applied sciences

MDPI

Article

Deep-Learning Model Selection and Parameter Estimation from a Wind Power Farm in Taiwan †

Wen-Hui Lin [1], Ping Wang [1,*], Kuo-Ming Chao [2], Hsiao-Chung Lin [1], Zong-Yu Yang [1] and Yu-Huang Lai [1]

[1] Green Energy Technology Research Center, Faculty of Department of Information Management, Kun Shan University, Tainan 710303, Taiwan; linwh@mail.ksu.edu.tw (W.-H.L.); fordlin@mail.ksu.edu.tw (H.-C.L.); s109000200@g.ksu.edu.tw (Z.-Y.Y.); s106001738@g.ksu.edu.tw (Y.-H.L.)

[2] Department of Computing & Informatics, Bournemouth University, Bournemouth BH12 5BB, UK; kchao@bournemouth.ac.uk

* Correspondence: pingwang@mail.ksu.edu.tw; Tel.: +886-6-205-0545

† This paper is an extended version of our paper published in 8th IEEE International Conference on Applied System Innovation 2022 (IEEE ICASI 2022), Nantou, Taiwan, 21–23 April 2022.

Abstract: Deep learning networks (DLNs) use multilayer neural networks for multiclass classification that exhibit better results in wind-power forecasting applications. However, improving the training process using proper parameter hyperisations and techniques, such as regularisation and Adam-based optimisation, remains a challenge in the design of DLNs for processing time-series data. Moreover, the most appropriate parameter for the DLN model is to solve the wind-power forecasting problem by considering the excess training algorithms, such as the optimiser, activation function, batch size, and dropout. Reinforcement learning (RN) schemes constitute a smart approach to explore the proper initial parameters for the developed DLN model, considering a balance between exploration and exploitation processes. Therefore, the present study focuses on determining the proper hyperparameters for DLN models using a Q-learning scheme for four developed models. To verify the effectiveness of the developed temporal convolution network (TCN) models, experiments with five different sets of initial parameters for the TCN model were determined by the output results of Q-learning computation. The experimental results showed that the TCN accuracy for 168 h wind power prediction reached a mean absolute percentage error of 1.41%. In evaluating the effectiveness of selection of hyperparameters for the proposed model, the performance of four DLN-based prediction models for power forecasting—TCN, long short-term memory (LSTM), recurrent neural network (RNN), and gated recurrence unit (GRU) models—were compared. The overall detection accuracy of the TCN model exhibited higher prediction accuracy compared to canonical recurrent networks (i.e., the GRU, LSTM, and RNN models).

Keywords: wind power forecasting; reinforcement learning; Q-learning; hyperparameters; parameter estimation

Citation: Lin, W.-H.; Wang, P.; Chao, K.-M.; Lin, H.-C.; Yang, Z.-Y.; Lai, Y.-H. Deep-Learning Model Selection and Parameter Estimation from a Wind Power Farm in Taiwan. *Appl. Sci.* **2022**, *12*, 7067. https://doi.org/10.3390/app12147067

Academic Editor: Mohamed Benbouzid

Received: 1 July 2022
Accepted: 12 July 2022
Published: 13 July 2022

1. Introduction

Wind power forecasting techniques refer to estimates of the expected electrical output from one turbine or many wind turbines of a wind farm in the near future, up to one year. The prediction method for wind power forecasting involves performing model pre-training and power prediction by collecting meteorological information and wind power generation historical data. Meteorological historical data include temperature, humidity, average wind speed, and wind direction. Conventionally, meteorological engineers use physical methods for wind forecasting associated with numerical weather prediction (NWP) to identify weather changes. Treating weather changes as deterministic events is not a trivial task. In fact, it requires that meteorological engineers handle numerical weather data updates for immediate weather changes. Basically, forecasting of wind power generation becomes imprecise when the NWP results are poor. Moreover, current statistical schemes, such as

autoregressive (AR) and autoregressive integrated moving average (ARIMA) [1], have been proposed to determine the relationship between weather features and predicted power, where weather changes are considered random processes. Statistical models generally perform well in simple time series forecasting, but their ability to process nonlinear data in dynamic states remains insufficient [2], and the fitting performance is limited.

Recently, numerous machine learning (ML) algorithms [3–5] have been proposed to help engineers distinguish the correlations between climate features and wind power outputs by accumulating historical meteorological information and wind power generation data. For example, artificial neural networks (ANNs), genetic algorithms, fuzzy regression, cluster analysis (K-means), and support vector machines have been introduced to solve the problems of wind-power forecasting in climate change.

However, traditional ML models for predicting the accuracy of wind-power forecasting have challenges with some problems, including, long-term stable prediction under climate change, as their complex learning architecture may result in low efficiency, long training time, and under-fitting [6]. Thus, developing a precise and robust approach for forecasting a wide range of meteorological phenomena with wind power forecasting remains a challenge.

To address the aforementioned problems in model training that often occur in the thin-layer network architecture of ML approaches, deep learning network (DLN) approaches use multilayer neural networks to extract the more complex climate features from historical weather information that accurately assist engineers in predicting the power generation (kw/h) for wind turbines. Typical deep learning models for sequence-to-sequence data include recurrent neural networks (RNN) [7], long short-term memory (LSTM) [8,9], gated recurrent units (GRU) [10], and temporal convolution networks (TCN) [11–13]. These four models have been used as learning models for the DLN to establish a classifier after their architecture hyperparameters and features are considered.

Practically, when attempting to build a DLN model, there are several questions that should be answered: (1) What is the best model for the wind power prediction problem to be solved? (2) How to select hyperparameters associated with the model? Hyperparameters include model architecture-related parameters, such as the size of hidden layers. Evidently, one should decide training-related parameters, such as activation function, batch size, the learning rate, and stopping time, given the alternatives of the optimizers, such as RMSprop, SDG, and Adam [14].

We attempt to answer these questions in this study and achieve high prediction precision with low convergence errors of the cost function in the predicted output.

To answer the first question, there are a number of options with regard to DLN model choice; for example, if the input data form a sequence with dependencies between the elements of the sequence, then an RNN is required. However, the RNNs suffered from gradient exploding and vanishing problems with long sequence inputs [8,9]. The choice of an appropriate model for wind-power forecasting is discussed later.

For the second question, the process of setting the hyperparameters requires expertise and extensive trial and error. Some drawbacks of the developments in wind power forecasting have been discussed, such as the use of manual trial and error for setting the initial parameters for the model in the training process. There are no simple and easy ways to set the hyperparameters. Recently, a reinforcement learning (RL)-based Q-learning algorithm [6,15,16] has shown great success in tuning hyperparameters in controlled environments, such as energy forecasting, Atari, robotic manipulation, and Go. The Q-learning algorithm uses a model predictive control process to optimise a reward by an agent interacting with the environment states and makes better decisions compared to the traditional trial and error approaches.

To improve forecasting accuracy for wind turbines, this study proposes a DLN-based model [7–13] with a Q-learning algorithm for wind power forecasting to determine the hyperparameters for four competing models by minimising the prediction error. There are several options with regard to DLN model choice; here, four sequence-to-sequence models for deep learning were compared to examine the model performance and recommendations

for wind power forecasting. In the experiment, four completing models were pre-trained using real historical climate data and power generation outputs of wind turbines from a wind power farm in the Chang-Bin industrial zone of Taiwan. The experimental results show that the TCN exhibited higher prediction accuracy compared to recognized recurrent networks, such as GRUs, LSTMs, and RNNs. In developing the proposed model, our work focused on three important aspects:

- In this study, the proper setting of the hyperparameters for the developed model was investigated using the Q-learning algorithm by minimising the loss of the cost function in the learning process.
- Five initial parameters for the developed model were analysed by incorporating the Q-learning algorithm in the learning process: (i) number of stacks, (ii) filter size, (iii) dilatation coefficient, (iv) activation function, and (v) optimiser to decide the proper setting of hyperparameters for model training by the input of 1-year dataset of wind farms from the Chang-Bin industrial zone (CBTP) for Tai-Power Corporation in Taiwan.
- In our experiment, the TCN mode for 168 h (7 days) ahead, mean absolute percentage error (MAPE) = 1.41%, had the highest prediction accuracy, followed by GRU MAPE = 3.72%, LSTM mode MAPE = 6.99%, and RNN mode MAPE = 28.18%.

The remainder of this paper is organised as follows. Section 2 reviews previous studies on the DLNs for sequence-to-sequence data and a reinforcement learning algorithm. Section 3 introduces the reinforcement learning framework and the proposed Q-learning approach for determining the proper hyperparameters for the wind-power forecast model. A performance analysis of the experimental results is presented in Section 4. Finally, Section 5 draws the conclusions.

2. Relate Work

In this section, we review convolutional neural networks for sequence-to-sequence data and a reinforcement learning algorithm.

2.1. Convolutional Neural Networks for Sequence-to-Sequence Data

Recently, researchers developed a DLN for time-series data processing that contains RNN, LSTM, GRU, and TCN used for handling sequence-to-sequence inputs, performing feature extraction of long-term sequence climate data, and estimating the expected electrical outputs for wind turbines [7–13].

Typically, an RNN is a class of artificial neural networks where connections between nodes form a directed graph along a sequence [7]. RNNs focus on sequential data forecasting, speech recognition, text generation, stock market prediction, and language translation. This specific design allows the network to process sequences of inputs using their internal state. This makes them suitable for tasks such as unsegmented connected handwriting recognition and speech recognition. However, RNNs suffer from the problem of vanishing gradients and have limited learning in long memory applications.

Due to the problem of exploding and vanishing gradients with long sequences, LSTMs were designed with three specific gates operating by controlling the cell states. Basically, LSTM units are units of an RNN developed to deal with the exploding and vanishing gradient problems that can be encountered when training traditional RNNs. Their relative insensitivity to gap length gives LSTM networks a key advantage over RNNs, hidden Markov models, and other sequence learning methods in numerous applications. The compact forms of the equations for an LSTM unit are as follows [8,9]:

Updating rules for the forget gate, input gate and output gate in an LSTM block

$$f_t = \sigma\Big(W_f[h_{t-1},\ x_t\Big] + b_t), \tag{1}$$

$$i_t = \sigma(W_i[h_{t-1},\ x_t] + b_i), \tag{2}$$

$$o_t = \sigma(W_o[h_{t-1},\ x_t] + b_o), \tag{3}$$

Candidate (memory) cell state

$$\widetilde{c}_t = tanh(W_c[h_{t-1},\ x_t] + b_c), \tag{4}$$

Memory cell and hidden state

$$c_t = f_t * c_{t-1} + i_t * \widetilde{c}_t, \tag{5}$$

$$h_t = o_t * tanh(c_t), \tag{6}$$

Cell output

$$y_t = h_t. \tag{7}$$

where symbol x_t denotes the input vector to the neural network, f is the forget gate (a neural network (NN) with sigmoid), $\widetilde{c}$ the candidate (memory) cell state (an NN with *tanh* function), i the input gate (an NN with sigmoid), W the weight matrices and b the bias vector parameters that need to be learned during training, o the output gate (an NN with sigmoid), h the hidden state (a vector), c the memory state (a vector) and σ is the activation function in an LSTM unit. Notably, b_t, b_i, b_o, and b_c represent bias vector parameters to be learned during training in forget gate, input gate, output gate, and memory cell, respectively.

