HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

School of Information and Comunication Technology

*****

GRADUATION THESIS

Topic:

Thesis title

Author: **Trung Tran**

Supervisor: **PhD. Binh Minh Nguyen**

Hanoi, 12/2019

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Problems

1.2 Summary

Chapter 2

Materials and background

2.1 Cloud Computing

2.2 Auto-scaling problem

2.3 Well-known machine learning models for Auto-scaling in cloud computing

Recent developments in cloud computing including resource management have resulted in a significant interest in resource usage prediction . Various methods have been proposed for solving this problem with different aspects, objectives and applications [1]. In this section, we focus on several Artificial Neural Network (ANN) models that are used for tackling the time-series characteristic in resource usage forecast in cloud computing environment. Deep Feed-forward Neural Network, also called Feed-forward Neural Network (FFNN) are the quintessential deep learning models. The goal of all FFNN is to approximate some functions f^*. In Regression problems, $y = f^*(x)$ maps an input x to a value y. A feed-forward network defines a mapping $y = f(x, \theta)$ and learns the value of the parameters θ that result in the best function approximation. [10]. These models are called feed-forward because information flows through the function being evaluated from x, through the intermediate computations used to define f, and finally to the output y. There are no feedback connections in which outputs of the model are fed back into itself. When feed-forward neural networks are extended to include feedback connections, they are called recurrent neural networks, which will be discussed in 2.3.2.

In general, Multi-Layer Perceptrons (MLPs) models contain several disparate layers. The first layer is input layer taking information x as input for the network. The last layer is called output layer, whose value is the result of y with input x. The layers between the input and output layers are hidden layers. The structures of hidden layers are extremely diverse, varying from model to model. As presented in Fig. 2.1, a hidden layer of a simple FFNN is a group of neurons with no connection to each other, while in Recurrent Neural Networks (RNN), and Convolution Neural Network (CNN) hidden layer is a recurrent layer, and convolution layer respectively.

The Deep Neural Networks that are applied for Time series prediction will have input neurons presenting the historical data. The models utilize information from data in the past for forecasting future data. Input data presented as $x_1, x_2, ..., x_t$ is considered as historical values up to time t, which is used to predict the value at the time $t + 1$. In other words, Deep Neural Networks will learn from data and approximate a function transforming the historical data up to time t to the data at the time $t + 1$ as follows:

$$x(t+1) = y = f(x_1, x_2, ..., x_t) \tag{2.1}$$

In this section, we summarize several Deep Neural Network models, which are widely used for time series forecasting. They are simple Multi-Layer Perceptrons (MLPs), Cascade Forward Neural Network (CFNN) and Recurrent-based Neural Network including traditional Recurrent Neural Network (RNN), Long-Short Term Memory (LSTM) and Gated Recurrent Units (GRU). Each method will be presented below with brief ideas and mathematical formulas.

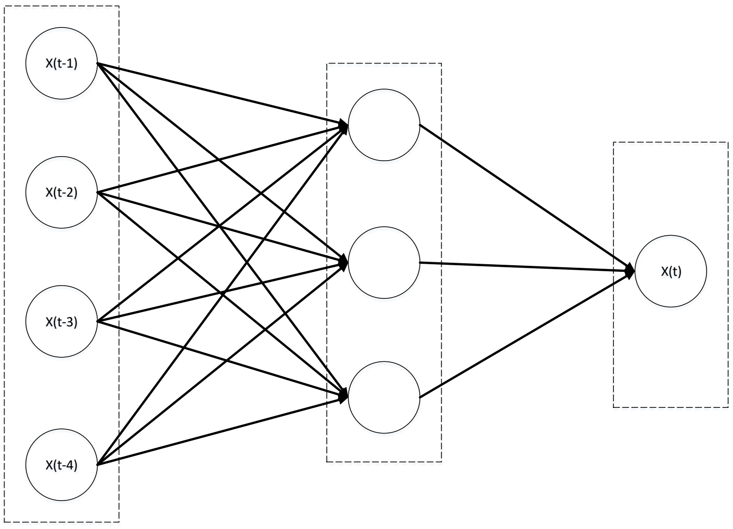

Figure 2.1: An example of MLPs model in time-series prediction.

