figure A.8 One-week ahead load forecasting accuracy results under the Seq2Seq setting using Transformer, Informer, and IAAS.



(a) RMSE (b) MAE

Figure A.9 One-week ahead wind power forecasting accuracy results using Transformer, Informer, and IAAS.

Appendix K: Heuristic Mechanism Exploration

In this section, we explore the heuristic mechanism with an ablation experiment to demonstrate how it works. Specifically, we develop three variants of IAAS framework considering this mechanism.

The first one is without the heuristic approach, named IAAS_h. Specifically, we initialize five networks in the net pool and transform them with RL. The five transformed networks will replace their parent networks in the net pool and go to the next-round transformation. Such an experiment is similar to running an EAS framework (Cai et al. 2018), which uses the transformed network directly for the next-round transformation. This setting follows the traditional paradigm of RL that uses equal fixed-length trajectories to collect the samples.

The second one is without the random network generation in the heuristic algorithm, named IAAS_r. Using this one, we can demonstrate the importance of randomly generating a network for each update of the net pool. Particularly, this setting in the heuristic algorithm is to provide more samples at the early stages of an MDP trajectory.

The third one is based on the first one but randomly deleting five networks from the net pool. We name this one IAAS_d. In particular, if there are five networks in the net pool, there will be ten networks in the net pool after transformation. At this time, we randomly eliminate five networks. Such a variant is a version after using a simple trajectory truncation strategy. With this, we can demonstrate whether our elimination strategy is appropriate.

As noted, all these three variants also use the same computational budget as ours, i.e., 1000 action times. Therefore, for the first variant, the length of each trajectory is 200. As to the second and third ones, due to the trajectory termination, the lengths of trajectories are not the same. We use the ME load dataset and the WF1 wind power dataset for this experiment. We present the results in the following table with eight scenarios. We then have the following findings:

1. Comparing IAAS with IAAS_h, our IAAS achieves the better performance in seven scenarios than IAAS_h. This indicates the importance of the heuristic mechanism in IAAS, compared to the traditional method using fixed-length trajectories.

2. Comparing IAAS with IAAS_r, our IAAS achieves better performance in seven scenarios than IAAS_r. This demonstrates the importance of designing a random network generation component in the heuristic algorithm.

3. Comparing IAAS with IAAS_d, our IAAS achieves a better performance than IAAS_d in all scenarios in terms of all metrics except the ME_summer scenario. As seen, IAAS_d achieves the lowest MAE value in ME_summer scenario, but our IAAS has the better performance in this scenario than the other three variants regarding the RMSE and RMSLE values. This result clearly demonstrates the importance of the network elimination strategy in the heuristic mechanism.

In addition to these three, we also find out that IAAS_r performs better than IAAS_h in five out of seven scenarios in terms of three metrics. As to the ME_winter scenario, we believe IAAS_r has a similar performance to IAAS_h since in this scenario, IAAS_r has a better RMSE value, IAAS_h has a better MAE value, and these two have an equal RMSLE value. Although IAAS_r does not have the component of randomly generating one network in each episode, this result also indicates the superiority of our proposed heuristic algorithm.

Table A.3 The ablation experiment result of heuristic mechanism exploration

Scenario	Metrices	IAAS	IAAS_h	IAAS_r	IAAS_d
ME_spring	RMSE	61.807	67.807	**61.277**	67.220
	MAE	48.879	54.089	**48.715**	54.079
	RMSLE	**0.049**	0.053	**0.049**	0.053
ME_summer	RMSE	**69.154**	96.909	74.734	75.200
	MAE	58.187	71.701	57.757	**56.529**
	RMSLE	**0.052**	0.070	0.055	0.055
ME_autumn	RMSE	**38.483**	42.983	39.765	43.342
	MAE	**29.526**	33.153	29.925	32.892
	RMSLE	**0.031**	0.034	0.032	0.035
ME_winter	RMSE	**85.386**	97.573	97.071	102.469
	MAE	**68.773**	77.845	80.431	82.645
	RMSLE	**0.062**	0.071	0.071	0.075
WF1_spring	RMSE	**3.901**	4.389	3.963	4.143
	MAE	**2.996**	3.497	3.012	3.360
WF1_summer	RMSE	3.290	**3.064**	3.381	3.149
	MAE	2.614	**2.204**	2.553	2.313
WF1_autumn	RMSE	**2.237**	2.680	2.633	2.798
	MAE	**1.633**	1.983	1.882	2.124
WF1_winter	RMSE	**4.500**	4.655	5.245	5.865
	MAE	**3.427**	3.775	4.078	4.664