Later, GRU was proposed to handle simple problems for shorter sequential data by reducing the number of gates. Unlike LSTM, GRU replaces the forget gate and input gate in the LSTM with an update gate. Engineers usually choose GRU, considering the effects of computing power and time overhead of the hardware [10].

In 2017, Lea et al. [11] proposed a new neural framework for sequence modelling tasks, namely temporal convolutional networks (TCNs) to address the problem for long sequential data of uncertain length. TCN consists of three parts: dilated causal convolution, nonlinear activation function, and residual connection. The TCN was designed using two basic principles: (i) Causal convolutions—the convolutions are causal, indicating that there is no information leakage from the future to past; (ii) One-dimensional fully convolutional network architecture—the architecture can take a sequence of any length and map it to an output sequence of the same length [12]. In other words, TCNs use one-dimensional (1D) separable convolutions that are a form of factorised convolutions that factorise a standard convolution into a depth-wise and pointwise convolution to preserve the massive memory requirements of deep learning models [13]. Further, a TCN was proposed to extend the input large memory processing using a dilated causal convolution architecture and residual connections to preserve more accurate prediction results.

A TCN block was regularly selected using the combinational 1D fully convolutional architecture followed by a nonlinear activation function (rectified linear unit, ReLU) to determine the weights of an input sequence that improve classification accuracy in model prediction, then adding a dropout function to reduce overfitting. In other words, a dropout function in a dense layer decreases the number of parameters to be learned and helps reduce overfitting. From a design perspective, the TCN is capable of handling a series of uncertain length and outputs it with the same length. Experimental results [17–20] have shown that TCNs exhibit better performance with sequences of uncertain length, where the TCN architecture performs well in very long sequences of inputs. Sequence data prediction is the most frequently encountered issue in the TCN model, which is illustrated as follows [11].

Consider a given training dataset $D(x_t, y_t)$ in a sequence modelling task, where x_t denotes input sequence ($x_t \in R^n, t = 1, \ldots,\ T$) and $\widetilde{y}_0, \widetilde{y}_1, \ldots, \widetilde{y}_T$ is a prediction sequence with the corresponding sequential input $\{x_0, x_1, \ldots, x_T\}$.

The mapping function f is defined for a temporal convolutional network as follows:

$$\widetilde{y}_0, \widetilde{y}_1, \ldots,\ \widetilde{y}_T = f(x_0,\ x_1, \ldots,\ x_T), \tag{8}$$

where $\widetilde{y}_t$ depends only on previous observations $x_0, x_1, \ldots, x_t$ at time t.

The network is trained by supervised learning to find the function f that minimizes the error between the actual outputs $(y_0, \ldots, y_T)$ and the predictions $(\widetilde{y}_0, \widetilde{y}_1, \ldots, \widetilde{y}_T)$. Under the causal constraint, the TCN develops dilated causal convolutions to expand the receptive field exponentially, taking more historical information into consideration. The dilated convolution operation F on element s of the 1-D sequence $x_t \in R^n$ for a filter f: $(0, 1, \ldots, k-1) \rightarrow R$ is formulated as Equation (9) [13]:

$$F(s) = x_t *_d (f)(s) = \sum_{i=0}^{k-1} f(i) \cdot x_{s-d.i}, \quad (9)$$

where k is the stack size, d is the dilation factor, and symbol * is the convolution operator. The dilated causal convolution used in this paper is illustrated in Figures 1 and 2 with dilation factors d = 1, 2, 4, 8 and stack size k = 2. Definitely, the larger the stack sizes k or dilation factor d are set, the broader the receptive field of the network is.

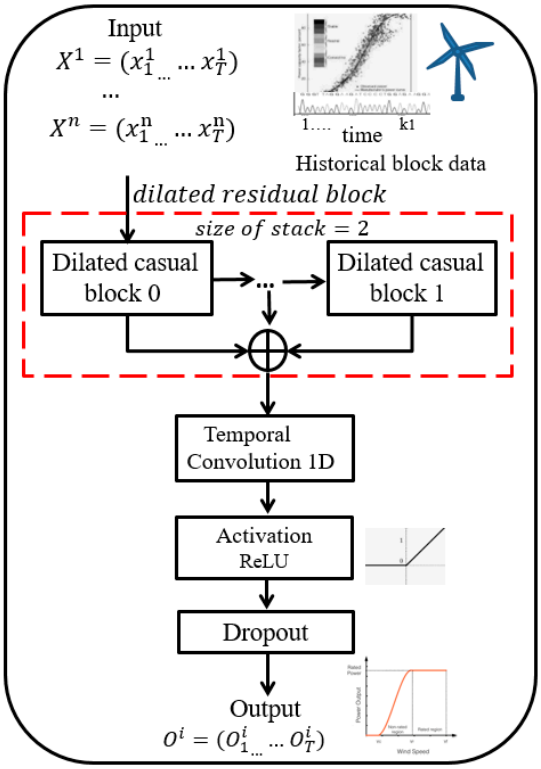

Figure 1. TCN-based architecture for wind power forecasting (dilated residual block k = 2).

2.2. Hyperparameter Optimization for Machine Learning Models

The performance of deep learning models strongly depends on choosing a set of optimal hyperparameters. In other words, the prediction performance of the deep learning network model is influenced quite heavily by the choice of hyperparameters, hence it can incorporate a reinforcement learning optimisation process to improve the searching efficiency in model development. Hyperparameter tuning is an optimization problem where the objective function of optimization is unknown. Traditional optimization techniques such as the Newton method or gradient descent cannot be applied.

Practically, data engineers find a set of hyperparameter values in the training phase to archive the best performance for machine learning models within a reasonable amount of time. Further, they continuously adjust hyperparameters for models with the selection of different sets of values for a learning algorithm and then compare the model performance to choose the best model. This process is called hyperparameter optimization.

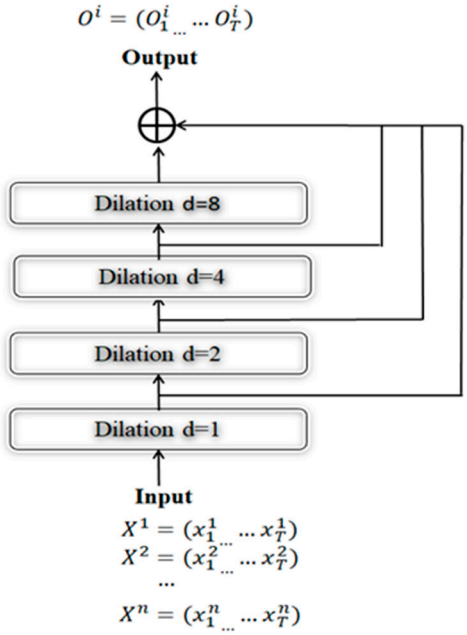

Figure 2. Detailed diagram of the dilated residual block.

Typically, there are mainly two kinds of hyperparameter optimization methods, i.e., manual search and automatic search methods. Manual search identifies the important parameters depending on the experience of expert users. The process of tuning hyperparameters is not easily reproducible. Moreover, as the number of hyperparameters and the range of values increase, it becomes quite difficult to manage [21]. To overcome the drawbacks of manual search, automatic search algorithms have been proposed, such as grid search [22–25]. Mainly, grid search trains machine learning models with different values of hyperparameters in the training set and compares the performance according to evaluation metrics. Finally, grid search outputs hyperparameters that have the optimal performance. Two hyperparameter optimization approaches are compared in Table 1.

Table 1. Comparison of two hyperparameter optimization methods.

Scheme	Features	Limitations
Manual search (MS)	The MS depends on the experience of expert users to identify the important parameters. It is suitable for expert users to handle simple or low-dimensional hyperparameter tuning problems using the visualization tools.	When the number of hyperparameters and the range of values increase, it becomes quite difficult to manage since humans are not good at handling high dimensional data.
Automatic search (AS)	The AS trains machine learning models with selection of the optimal hyperparameters for a model that result in the most precise performance.	Basically, the AS is a brute-force and exhaustive searching approach, and it suffers from a high computational load with time cost.

Automatically tuning hyperparameters for the ideal model structure is a challenging task. Numerous automatic search techniques comprising machine learning approaches with optimised algorithms have been used for achieving high-precision performance. These AS schemes are summarised in Table 2.

Table 2. Machine learning approaches for hyperparameter tuning.

Scheme	Achievement	Limitations
Lee, Park, Sim (2018) [21]	Proposed a novel approach to improve CNN performance by hyperparameter tuning in the feature extraction step using a parameter-setting-free harmony search (PSF-HS) approach. By two simulations, it is possible to improve the performance by tuning the hyperparameters in CNN architectures with reference to CifarNet and a Cifar-10 dataset proposed in the past.	A host needs relatively high computational capabilities to train the PSF-HS algorithms for image recognition.
Wu, Chen, et al. (2019) [22]	Proposed a hyperparameter tuning algorithm for machine learning models based on Bayesian optimization to find a set of hyperparameters that archive the best performance. Experimental results show that the proposed method can find the best hyperparameters for the widely used machine learning models, such as the random forest algorithm and the neural networks.	The model accuracy highly relies on large amounts of prior information to tune the hyperparameters for developing model.
Amirabadi, Kahaei, and Nezamalhosseini (2020) [23]	Proposed two novel suboptimal grid search methods to tune hyperparameters for deep learning networks for optical communication (OC) systems. Results indicate that despite greatly reducing computational complexity, the proposed methods achieved favourable performance on four experimental datasets.	The trained deep learning network (DNN) is a state-of-the-art deep learning algorithm used for modelling free space optical (FSO) communication in an optical communication system.
Lokku, Reddy, Prasada (2022) [24]	Proposed a CNN-based classifier for Optimal Face Recognition Network (OPFaceNet) to recognize facial images affected by high noise and occlusion. This algorithm identifies the hyperparameters for CNN model by optimizing the Fitness Sorted Rider Optimization Algorithm (FS-ROA) and achieves a good recognition rate of 97.2%.	The model accuracy highly relies on the FS-ROA, and the trained CNN is a specific deep learning algorithm used for facial recognition only.
Nematzadeh, Kiani, Torkamanian-Afshar, and Aydin (2022) [25]	Proposed a method to tune hyperparameters for machine learning algorithms using Grey Wolf Optimization (GWO) and Genetic algorithm (GA) metaheuristics. Experimental results show that in all trials, the performance of the training phases is improved. Additionally, GWO demonstrates better performance with a p-value of 2.6E	Providing a ready-to-use library and packages for different platforms and also an online web tool is needed in practice due to investigating a wider range of several machine learning training algorithms and different metaheuristics.