2.3.1 Multi-Layer Perceptrons (MLPs)

The additional layers added between input layer and output layer make network architecture contain hidden layers and called Multi-Layer Perceptrons (MLPs). The input data is fed through input layer to hidden layers in the weighted form. The information from input data X is distributed to the neurons in hidden layers and then processed by an activation function. The activation function in hidden layers are non-linear function playing a role as a transfer function, helping MLPs learn non-linear characteristics of the data. The information after being processed by hidden layers then are sent to output layer in the weighted sum, and also go through an activation function as well, creating the output value y. MLPs model is used in predicting time series data [2], [14]. Fig. 2.1 shows a MLPs with a 4-neuron input layer and one output layer. In general, the mathematical equation of the MLPs architecture can be written as follows:

$$H = f_h(W_h^T X + b_h) \tag{2.2}$$

$$y = O = f_o(W_o^T H + b_o \tag{2.3}$$

Where X is the input data, H and O are the information after being fed through the hidden and output layers. W_h, b_h and W_o, b_o are weights and biases, while f_h and f_o are activation functions of hidden layer and output layer, respectively.

2.3.1.1 Cascade Forward Neural Network (CFNN)

The main difference between CFNN and MLPs is that in CFNN, perceptron connection is added directly between neurons in input layer and output layer, while in MLPs, that connection is indirect through the hidden layer. The output layer of CFNN perceives both transformed information that is output of hidden layer, and the raw information from input data. This Deep Neural Network model was first used for forecasting monthly palm oil price in the Europe market in [23]. The architecture of CFNN with 4-neuron input layer is illustrated in Fig. 2.2, and the mathematical formulas for CFNN model are presented as follows:

$$H = f_h(W_h^T X + b_h) \tag{2.4}$$

$$C = f_c(W_c^T X) + b_c \tag{2.5}$$

$$y = O = f_o(W_o^T H + b_o) + C \tag{2.6}$$

where f_c is the activation function from the input layer to output layer, C is the output value of f_c, and W_c, b_c are weights and biases of the connection, respectively.

2.3.2 Recurrent Neural Network (RNN)

Recurrent neural networks (RNNs) are dynamical systems that are specifically designed for temporal problems, as they have both feed-back and feed-forward connections (Fig. 2.3.2). RNN remembers the past and its decisions are influenced by what it has learned from the past. RNNs can take one or more input vectors and produce one or more output vectors and the output(s) are influenced not just by weights applied on inputs like a regular MLPs, but also by a state vector rep-

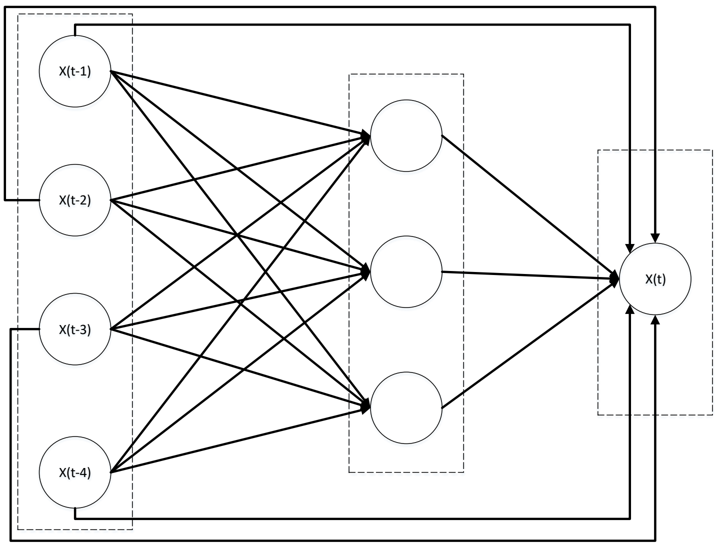

Figure 2.2: An example of CFNN model in time-series prediction.

resenting the context based on prior input(s)/output(s), so the same input could produce a different output depending on previous inputs in the series. For that reason, RNN is one of the most popular models being used for modeling time series data [24], [6], [4]. There are two popular and efficient RNN models that work really well: Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) which are discussed below.