Appendix L: Details of Ablation Studies

Observing Table A.4, IAAS_LoadNet has better performance in all cases as compared to IAAS_n_LoadNet and IAAS_sn_LoadNet, in terms of the RMSE, MAE and RMSLE values. Averagely, IAAS_LoadNet performs better than IAAS_n_LoadNet with 13.5% (RMSE), 13.5% (MAE), and 13.4% (RMSLE), whereas IAAS_LoadNet is also better than IAAS_sn_LoadNet with 15.3% (RMSE), 14.8% (MAE), and 14.7% (RMSLE). In addition, IAAS_LoadNet achieves a better performance than IAAS_s_LoadNet in six out of eight cases in terms of three evaluation metrics. However, observing the NH-summer and NH-autumn cases, the forecasting accuracies from IAAS_LoadNet are close to those from IAAS_s_LoadNet. Averagely, IAAS_LoadNet has a better performance than IAAS_s_LoadNet with 10.4% (RMSE), 10.0% (MAE) and 9.4% (RMSLE).

Table A.4 Load forecasting accuracy results in ablation experiments

Case	Metrices	IAAS_LoadNet	IAAS_s_LoadNet	IAAS_n_LoadNet	IAAS_sn_LoadNet
ME-spring	RMSE	**61.807**	69.398	68.646	65.031
	MAE	**48.879**	56.255	55.988	49.814
	RMSLE	**0.049**	0.055	0.054	0.052
ME-summer	RMSE	**69.154**	93.573	83.095	83.754
	MAE	**58.187**	70.889	67.327	64.173
	RMSLE	**0.052**	0.067	0.061	0.06
ME-autumn	RMSE	**38.483**	61.95	46.891	43.282
	MAE	**30.526**	47.82	35.718	33.441
	RMSLE	**0.031**	0.049	0.038	0.035
ME-winter	RMSE	**85.386**	102.263	105.682	102.728
	MAE	**68.773**	79.342	81.106	84.5
	RMSLE	**0.062**	0.075	0.077	0.075
NH-spring	RMSE	**68.749**	73.36	80.832	80.54
	MAE	**53.759**	59.191	63.534	62.833
	RMSLE	**0.058**	0.062	0.067	0.068
NH-summer	RMSE	144.016	**140.237**	152.769	157.385
	MAE	114.45	**113.796**	123.907	129.272
	RMSLE	0.096	**0.093**	0.101	0.106
NH-autumn	RMSE	62.468	**60.077**	71.592	95.843
	MAE	45.985	**45.005**	54.309	69.45
	RMSLE	0.052	**0.049**	0.06	0.08
NH-winter	RMSE	**82.911**	83.442	98.854	95.489
	MAE	**65.506**	67.772	80.264	76.723
	RMSLE	**0.062**	**0.062**	0.074	0.072
Average	RMSE	**76.622**	85.537	88.545	90.506
	MAE	**60.758**	67.509	70.269	71.276
	RMSLE	**0.058**	0.064	0.067	0.068

Observing Table A.5, IAAS_WindNet also provides much better performance than IAAS_n_WindNet and IAAS_sn_WindNet in terms of the forecasting accuracy. On average, IAAS_WindNet obtains the better forecasting accuracy than IAAS_n_WindNet with 16.7% (RMSE) and 17.2% (MAE), and obtains the better forecasting accuracy than IAAS_sn_WindNet with 17.5% (RMSE) and 18.6% (MAE). Moreover, IAAS_WindNet performs better than IAAS_s_WindNet in most of cases regarding the two evaluation metrics. On average, IAAS_WindNet performs better than IAAS_s_WindNet with 2.4% (RMSE) and 2.4% (MAE).