Although the methods in Table 2 achieve automatic tuning approaches such as grid search and can theoretically obtain the near optimal value of the hyperparameters, they suffer from a high computational load with time cost.

2.3. Reinforcement Learning Algorithm

To solve the problem of expensive cost in grid search, a dynamic programming-based reinforcement learning algorithm has been proposed to assist engineers in finding the optimal performance for specific problems by adjusting the definite hyperparameters for the developed model.

Notably, the reinforcement learning-based algorithm has shown great success in many real-world hyperparameter optimization problems, including hyperparameter tuning in controlled environments for deep learning network modelling [6,15,16,26–28]. Many researchers have focused on neural architecture search (NAS) and hyperparameter optimization using reinforcement learning approaches. For example, Zoph and Le addressed hyperparameter optimization of RNN and LSTM using reinforcement learning to design automation of deep learning networks [26]. Later, Iranfar et al. used reinforcement learning to tune the hyperparameters for a multi-layer perceptron (MLP) and CNN [27]. Jalali et al. used an ensemble learning scheme with reinforcement learning strategy to increase the prediction accuracy of wind power forecasting [28].

In the following, we review the basic principle of reinforcement learning algorithm.

In real applications, decisions seldom observe the complete state of the system; it is more common to receive observations of the state that are noisy or incomplete. Markov decision processes (MDPs) with stochastic theory have been proposed to solve these model decision problems in incomplete states. The MDP focuses on finding a good policy that will yield the maximum cumulative rewards by taking a series of actions with respect to specific states of the observed system.

In the problem model for the MDP, an agent (decision maker) may interact with its environment and change its behavior using an action in available states S from the consequences of actions A. The dynamic process responds at the next time step by randomly moving into a new state and giving the decision maker a corresponding reward R [6].

A major approach in reinforcement learning, namely the Q-value approximation, is to select an action a_t that maximises the expected reward at any state s_t. The goal of the agent in Q-value approximation is to find an optimal policy in the sense of maximizing the expected value of the total reward over any and all successive steps, starting from the current state. Therefore, this algorithm calculates the quantity of a state-action pair as [26].

$$Q: S \times A \rightarrow R \tag{10}$$

The update rule Q-value approximation is shown as

$$Q(s,a) = (1-\alpha)Q(s_t,a_t) + \alpha\left(r_{t+1} + \gamma \max_a Q(s_{t+1},a)\right) \tag{11}$$

Here, $\alpha \in (0, 1)$ is the learning rate. A factor of value '0' lets the agent not learn anything, while a factor of value '1' lets the agent consider only the most recent information. Equation (11) indicates that the Q-value approximation is a multi-decision process based on the MDP framework to provide a generalised framework for optimal behaviour based on actions of the agent to maximise the cumulative rewards of a series of decisions.

Recently, Iranfar et al. addressed hyperparameter optimization of CNNs through a Q-learning-based approach with a multi-agent system [27]. In the proposed multi-agent Q-learning-based approach, agents are able to communicate through our novel definition of state-action pairs, Q-tables, and a Q-table update rule. Thus, a Q-learning scheme was adopted for hyperparameter optimization and to examine the predicted accuracy of the model for overcoming an expensive cost problem scheme in grid search.

3. Determining the Hyperparameters for the Developed Model with Q-Learning Scheme

This section presents our proposed TCN model design for wind power forecasting 24–168 h ahead, incorporating a reinforcement learning algorithm for long-term, wind

power forecasting. The proposed model comprises three phases: the architecture design phase, hyperparameter selection phase and the overall process for model development phase.

3.1. Architecture Design for the TCN Model

For a TCN model design for wind power forecasting 24–168 h ahead, it is expected that the resulting deep features will consist of climate features obtained through the convolutional feature learning process from historical weather information, thereby reducing the prediction errors of power generation (kw/h) for wind turbines as much as possible.

Adopting TCNs for wind power forecasting has two advantages. First, in the training phase, TCNs automatically learn the correlations to capture complex variations with wind power output by leveraging a large amount of training data. Second, during the learning and testing phase, deep learning networks can be easily parallelised on GPU cores for acceleration, allowing the results to be quickly obtained [17]. Thus, we incorporated the convolutional network architecture involved in casual convolution with residual connections to construct a stable TCN-based prediction model for 24–168 h wind power forecasting.

In the present study, a wind power forecasting technique was developed based on a TCN model, with the goal of achieving high predictive accuracy with the minimum convergence error where the input sequence comprises historical wind speed and direction observations over past periods, while the actual output is the current wind power output.

First, TCN is used to learn the correlations between meteorological features and wind power generation from various historical data. In the case of wind power forecasting, the TCN model can be defined as follows [11]:

$$x_t^l = \sigma(W_x^l \cdot F_d\left(x_t^{l-1}\right) + b_x^l + x_t^{l-1}), \tag{12}$$

where $F_d(.)$ is the dilated convolution function of d-factor; x_t^l is the value of the neuron of l-th hidden layer at time t; W_x^l and b_x^l are the weights and bias corresponding to the l-th hidden layer, respectively; and σ is the activation function.

A detailed workflow of the wind power forecasting using dilated residual blocks to extend a large receptive field for processing historical data is shown in Figure 1. In theory, TCN has two effective approaches to increase the receptive field size, that is, increase the dilation factor d and choose a larger filter size k.

As shown in Figure 1, we proposed a TCN based dilated causal convolution design with dilation factors d = 1, 2, 4, 8 and filter size k = 2. In the residual block, TCN uses a two-layer dilated causal convolution, the activation function of the convolution is a linear rectification function (ReLU), and weight normalization is used to normalize the convolution filter. Further, we inserted a dropout layer after each convolutional layer to avoid overfitting.

In essence, the TCN model was composed of one or more 1D convolutional layers with dilated residual blocks for input processing. Through use of the residual block in the convolutional layers in TCN, deep and large TCNs can achieve further stabilisation. TCNs are useful for all types of long sequence problems because they can perform feature extraction and correlation analyses. An example of the residual block (d = 8) used between each layer in the TCN to accelerate convergence and enable the training of deeper models is shown in Figure 2.

3.2. Hyperparameter Selection for the TCN Model Using Q-Learning Algorithm

Inspired by Iranfar, Zapater, and Atienza in [27], we adapted Q-learning agents to split the design space into independent smaller design sub-spaces such that the agents fine-tune the hyperparameters for the assigned layer and provide faster, yet accurate design space search. Thus, the optimal parameters for the developed wind power prediction model with Q-learning algorithm are analyzed as follows.

In the training phase, the present study used a TCN model pre-trained on the experimental dataset containing collected wind turbine data from the Chang-Bin wind power

plant in Taiwan from 2020 to 2021 (2-year period) and incorporated a Q-learning algorithm to determine the optimal parameters for the developed models in the training process (Figure 3). As shown in Figure 4, the selection process for architecture parameters is modelled as an MDP.

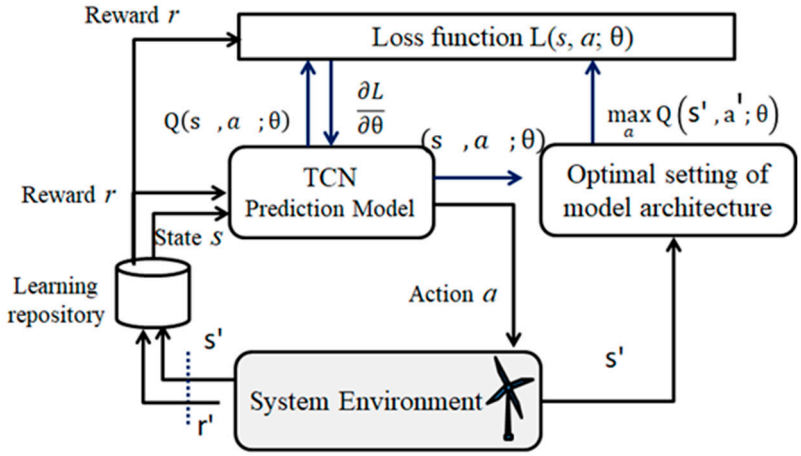

Figure 3. Architecture of a TCN model for wind power forecasting enhanced with Q-learning algorithm.

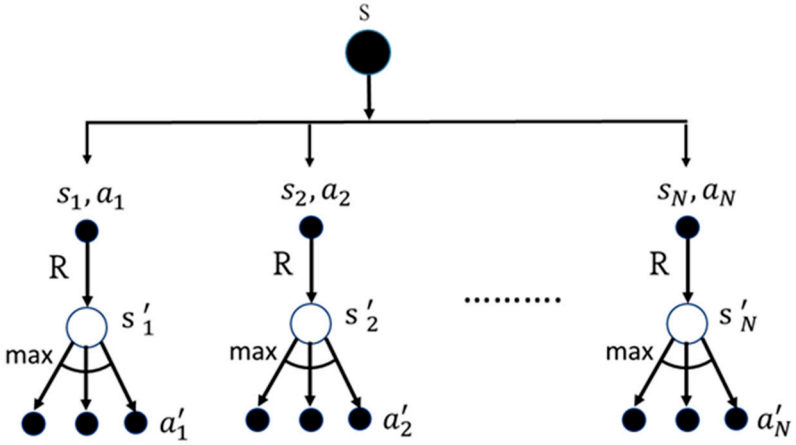

Figure 4. Update process for *Q*-value computation.

To solve the problem of exploring the proper setting of hyperparameters for the TCN model, this study proposed a prediction model involving Q-learning with the reinforcement learning approach as follows.