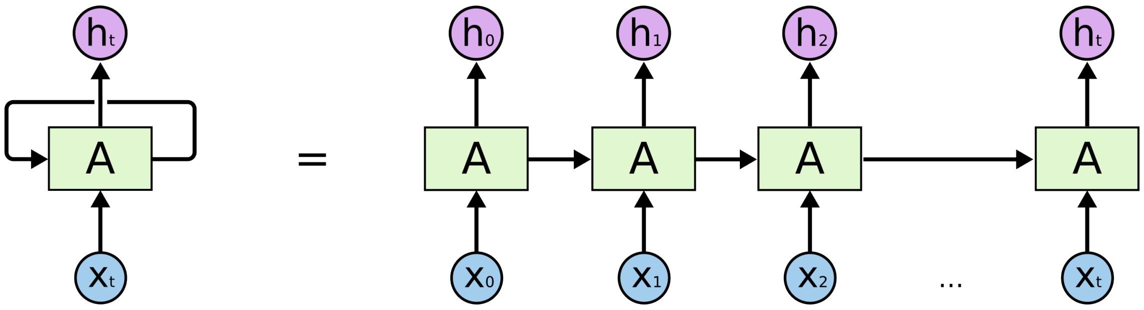

Figure 2.3: Traditional RNN architechture. Source: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

2.3.3 Short Term Memory (LSTM)

Long short-term memory (LSTM) [12] is a special kind of RNN created for learning long-term dependencies. LSTM units have 3 gates managing the contents of the memory. These gates are simple logistic functions of weighted sums, where the weights might be learnt by backpropagation. It means that, even though it seems a bit complicated, the LSTM perfectly fits into the neural network and its training process. With combining a forget gate in LSTM units, LSTM is capable to determine what it needs to remember and forget, so LSTM can work very well with dependent data , especially with time series data [9], [11], [8]. The architecture of LSTM units is illustrated in Fig. 2.4, and its mathematical model is briefly described as follows: The input gate (2.7) and the forget gate (2.8) manage the cell state (2.10), which is the long-term memory. The output gate (2.9) produces the output vector or hidden state (2.11), which is the memory focused for use. This memory system enables the network to remember for a long time, which was badly missing from vanilla recurrent neural networks.

$$i_t = sigmoid(W_i x_t + U_i h_{t-1} + b_i) \tag{2.7}$$

$$f_t = sigmoid(W_f x_t + U_f h_{t-1} + b_f) \tag{2.8}$$

$$o_t = sigmoid(W_o x_t + U_o h_{t-1} + b_o) \tag{2.9}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot tanh(W_c x_t + U_c h_{t-1} + b_c) \tag{2.10}$$

$$h_t = o_t \odot tanh(c_t) \tag{2.11}$$

2.3.4 Gated Recurrent Units (GRU)

Gated recurrent unit (GRU) [5] is essentially a simplified LSTM. Different form LSTM, GRU uses two gated called update gate and reset gate. Basically, these gates are two vectors managing what information should be passed to the output. In GRU, its gates can be trained to keep information from long ago, without washing it through time or remove information which is irrelevant to the prediction. It has the exact same role as LSTM in the network. The main difference is in the number of gates and weights - GRU is somewhat simpler. Like LSTM, GRU is also a widely