Table A.5 Wind power forecasting accuracy results in ablation experiments

Case	Metrices	IAAS_WindNet	IAAS_s_WindNet	IAAS_n_WindNet	IAAS_sn_WindNet
WF1-spring	RMSE	**3.901**	3.953	4.845	4.473
	MAE	**2.996**	3.095	3.731	3.402
WF1-summer	RMSE	3.29	**3.148**	3.649	3.866
	MAE	2.614	**2.286**	2.754	3.246
WF1-autumn	RMSE	**2.237**	2.411	2.658	3.138
	MAE	**1.633**	1.817	1.911	2.298
WF1-winter	RMSE	**4.5**	4.555	5.291	5.249
	MAE	**3.427**	3.582	4.331	4.146
WF2-spring	RMSE	**4.922**	5.147	5.762	5.967
	MAE	**3.953**	4.027	4.147	4.704
WF2-summer	RMSE	4.87	**4.854**	5.193	5.077
	MAE	**3.591**	3.774	4.095	4.067
WF2-autumn	RMSE	6.031	**5.824**	7.655	7.22
	MAE	4.439	**4.386**	6.164	5.284
WF2-winter	RMSE	**3.104**	3.777	4.287	4.84
	MAE	**2.402**	2.711	3.136	3.651
Average	RMSE	**4.107**	4.209	4.917	4.979
	MAE	**3.132**	3.210	3.784	3.850

Appendix M: Further Discussion

The IAAS framework essentially contains two basic operations (i.e., wider transformation and deeper transformation) for network enlargement, and one pruning operation to shrink the network. Literature works usually use some other techniques in the forecasting model development, such as residual connections in Chen et al. (2018) and dropout strategy in Hossain et al. (2021). In this section, we will discuss whether these techniques can be integrated with IAAS.

In IAAS, we limit the use of the ReLU activation function considering the identity mapping. However, other activation functions (e.g., SeLU and Sigmoid) can also be applied if we use residual connections and set the output of the newly inserted layer to produce a zero tensor as the output. In other words, we can also use residual connections in IAAS for the model performance improvement. Note that in EAS, Cai et al. (2018) has already used residual connections in NAS. Moreover, we can also consider the dropout strategy in IAAS since it does not conflict with the network enlargement and shrinkage transformations. Furthermore, we can add a regularization term to the loss function for the model generalization improvement and exploit the early stopping strategy instead of the fixed number of episodes to shorten the computational time. Note that pruning, dropout, regularization, and early stopping techniques can all mitigate the overfitting issues. Therefore, one of the future research areas may consider the integration of these four techniques to address the problem of overfitting in NAS.

In addition to these techniques, we can also consider other network structures as search candidates. In this study, we exploit the RNN and LSTM to extract features from the electricity time-series data. However, the LSTM we use is variant of the vanilla LSTM (Graves and Schmidhuber 2005), named "no-peepholes" (NP) LSTM. Besides NP LSTM, Greff et al. (2016) also derived the other seven variants (e.g., no-input-gate LSTM, no-forget-gate LSTM, etc.) by separately deleting one component from the vanilla LSTM. Since the main structures of these LSTM are similar, the proposed function-preserving network transformation and the movement pruning can be applied to all of them with minor changes. For example, when using vanilla LSTM, we need to initialize the parameters of peepholes at the transformation process since our formulation in Section 3 is based on NP LSTM. In the future, we will consider these existing techniques and the LSTM variants in IAAS so that we can build a more accurate forecasting model for power systems.