Initially, the loss function of the Q-learning model, $\mathrm{L}(\theta)$, is given by

$$\mathrm{L}(\theta) = E\left[\left(r_k(s,a;\theta) + \gamma \max_a \mathrm{Q}_k(s',a';\theta) - \mathrm{Q}_k(s,a;\theta)\right)\right], \tag{13}$$

where $r_k(s,a;\theta)$ represents the reward value of the current state s, θ is the parameter from the previous iteration, s' is the state after performing action a, and γ is a number between 0 and 1, $0 \leq \gamma \leq 1$, called the discount factor, which trades off the importance of sooner against later rewards. Further, γ may also be interpreted as the likelihood of succeeding at every step k. $\mathrm{Q}(s,\ a\ ;\theta)$ denotes the quantity of a state-action pair (s, a). By performing action a, the agent can move from a state s to a new state s', in which each state provides the agent with a reward.

As described above, the agent is used for maximising its total reward by learning the optimal action for each state.

To minimise the errors between the predicted values and actual outputs (i.e., maximise the reward to minimise the prediction errors between prediction value and real output) for wind power forecasting, we define the reward (R) between state s and new state s' as

$$R = \left(TCN_{Loss(s)} - TCN_{Loss(s')}\right). \tag{14}$$

In finite state-action spaces, the Q-learning approach can solve for Q-value using an update rule as follows (Figure 4).

$$Q_{k+1}(s,a;\theta) = Q_k(s,a;\theta) + \alpha[r_k(s,a;\theta) + \gamma \max_a Q_k(s',a';\theta) - Q_k(s,a;\theta)], \tag{15}$$

where $Q_{k+1}(s,a;\theta)$ is a new Q-value of the state-action pair for state s, $Q_k(s,a;\theta)$ is the Q-value of the state-action pair for current state s; $r_k(s,a;\theta)$ is the reward value of the state-action pair for current state s, and $Q_k(s',a';\theta)$ represents the Q-value of the state-action pair for a new state s'.

In summary, the Q-learning computation was performed in the following five sub-steps [21,22]:

1. Initialise Q-values $Q(s,\ a;\theta)$ arbitrarily for all state-action pairs.
2. For life, or until learning is stopped.
3. The agent selects action (a) in the current state (s) based on the current Q-value and estimates state-action pair $Q(s,a;\theta)$ using Equation (15).
4. Applying Equation (15), Q-values are updated after a new state and reward are observed.
5. Calculate $\mathrm{L}(\theta)$, and then stop updating when $\Delta\mathrm{L}(\theta) < \varepsilon$.

Selection of the optimal parameters (θ) for the developed TCN model is illustrated in Figure 5.

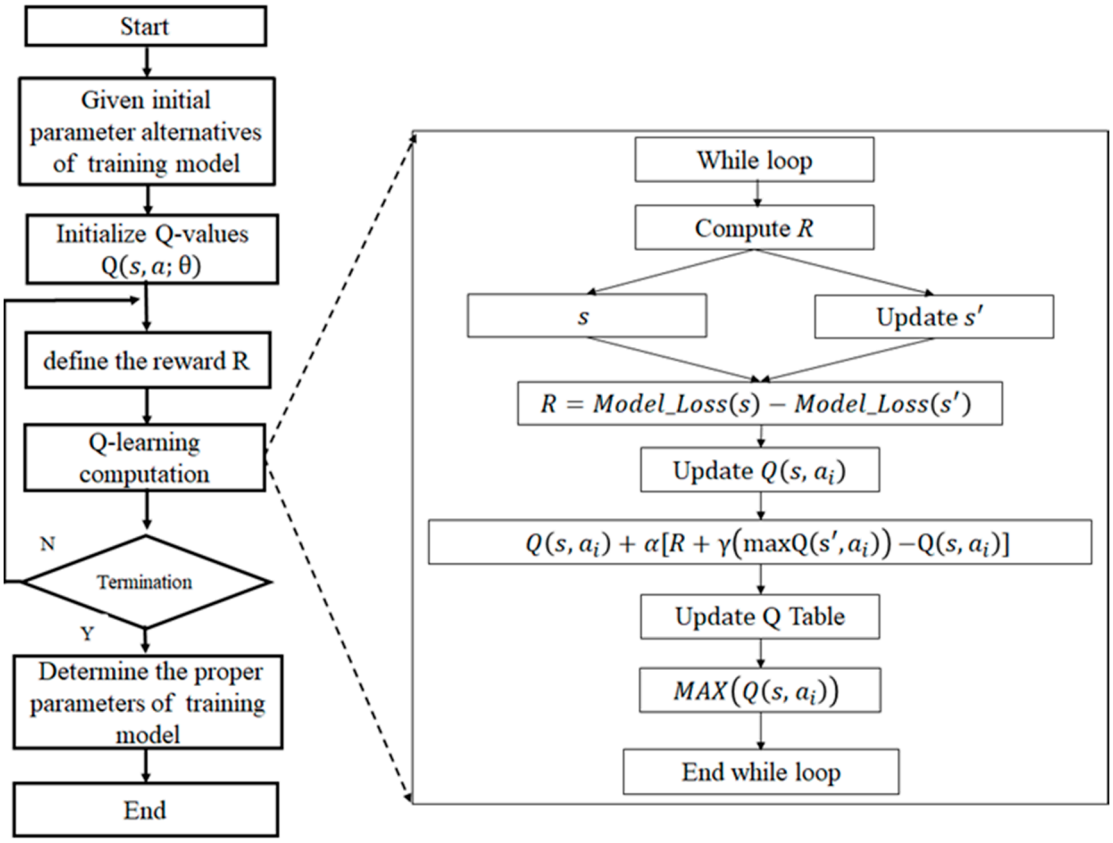

Figure 5. Parameter selection for the TCN model using the Q-learning algorithm.

In summary, the detailed Q-learning algorithm for wind power forecasting with the TCN learning model described by the PDL is as Algorithm 1.

Algorithm 1. Q-learning for Hyperparameter Tuning of TCNModel

Repeat
The state vector S consists of N independent states
$S = [s_1, s_2, \ldots, s_N]$
Set $R = 0$
For each state s_i in S
Initialize $Q(s_i, a_i)$, $\forall s_i \in S_i$, $a_i \in A(s_i)$, arbitrarily, and $Q(terminal - state, 0) = 0$
Repeat (for each episode of ith state s_i)
Initialize S_i
Repeat (for each step of episode)
Choose A from S_i using policy derived from Q
Take action A, observe R, S_i'

$$S' = [s_1, \ldots, s_i', \ldots, s_N]$$

$$R = TCN_{Loss(s)} - TCN_{Loss(s')}$$

$$Q_{k+1}(S_i, A; \theta) \leftarrow Q_k(s_i, a; \theta) + \alpha[R + \gamma \max_{a_i} Q_k(s_i', a_i'; \theta) - Q_k(s_i, a_i; \theta)]$$

$S_i \leftarrow S_i'$
Until S_i is terminal
Until $\Delta L(\theta) < \varepsilon$ (preset constant ε)

3.3. Overall Process for Model Operations

A detailed flowchart of the developed model for wind power forecasting is shown in Figure 6. The figure also illustrates the proposed TCN model incorporating three subphases in the model operation process: (i) data pre-processing, (ii) model training (vs. determination of proper setting for hyperparameters), and (iii) model validation.

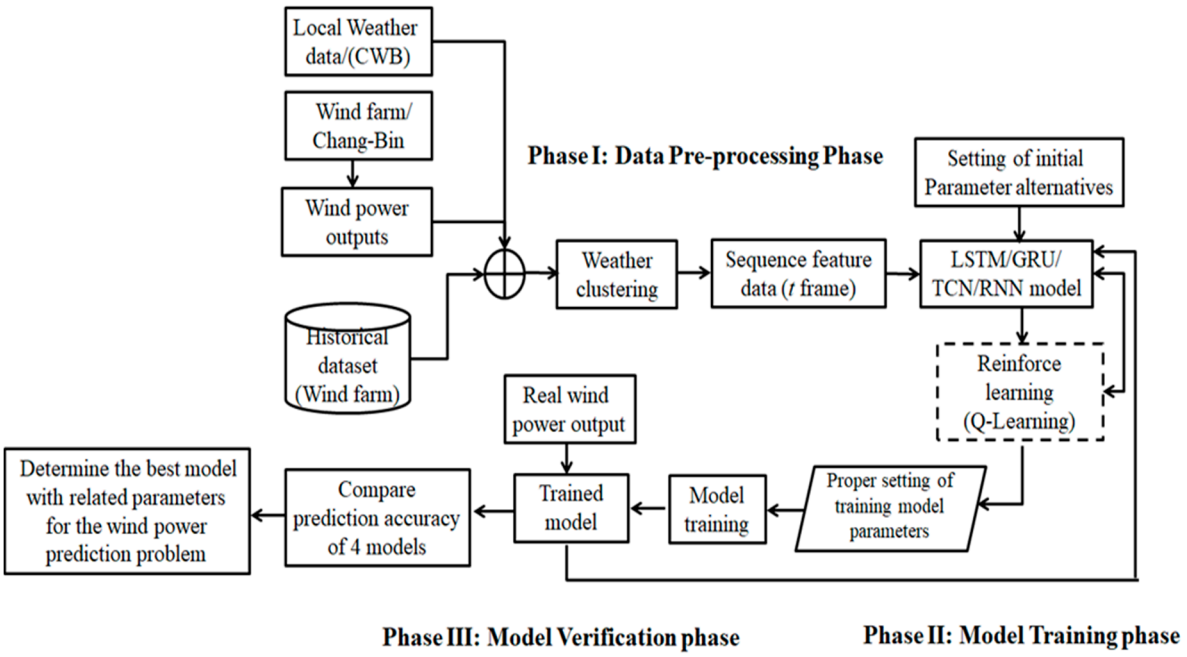

Figure 6. Execution process of the developed model for wind power forecasting.

Step 1. Data pre-processing

The wind turbine built in the Chang-Bin industrial zone is a Vestas V80 2 MW exemplar turbine. It has a rotor diameter of 80 m, hub height of 67 m, and rated speed of 14.5 m/s. A wind power farm in an industrial zone was built in 2006. Tai-Power Company constructed

a total of 23 wind turbines with a capacity of 2 MW, of which eight were erected in the western area of the Chang-Bin industrial zone, and the remaining 15 turbines were erected in the LunWei District. The experimental data are summarised as shown in Table 3.

Table 3. The basic features of the wind turbines in the Chang-Bin industrial zone.

Item	Feature
# of Wind turbines	31 units (2 MW)
Wind turbine	Vestas V80/2000 wind turbine, Denmark
Device capacity	62 MW
Rated power	2000 kW
Annual power generation	135 million kWh

Once the wind turbine location is located, the forecasting algorithm can include future weather forecasting data from the Central Weather Bureau (CWB) in Taiwan, where the forecasts on the CWB website are issued four times daily, updating future 7-day ahead weather forecasts for various areas of Taiwan. Moreover, historical data contain real weather observations and wind turbine power outputs with anomalous data, and engineers need to pre-process data for wind farm datasets before performing model pre-training.