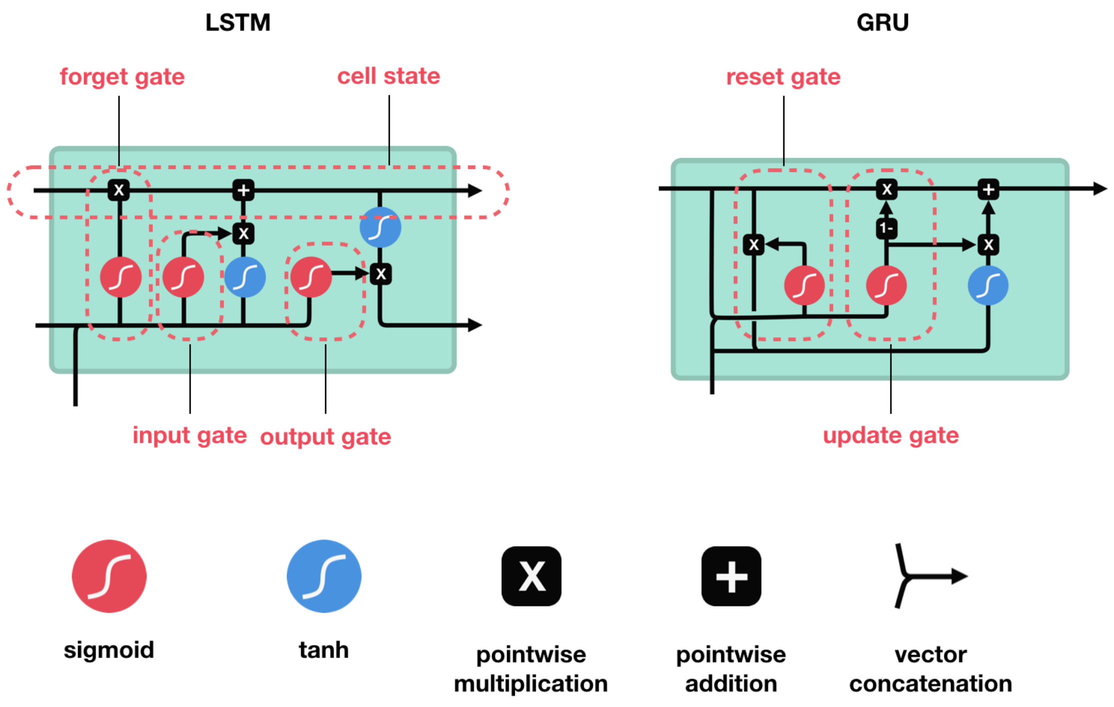

Figure 2.4: LSTM and GRU architechture. Source: https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21

chosen solution for time series forecasting such as in [15] and [3] The srtucture of GRU units is presented in Fig. 2.4 following the mathematical model as below:

The update gate 2.12 controls the information flow from the previous activation, and the addition of new information as well 2.14, while the reset gate 2.13 is inserted into the candidate activation. Overall, it is pretty similar to LSTM. From these differences alone, it is hard to tell, which one is the better choice for a given problem.

$$z_t = sigmoid(W_z x_t + U_z h_{t1} + b_z) \tag{2.12}$$

$$r_t = sigmoid(W_r x_t + U_r h_{t1} + b_r) \tag{2.13}$$

$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot tanh(W_h x_t + U_h(r_t h_{t-1}) + b_h) \tag{2.14}$$

2.4 Fundamental knowledge

2.4.1 Artificial Neural Network (ANN)

2.4.1.1 Activation functions

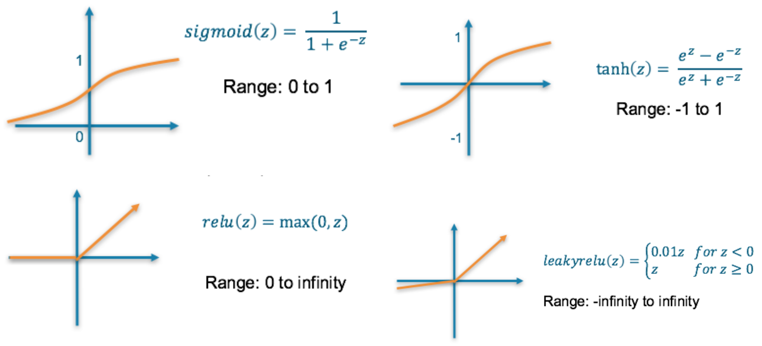

Figure 2.5: Activation functions. Source: https://medium.com/@srnghn/deep-learning-overview-of-neurons-and-activation-functions-1d98286cf1e4

Activation functions also known as transfer functions are used to map input nodes to output nodes in certain fashion. Activation functions are extremely important for an ANN to learn and figure out features and characteristics of data which need a non-linear transformation to become outputs. There are the four most popular functions used in Deep Learning, their names are Sigmoid, Hyperbolic Tangent (Tanh), Rectified Linear Units (ReLU), Leaky ReLU and Exponential Linear Unit (ELU) (See in Fig. 2.5).