Step 2. Model training

Step 2.1. Initial setting of hyperparameters from GitHub

Then, the proposed model uses a set of pre-training parameters from keras-tcn at GitHub [29] as initial parameters for model training. keras-tcn was used as the transfer learning (TL) model to fine-tune the developed model and improve learning speed and training accuracy. In the experiment, four crucial architecture-related parameters (hyperparameters) were selected from the transferring learning cases in the TCN predictor, namely (i) number of stacks, (ii) filter size, (iii) dilatation coefficient, (iv) activation function, and (v) optimiser. The accuracy (%) associated with the distinct model parameters for wind power prediction will be examined using Q-learning algorithm.

Step 2.2. Hyperparameter tuning of the model

In this phase, the following sub-steps were used to determine the proper parameters for the training model for wind-power forecasting using the Q-learning algorithm (Algorithm 1), by exploring these possible solutions and deciding on the proper parameter selection for the training model.

Step 3. Model validation

In the following, the system provides the benefit of a quick response for wind power forecasting through the use of neural net weights using four trained DLN models. To validate the accuracy of the four experimental models, an evaluation index was chosen to evaluate the prediction performance of the proposed model as follows:

The *MAPE* can be defined as

$$MAPE = \frac{1}{N}\sum_{t=0}^{N}\left|\frac{O_t - \hat{O}_t}{O_t}\right|, \tag{16}$$

where O_t is the real wind power value, $\hat{O}_t$ is the estimated wind power output, and N is the number of data points. As presented in Equation (15), the MAPE is calculated by dividing the estimation error by the real value of wind power generation.

4. Experimental Results

In this section, the performance of the proposed TCN-based model was demonstrated using an example of wind power prediction. The prediction system was executed in the Linux environment, and the software packages installed are listed in Table 4.

Table 4. Software packages installed in the wind power forecasting platform.

Item	Hardware/Software Installed
CPU	AMD Ryzen Threadripper, 1920X (3.4 G)
OS	Ubuntu 14.04 LTS 64
GPU	Nvidia GTX-2080TI x2
Language	Python 3.5
Library Package	TensorFlow-gpu 1.13, Pandas 0.23.4, Numpy 1.18, Matplotlib 3.3.2

Step 1. Data pre-processing phase

In the experiment, model pre-training used a project dataset of wind farms from the Chang-Bin industrial zone (CBTP) for Tai-Power Corporation in Taiwan, including real weather observations and wind turbine power outputs to pre-train the model. Each record of the CBTP dataset contained five parameters sampled every 10 min with a total of 192,817 samples listed. The five parameters in the historical file are listed as follows:

6. Date/time: 10 min intervals
7. Wind speed (m/s): The wind speed at the hub height of the turbine.
8. Wind direction (°): The wind direction at the hub height of the turbine
9. Real power output (kW): The power generated by the turbine for that moment.
10. Theoretical power curve (KWh): The theoretical power values that the turbine generates with that wind speed, which is generated by the turbine manufacturer.

To increase the recognition accuracy, the correct value of null fields was mislabelled to build a clean dataset for the experiment. The data cleaning process in experimental data was manually pre-processed to remove null fields and were pre-filled in the linear proportions of the neighbouring observation data. The training dataset used samples collected every 10 min from 1 January 2018 to 31 December 2018. Further, the test dataset in the experiment was 7 days in total, from 1 January to 7 January 2019.

Step 2. Model training phase

To examine model efficiency, the experiment incorporated four deep neural models for series data processing, that is, GNN, LSTM, GRU, and TCN, to conduct wind power forecasting 24–168 h (1–7 days) ahead.

First, a series of experiments were pre-trained to investigate the performance of the TCN-based classifier effectiveness using the training dataset, where the learning results were regarded as the basis for the proper selection of model parameters, including five important hyperparameters for structure design, i.e., the number of filters, dilatation coefficient, activation function, optimiser, and number of stacks.

To achieve proper selection of hyperparameters in a TCN model, we used a reward function defined in Equation (10) for neural network generation and hyperparameter tuning based on Markov decision process. The decision process for generating a TCN architecture with the optimal parameters on neural network layers using Q-learning algorithm is shown in Figure 7 [30].

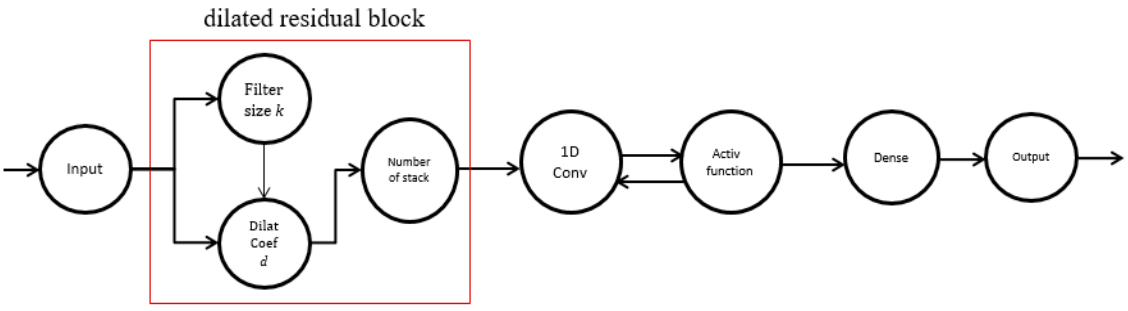

Figure 7. Decision process for generating a TCN architecture.

To improve input processing capability, the TCN model was developed to generate the receptive field to solve series data input problems. In particular, TCNs address the problems associated with dilated causal convolution, and residual connection. In theory, the receptive field size of the TCN depends on the network depth n (number of stacks), filter size k, and dilation factor d; making the TCN deeper and larger is important to obtain a sufficiently large receptive field. Empirically, making the network deep and narrow, which means stacking a large number of layers and choosing a thin filter, results in an effective architecture [31].

(i) Number of stacks

First, we need to decide the number of stacks for the TCN model. In essence, the network architecture of a TCN is an extension of a 1D CNN in which a series of 1D convolutional layers are stacked on one another. [24] In the experiment, we analysed three different stack numbers n (i.e., 2, 3, 4) from transferring learning cases in the TCN predictor.

(ii) Filter size

Similarly, we analysed three different filter sizes k (i.e., 8, 16, 32) for the receptive field size of the TCN for the designed TCN-based prediction model using iterations of the Q-learning computation.

(iii) Dilatation coefficient

As described before, the residual block (d) was used between each layer in the TCN to accelerate convergence and enable the training of deeper models. From [14], we analysed three different sets of combinations for the dilatation coefficient d (i.e., 1, 2, 4; 1, 2, 4, 8; 1, 2, 4, 8, 16) in the receptive field size of the model input.

(iv) Activation function

We analysed two major types of nonlinear activation functions, that is, Norm_relu and Tanh*Sigmoid activation functions used in Wavenet. Then, we calculated the corresponding convergence error of the cost function for the two activation functions.

(v) Optimiser

To minimise loss during model training, an optimiser was adopted to improve the predictive accuracy of the model by adjusting the filter weights. We assessed three popular optimisers for the TCN model: Adam, SGD, and RMSprop.

In summary, the computation process for the proper selection of TCN model parameters is as follows:

1. Initialize the parameters for the model.

The architectural components of the proposed TCN model from GitHub [24] were regarded as the initial values and are listed in Table 5.

Table 5. Initial parameters for the proposed TCN model.

Parameter	Stacks	Filter	Dilatations	Activation Function	Optimizer
Initial value	2, 3, 4	8, 16, 32	[1, 2, 4] [1, 2, 4, 8] [1, 2, 4, 8, 16]	norm_relu, wavenet	Adam, SGD, RMSprop

2. Repeat the computation of Algorithm 1 for the developed model until $R < 0.1$. Select the appropriate parameter values for the five model parameters in Table 5.
   a. The parameter Q table is iterated until the termination condition.
   b. Compute Reward R using Equation (14).
   c. Update the Q-table using Equation (15) as shown in Table 6.

Table 6. A record of the Q-learning computation process for selection of optimizer.

R Table

t state		t + 1		t + 2		t + 3	Adam	RMSprop	SGD
Adam	0	Adam	0	Adam	0	Adam	0	−53.60	−6643.26
				RMSprop	−53.60	RMSprop	53.60	0	−6589.66
				SGD	−6643.26	SGD	6643.26	6589.66	0
		RMSprop	−53.60	Adam	53.60	Adam	0	−53.60	−6643.26
				RMSprop	0	RMSprop	53.60	0	−6589.66
				SGD	−6589.66	SGD	6643.26	6589.66	0
		SGD	−6643.26	Adam	6643.26	Adam	0	−53.60	−6643.26
				RMSprop	6589.66	RMSprop	53.60	0	−6589.66
				SGD	0	SGD	6643.26	6589.66	0

Q Table

1nd literation

t state		t + 1 state	
Adam	0	Adam	0
		RMSprop	0
		SGD	0

2st iteration

t state		t + 1		t + 2	
Adam	0	Adam	0	Adam	0
				RMSprop	0
				SGD	0
		RMSprop	−37.52	Adam	0
				RMSprop	0
				SGD	0
		SGD	−4650.28	Adam	0
				RMSprop	0
				SGD	0

3rd iteration

t state		t + 1		t + 2		t + 3	Adam	RMSprop	SGD
Adam	0	Adam	0	Adam	0	Adam	0	0	0
				RMSprop	−37.52	RMSprop	0	0	0
				SGD	−4650.28	SGD	0	0	0
		RMSprop	−25.14	Adam	37.52	Adam	0	0	0
				RMSprop	0	RMSprop	0	0	0
				SGD	−4612.76	SGD	0	0	0
		SGD	−3115.69	Adam	4650.28	Adam	0	0	0
				RMSprop	4612.76	RMSprop	0	0	0
				SGD	0	SGD	0	0	0

In the experiment, the architecture parameters were determined by computing the Q-table (Figure 6). Finally, the architecture parameters were selected according to the results of the Q-learning computation and are listed in Table 7.

Table 7. Architecture parameter settings in the TCN model.

Hyperparameter	Stacks	Filter	Dilatations	AF	Optimizer
Optimal value	4	32	[1, 2, 4, 8]	norm_relu	RMSprop

AF = activation function, loss in object function = root_mean_squared_error, max_Epoch = 100.

Then, the selected hyperparameters for the TCN were used to draw comparisons and calculate the predictive precision of the proposed scheme. As shown in Figure 8, we evaluated the accuracy of the model in terms of prediction ability to examine the expected outcome and whether the appropriate model parameters were selected. The optimal hyperparameters for the TCN can be determined through the error value and error reduction speed of the loss function. Specifically, the model parameters may require adjustment if the convergence error decrease is not smooth. From Figure 8, the TCN accuracy for wind power prediction increased with a decrease in the iteration of the training data, leading to achievement of stable convergence.

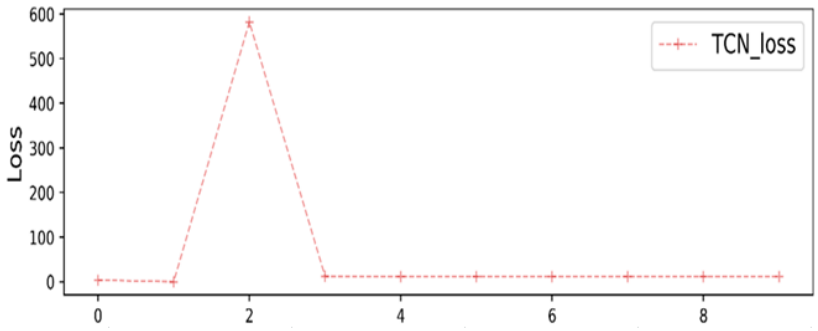

Figure 8. Convergence error training of the TCN model.

To compare predictive accuracy of DLNs for sequence data processing, three deep neural models, namely RNN, LSTM, and GRU, were incorporated to examine the proper setting of the model parameters. First, the initial parameters for the three models from the transfer learning were regarded as the initial values and are listed in Table 8.

Table 8. Initial parameters for the RNN, LSTM, and GRU models.

Model \ Parameter	Output Unit	Optimizers	Learning Rate (lr)
RNN	8, 16, 32	Adam, SGD, RMSprop	0.05, 0.02, 0.01
LSTM	8, 16, 32	Adam, SGD, RMSprop	0.05, 0.02, 0.01
GRU	8, 16, 32	Adam, SGD, RMSprop	0.05, 0.02, 0.01

In the experiment, the architecture parameters were determined based on the results of the Q-learning computation process. Tables 9 and 10 and Figures 9 and 10 show the corresponding convergence errors of the cost function for the RNN, LSTM, GRU and TCN prediction models for wind power forecasting 72 and 168 h ahead. Finally, the hyperparameters for the model were selected based on the results of the Q-learning computation and are listed in Table 11.

Once the optimal parameters for the RNN, LSTM, and GRU models were selected, three predictive models were validated with the same experimental data to conduct wind power forecasting 72–168 h ahead using the model parameters selected, as listed in Tables 7 and 11.

Table 9. Performance of wind power forecasting (72 h ahead).

DLN Models	Training Set	Test Set
	MAPE (%)	MAPE (%)
RNN	6.2	2.2
GRU	1.7	1.3
LSTM	4.1	0.4
TCN	0.9	0.1

Table 10. Performance of wind power forecasting (168 h ahead).

DLN Models	Training Set	Test Set
	MAPE (%)	MAPE (%)
RNN	5.7	2.8
GRU	2.0	1.3
LSTM	4.0	1.3
TCN	1.1	0.3

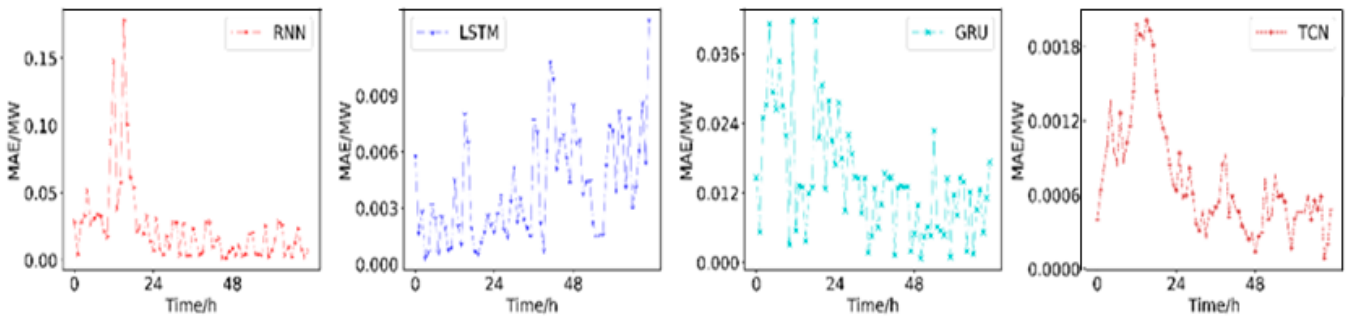

Figure 9. The prediction errors for the RNN, LSTM, GRU and TCN models (72 h ahead).

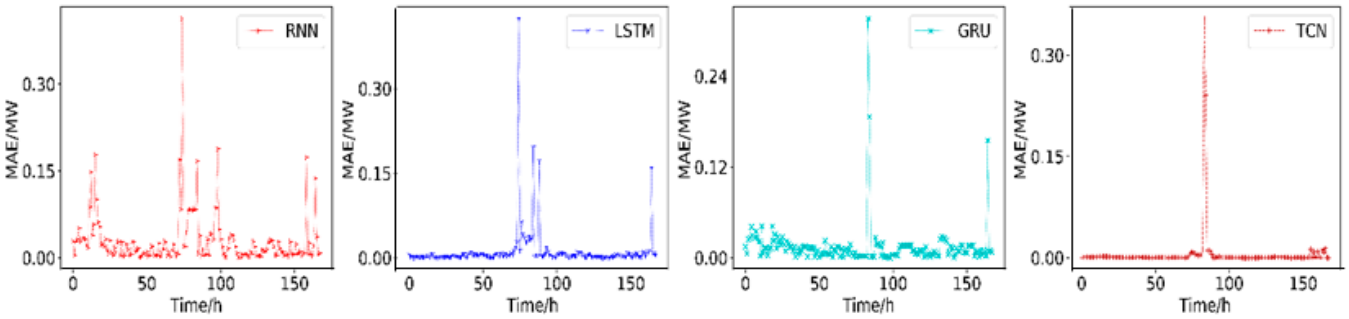

Figure 10. The prediction errors for the RNN, LSTM, GRU and TCN models (168 h ahead).

Table 11. Selected parameters for the RNN, LSTM, and GRU models.

Model \ Parameter	Output Unit	Optimizers	Learning Rate (lr)
RNN	8	Adam	0.005
LSTM	32	SGD	0.001
GRU	32	Adam	0.005

Step 3. Model validation phase

In the experiment, 1-year historical data were used to train four DNNs for wind power data, and the prediction results for 72, 120, and 168 h ahead are shown in Figures 11–13, respectively. Figure 13 shows the accuracy associated with using four different prediction

models where the prediction error of the TCN mode for 168 h (7 d) ahead had the highest prediction accuracy, MAPE = 1.41%, followed by GRU mode MAPE = 3.72%, LSTM mode MAPE = 6.99%, and RNN mode MAPE = 28.18%.

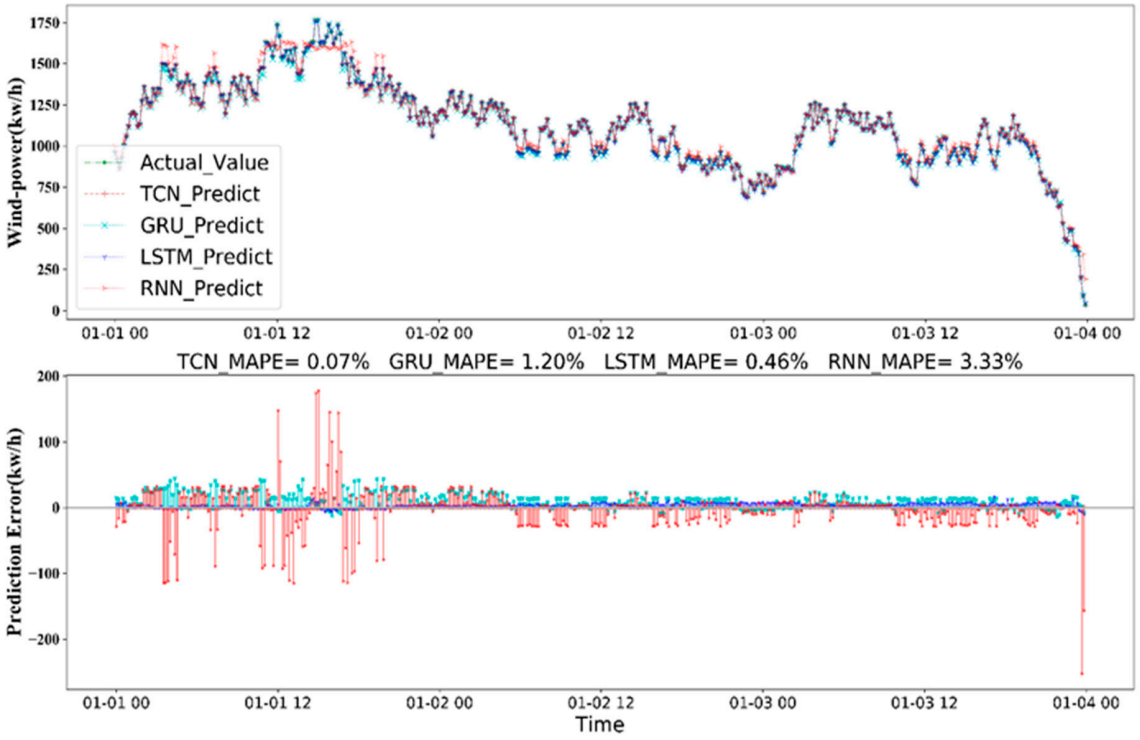

Figure 11. Experimental results for 72 h wind power forecasting.