2.4.1.1.1 Sigmoid function

: takes a number as input and returns a value in $[0, 1]$.

$$f(x) = \frac{1}{1+e^x}$$

2.4.1.1.2 Hyperbolic Tangent (Tanh) function:

$f(x) = \frac{2}{1+e^{-2x}} = 2\sigma(2x) - 1$ is also like Sigmoid function but better, the range of the output from Tanh funtion is in $[-1, 1]$.

$$f(x) = \frac{2}{1+e^{-2x}} = 2\sigma(2x) - 1$$

2.4.1.1.3 Rectified Linear Unit (ReLU) function:

is a function having threshold at 0 value. It helps accelerate the training process in ANN, so it is used in almost all the complicated deep learning models such as RNNs and CNNs.

$$f(x) = max(0, x)$$

2.4.1.1.4 Leaky ReLU function:

is like ReLU function, but instead of setting thresholf value at 0, Leaky ReLU extends the domain to αx.

$$f(x) = \begin{cases} x, & \text{if } x > 1 \\ \alpha x, & \text{otherwise} \end{cases}$$

2.4.1.2 Loss functions

Neural Networks are trained by backpropagation algorithm, which updates the weights parameters of ANN according to a loss value. The loss value is calculated by a loss function, so loss functions are totally vital when an ANN model is built for learning information from data. These functions will essentially measure how poorly a model is performing by comparing what the model is predicting with the actual value it is supposed to output. Therefore, choosing a loss function that is appropriate for penalizing model effectively is one of the most important tasks while working with data. There are a number of loss functions for deep learning models, and each of them have its own pros and cons. The common loss functions that are widely used in time-series forecasting will be presented as below:

- **Mean Absolute Error (MAE)**:

$$MAE = \frac{\sum |e_t|}{N}$$

- **Sum Square Error (SSE)**:

$$SSE = \sum (e_t^2)$$

- **Mean Square Error (MSE)**:

$$MSE = \frac{\sum (e_t^2)}{N}$$

- **Root Mean Square Error (RMSE)**:

$$RMSE = \sqrt{MSE}$$

- **Mean Absolute Percentage Error (MAPE)**:

$$MAPE = \frac{1}{N} \sum |\frac{e_t}{y_t}|$$

Where N is the number of data points, y_t is the actual output value, d_t is the output value predicted by models, $e_t = d_t - y_t$ is the error value of the data point t.

Algorithm 1: Backpropagation algorithm applied for FFNN with 1 hidden layer

```
1  Initialize randomly weights' value $w_p$
2  repeat
3      Calculate output value(s) according to weights and input
4      for $j = 1$ to $h$ do
5          $H_j = \phi(\sum_i^n x_i * w_{ij}^{[1]} + b_{ij}^{[1]})$
6      end
7      $\widehat{y_j} = \phi(\sum_i^h H_i * w_j^{[2]} + b_j^{[2]})$
8      Calculate Loss value by loss function
9      $L(w) = loss(\widehat{y_j}, y_j)$
10     Backpropagating Loss to weights
11     $\triangle(w_{ij}^2) = \frac{\partial(L(w))}{\partial(w_{ij}^2)}$
12     $\triangle(w_{ij}^1) = \frac{\partial(L(w))}{\partial(w_{ij}^1)}$
13     Update weights' value
14     $w_{ij}^2 = w_{ij}^2 - \eta * \triangle(w_{ij}^2)$
15     $w_{ij}^1 = w_{ij}^1 - \eta * \triangle(w_{ij}^1)$
16 until Until convergence or the number of iterations is enough;
```

2.4.1.3 Backpropagation - the ANN Training Algorithm

Backpropagation algorithm is undoubtedly the most fundamental buiding block in an ANN. It It was first introduced in 1960s and almost 30 years later (1989) popularized by Rumelhart, Hinton and Williams in [19]. The algorithm is used to effectively train a neural network through a method called chain rule. In simple terms, after each forward pass through a network (propagation phase), backpropagation performs a backward pass while adjusting the model's parameters (weights and biases) (weights updating phase).