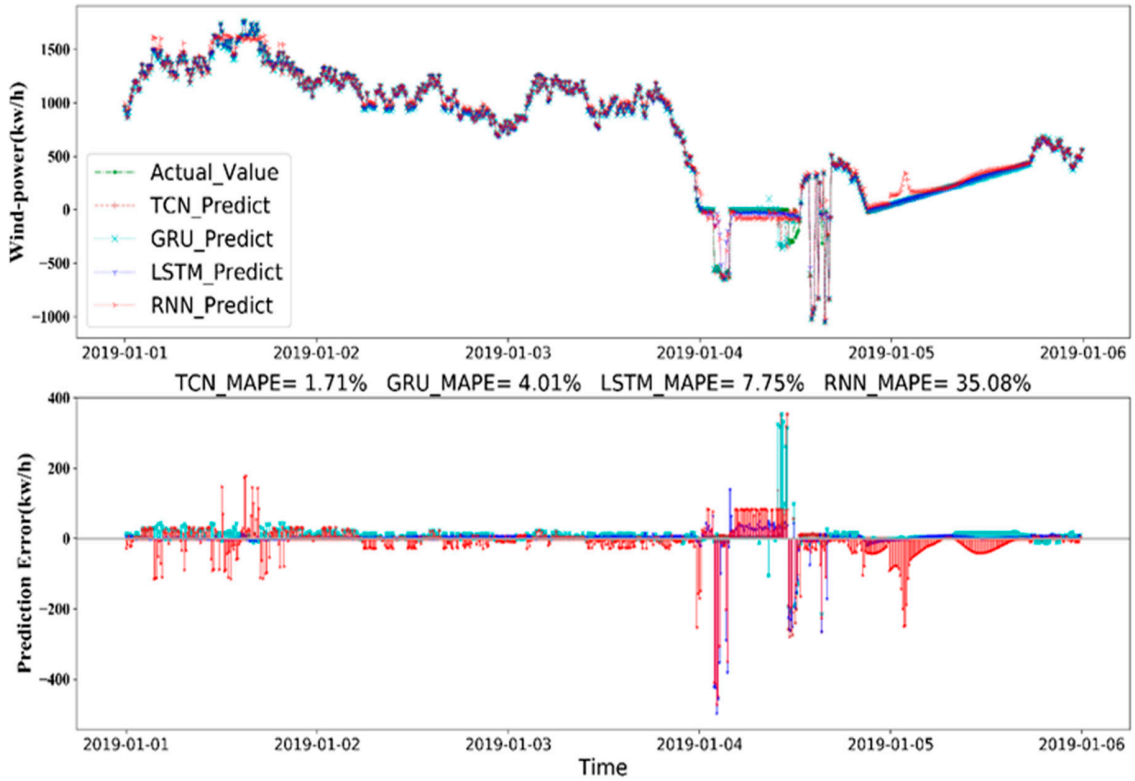

Figure 12. Experimental results for 120 h wind power forecasting.

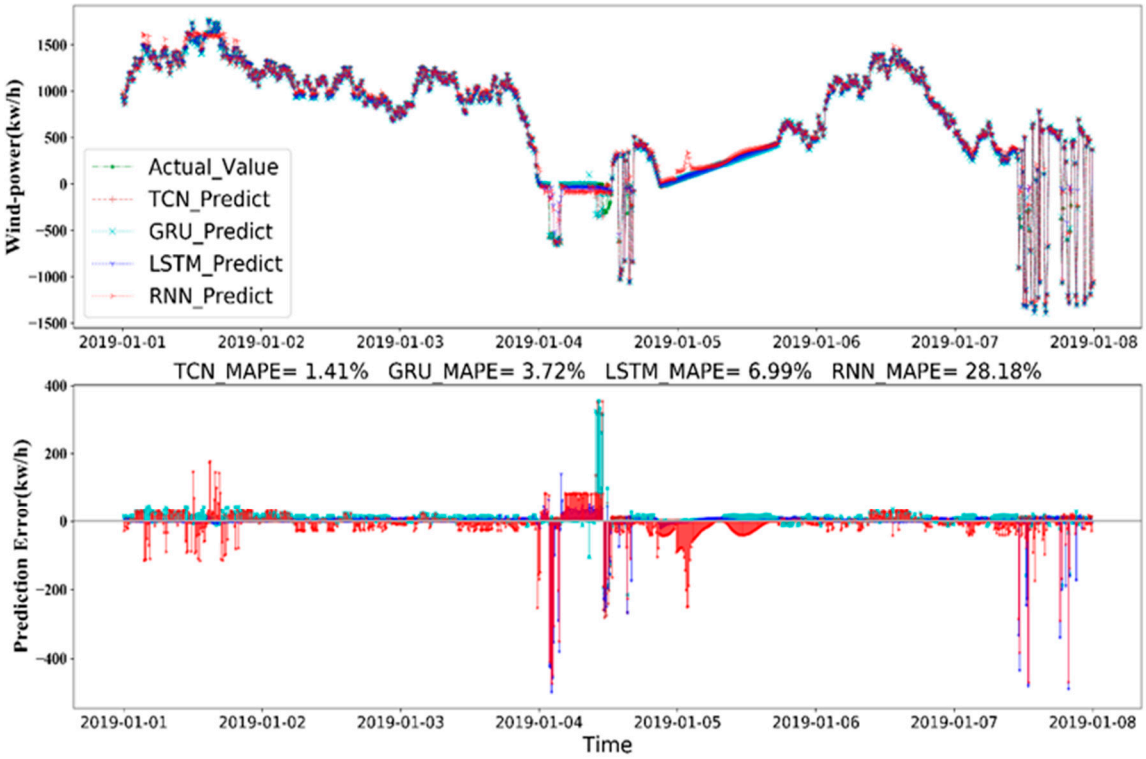

Figure 13. Experimental results for 168 h wind power forecasting.

5. Method Comparisons

As shown in Figure 14a, the precision of four prediction models was increased by the decreased convergence error of the cost function in each iteration of the model training. More concisely, the descending speed of error convergence for the cost function was rapid with the increasing number of iterations (epochs). After 60 iterations, the loss function tended to be stable when TCN and GRU prediction models were applied (Figure 14b). However, the errors for the RNN and LSTM models did not converge slowly. Notably, the convergence error of the TCN model decreased more than that of the GRU model close to the fifteenth epoch, and the convergence error decreased steadily as the number of iterations increased. The experimental results revealed that the prediction error of the TCN model decreased the most steadily among the four models, followed by GRU and LSTM.

To discriminate the performance of four prediction models in long-term prediction, different lengths of the prediction period based on the same amount of training data were tested. Notably, the experiment varied the lengths of the prediction periods set to examine the accuracy of the convergence error in practice; Table 12 lists the prediction error (i.e., MAPE), and the output results of the modules were sorted for a test set of distinct lengths of prediction periods with the same data size.

Experimental results show that the prediction error increased as the length of the prediction period increased. In other words, the prediction error of wind power output increased with increasing length of prediction period. Particularly, the prediction error of the TCN-based model remained below 2% with increasing length of the prediction period, and the forecast errors for 72, 120, and 168 h were 0.07%, 1.71%, and 1.41%, respectively. In summary, the average prediction error of the proposed TCN-based model was approximately 1.063% based on 1-year historical data in three different predictive periods. In other words, the TCN model achieved highly precise prediction accuracy with consistent and stable results using selected parameters from the Q-learning algorithm.

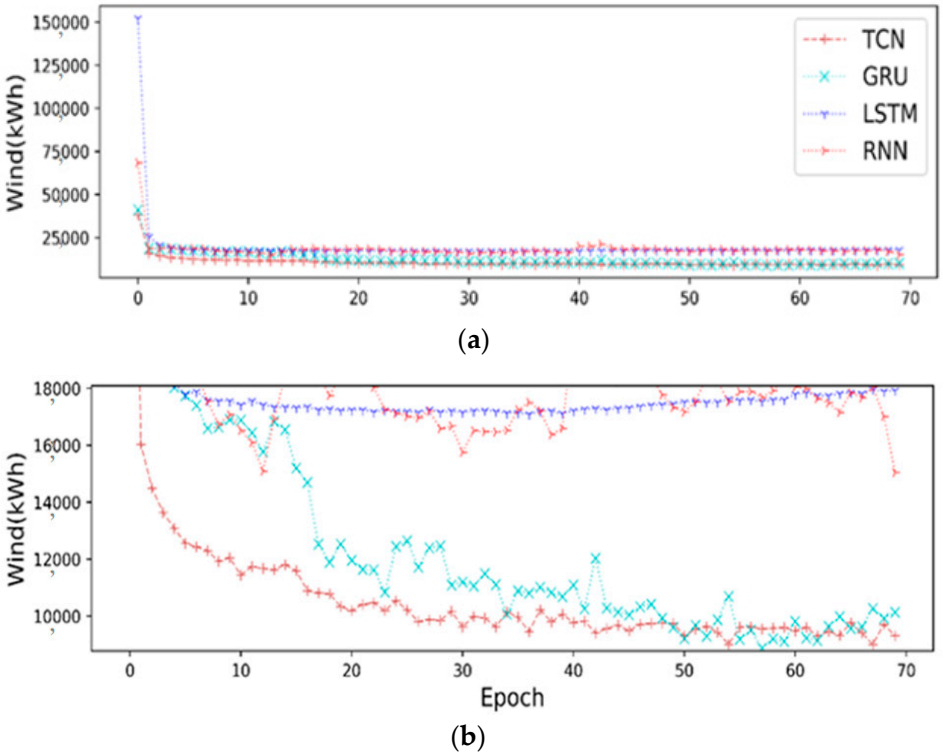

Figure 14. (**a**) Convergence error of the cost function for the RNN, LSTM, GRU and TCN models. (**b**) Partial enlarged image of (**a**).

Table 12. Performance comparison of four DNN models with prediction error for wind power forecasting.

Performance / Period	Excellent	Good	Medium	Bad
72 h (3 days)	TCN (0.07%)	LSTM (1.20%)	GRU (1.46%)	RNN (3.33%)
120 h (5 days)	TCN (1.71%)	GRU (4.01%)	LSTM (7.75%)	RNN (35.08%)
168 h (7 days)	TCN (1.41%)	GRU (3.72%)	LSTM (6.99%)	RNN (28.18%)

Compared to other studies in wind power forecasting, such as with LSTM [17], LSTM with grid search (LSTM-grid) [18], bidirectional LSTM with grid search (BiLSTM-grid) [18], GRU [17], TCN [17], and the ensemble learning-based AAA aggregation model [19], the detailed comparison information for distinct approaches is listed in Table 13. Notably, the project of the BSI electric power company used an AAA aggregation model based on an ensemble-based learning algorithm for 120 h forecasts by aggregating numerous deep learning models in different climate scenarios and reached approximately 2% prediction error in 2016. Overall, the proposed TCN-based model with Q-learning algorithm provides a lower prediction error with higher prediction accuracy than those of earlier deep learning schemes such as LSTM, BiLSTM, TCN and AAA aggregation models [32] in studies of wind power forecasting.

Table 13. Performance comparison with other real projects for wind power forecasting.

Approach	Prediction Cycle	Prediction Intervals	Training Prediction Error	Testing Prediction Error
LSTM [17]	Short term	Within 1 h	6.30%	5.40%
LSTM-grid [18]	Short term	Within 1 h	8.65%	8.66%
BiLSTM-grid [18]	Short term	Within 1 h	8.66%	8.64%
GRU [17]	Short term	Within 1 h	6.00%	5.00%
TCN [17]	Short term	Within 1 h	2.80%	1.60%
AAA aggregation model, 2016 [32]	Medium term	Within 120 h	2.00%	2.00%
Proposed model	Medium term	Within 168 h	1.41%	1.41%

6. Conclusions

This paper presented a TCN-based model based on a casual convolution architecture with a Q-learning algorithm to efficiently learn the correlations between meteorological features and wind power generation. The experimental results show that the Q-learning algorithm efficiently helps data engineers in determining appropriate parameters for temporal convolutional networks. Additionally, the TCN has a great capability for feature extraction of long-term sequence data and retains higher prediction accuracy than GRU and LSTM. Overall, the proposed TCN-based approach outperformed GRU, LSTM, and RNN in our experiments for wind power forecasting.

Author Contributions: Conceptualization, P.W.; methodology H.-C.L. and K.-M.C.; resources, P.W.; formal analysis, W.-H.L. and Z.-Y.Y.; data curation, W.-H.L. and Z.-Y.Y.; writing—original draft, W.-H.L.; writing—review and editing, P.W. and K.-M.C.; software, H.-C.L.; validation Y.-H.L., H.-C.L. and W.-H.L.; visualization, H.-C.L., Z.-Y.Y. and Y.-H.L.; project administration, P.W.; funding acquisition, P.W. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Ministry of Science and Technology of Taiwan under grant no. MOST 110-2410–H-168-003 and the green energy technology research center on the featured areas research center program within the framework of the higher education sprout project from Ministry of Education (MOE) of Taiwan under Grant No. MOE 2000-109CC5-001.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data presented in this study are available upon request from the corresponding author.

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

Nomenclature

a	the action to the current state s
a'	the action to the new state s'
b	bias vector parameters that need to be learned during training
b_x^l	bias vector corresponding to the l-th hidden layer
c	the memory state
$\widetilde{c}$	the candidate (memory) cell state
d	dilation factors
f	the forget gate
$F_d(.)$	the dilated convolution function of d-factor
h	the hidden state
k	filter size
$\mathrm{L}(\theta)$	cost function
o	the output gate

$Q(s,\ a\ ;\theta)$	the quantity of a state-action pair (s, a) for current state s
$Q_k(s', a'; \theta)$	the quantity of a state-action pair (s, a) for new state s'
$r_k(s, a; \theta)$	the reward value of the current state s,
s	the current state
s'	a new state after performing action a
x_t	input vector to the neuron
x_t^l	the value of the neuron of l-th hidden layer at time t
W	the weights that need to be learned during training
W_x^l	the weights corresponding to the l-th hidden layer
α	the learning rate
σ	the activation function
γ	the discount factor

References

1. Cadenas, E.; Rivera, W.; Campos-Amezcua, R.; Heard, C. Wind speed prediction using a univariate ARIMA model and a multivariate NARX model. *Energies* **2016**, *9*, 109. [CrossRef]
2. Liu, H.; Chen, C.; Lv, X.; Wu, X.; Liu, M. Deterministic wind energy forecasting: A review of intelligent predictors and auxiliary methods. *Energy Convers. Manag.* **2019**, *195*, 328–345. [CrossRef]
3. Jiang, P.; Liu, Z.; Niu, X.; Zhang, L. A combined forecasting system based on statistical method, artificial neural networks, and deep learning methods for short-term wind speed forecasting. *Energy* **2021**, *217*, 119361. [CrossRef]
4. Donadio, L.; Fang, J.; Porté-Agel, F. Numerical weather prediction and artificial neural network coupling for wind energy forecast. *Energies* **2021**, *14*, 338. [CrossRef]
5. Lin, W.H.; Wang, P.; Chao, K.M.; Lin, H.C.; Yang, Z.Y.; Lai, Y.H. Wind power forecasting with deep learning networks: Time-series forecasting. *Appl. Sci.* **2021**, *11*, 10335. [CrossRef]
6. Perera, A.T.D.; Kamalarubanb, P. Applications of reinforcement learning in energy systems. *Renew. Sustain. Energy Rev.* **2021**, *137*, 110618. [CrossRef]
7. Firat, O.; Oztekin, I. Learning Deep Temporal Representations for Brain Decoding. In Proceedings of the First International Workshop, MLMMI 2015, Lille, France, 11 July 2015; pp. 25–34.
8. Donahue, J.; Hendricks, L.A.; Rohrbach, M.; Venugopalan, S.; Guadarrama, S.; Saenko, K.; Darrell, T. Long-term Recurrent Convolutional Networks for Visual Recognition and Description. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Boston, MA, USA, 7–12 June 2015; pp. 2625–2634.
9. Sainath, T.N.; Vinyals, O.; Senior, A.; Sak, H. Convolutional Long Short-Term Memory, Fully Connected Deep Neural Networks. In Proceedings of the 2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), South Brisbane, Australia, 19–24 April 2015; pp. 4580–4584.
10. Ravanelli, M.; Brakel, P.; Omologo, M.; Bengio, Y. Light Gated Recurrent Units for Speech Recognition. *IEEE Trans. Emerg. Top. Comput. Intell.* **2018**, *2*, 92–102. [CrossRef]
11. Lea, C.; Flynn, M.D.; Vidal, R.; Reiter, A.; Hager, G.D. Temporal Convolutional Networks for Action Segmentation and Detection. In Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), CVPR2017, Honolulu, HI, USA, 21–26 July 2017; pp. 1003–1012.
12. Richter, J. dida—Temporal Convolutional Networks for Sequence Modeling. Available online: https://dida.do/blog/temporal-convolutional-networks-for-sequence-modeling (accessed on 9 March 2022).
13. Bai, S.; Kolter, J.Z.; Koltun, V. An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling. *arXiv* **2018**, arXiv:1803.01271.
14. Varma, S.; Das, S. Deep Learning: Training Neural Networks. Available online: https://srdas.github.io/DLBook/HyperParameterSelection.html (accessed on 9 March 2022).
15. Oh, E.; Wang, H. Reinforcement-learning-based energy storage system operation strategies to manage wind power forecast uncertainty. *IEEE Access* **2020**, *8*, 20965–20976. [CrossRef]
16. Sharma, R.; Shikhola, T.; Kohli, J.K. Modified fuzzy Q-learning based wind speed prediction. *J. Wind Eng. Ind. Aerodyn.* **2020**, *206*, 104361. [CrossRef]
17. Zhu, R.; Liao, W.; Wang, Y. Short-term prediction for wind power based on temporal convolutional network. *Energy Rep.* **2020**, *6*, 424–429. [CrossRef]
18. Neshat, M.; Nezhad, M.M.; Abbasnejad, E.; Mirjalili, S.; Tjernberg, L.B.; Garcia, D.A.; Alexander, B.; Wagner, M. A deep learning-based evolutionary model for short-term wind speed forecasting: A case study of the Lillgrund offshore wind farm. *Energy Convers. Manag.* **2021**, *236*, 114002. [CrossRef]
19. Li, D.; Jiang, F.; Chen, M.; Qian, T. Multi-step-ahead wind speed forecasting based on a hybrid decomposition method and temporal convolutional networks. *Energy* **2022**, *238*, 121981. [CrossRef]
20. Yan, J.; Mu, L.; Wang, L.; Ranjan, R. Temporal convolutional networks for the advance prediction of ENSO. *Sci. Rep.* **2020**, *10*, 8055. [CrossRef] [PubMed]

21. Lee, W.Y.; Park, S.; Sim, K.B. Optimal hyperparameter tuning of convolutional neural networks based on the parameter-setting-free harmony search algorithm. *Int. J. Light Electron Opt.* **2018**, *172*, 359–367. [CrossRef]
22. Wu, J.; Chen, X.Y.; Zhang, H.; Xiong, L.D.; Lei, H.; Deng, S.H. Hyperparameter optimization for machine learning models based on Bayesian optimization. *J. Electron. Sci. Technol.* **2019**, *17*, 26–40.
23. Amirabadi, M.A.; Kahaei, M.H.; Nezamalhosseini, S.A. Novel suboptimal approaches for hyperparameter tuning of deep neural network. *Phys. Commun.* **2020**, *41*, 101057. [CrossRef]
24. Lokku, G.; Reddy, G.H.; Prasad, M.G. OPFaceNet: Optimized Face Recognition Network for noise and occlusion affected face images using Hyperparameters tuned Convolutional Neural Network. *Appl. Soft Comput.* **2022**, *117*, 108365. [CrossRef]
25. Nematzadeh, S.; Kiani, F.; Torkamanian-Afshar, M.; Aydinc, N. Tuning hyperparameters of machine learning algorithms and deep neural networks using metaheuristics: A bioinformatics study on biomedical and biological cases. *Comput. Biol. Chem.* **2022**, *97*, 107619. [CrossRef]
26. Zoph, B.; Le, Q.V. Neural architecture search with reinforcement learning, optimal transport. In Proceedings of the 32nd International Conference on Neural Information Processing Systems, Montreal, QC, Canada, 3–8 December 2018; pp. 2016–2025.
27. Iranfar, A.; Zapater, M.; Atienza, D. Multi-agent reinforcement learning for hyperparameter optimization of convolutional neural networks. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **2021**, *41*, 1034–1047. [CrossRef]
28. Jalali, S.M.J.; Osorio, G.J.; Ahmadian, S.; Lotfi, M.; Campos, V.M.A.; Shafie-Khah, M.; Khosravi, A.; Catalao, J.P.S. New Hybrid Deep Neural Architectural Search-Based Ensemble Reinforcement Learning Strategy for Wind Power Forecasting. *IEEE Trans. Ind. Appl.* **2022**, *58*, 15–27. [CrossRef]
29. Rémy, P. Keras-tcn, GitHub. Available online: https://github.com/philipperemy/keras-tcn (accessed on 12 February 2022).
30. Rijsdijk, J.; Wu, L.; Perin, G.; Picek, S. Reinforcement Learning for Hyperparameter Tuning in Deep Learning-based Side-channel Analysis. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* **2021**, *3*, 677–707. [CrossRef]
31. He, Y.; Zhao, J. Temporal convolutional networks for anomaly detection in time series. *J. Phys. Conf. Ser.* **2019**, *1213*, 042050. [CrossRef]
32. UK Power Networks. *KASM SDRC 9.3: Installation of Forecasting Modules*; UK Power Networks: London, UK, 2016.