2.4.1.3.1 forward propagation phase

1. The input values will be fed into ANN through input layer, going forward to hidden layers, and finally to output layer, creating predicted output values. While propagation process, each layer uses its own activation function (sec. 2.4.1.1)

2. Error values are calculated by the loss function and propagated back to pre-

vious layers.

2.4.1.3.2 weights updating phase

1. Calculating gradients of loss function in weights and biases following the chain rule.
2. Updating weights and biases is done according to gradients' values

These two phases are repeated in each iteration during training. The algorithm will be stopped when the error from loss function reach a acceptable value or when the training iteration is large enough. The algorithm's pseudo code is presented in short in Algorithm 1.

2.4.2 Swarm Optimization Algorithms

2.4.2.1 Idea and motivation

2.4.2.2 Particle Swarm Optimization (PSO)

PSO [13] is a very first swarm-based optimization, which is the premise of many other algorithms proposed in recent years. It emulates the behaviors of birds, fish and so forth when they forage for food and communicate as a swarm. In PSO system, a swarm contains several candidate solutions (also known as particles), which coexist in the search space of the problem with D dimensions. The solution often cooperate and fly together to land on personal optimal positions. Over the course of time, the best personal position (its own best position in the past) of each particle and the global best position (the current best position of entire swarm) are recorded. The next position of a particle is updated based on the personal best (cognitive behavior) and the global best (social communication). With this approach, PSO combines local search (through personal best) with global search (through global best) to balance exploitation and exploration processes. PSO operation workflow is presented in Figure 2.6.

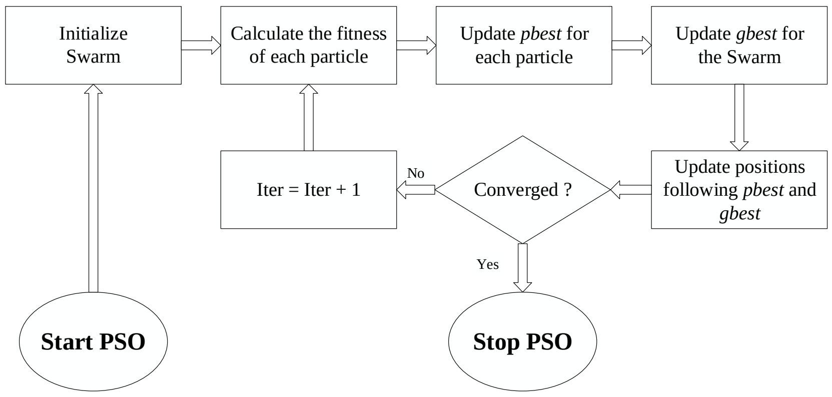
Initialize Swarm
Calculate the fitness of each particle
Update *pbest* for each particle
Update *gbest* for the Swarm
Update positions following *pbest* and *gbest*
Converged ?
No
Iter = Iter + 1
Yes
Start PSO
Stop PSO

Figure 2.6: PSO flowchart

Thus, each particle i in swarm is described by two properties: its velocity v_i and position x_i in the search space. In each iteration, they are updated following the equation:

$$v_i^{t+1} = \omega.v_i^t + c_1.r_1(p_i^t - x_i^t) + c_2.r_2(g^t - x_i^t) \tag{2.15}$$

$$x_i^{t+1} = x_i^t + v_i^t \tag{2.16}$$

where

ω	is inertia weight reduced linearly to zero through time;
v_i^t	$v_i^t = [v_{i1}, v_{i2}, ..., v_{iD}]$ and is the velocity;
x_i^t	$x_i^t = [x_{i1}, x_{i2}, ..., x_{iD}]$ and is the position of particle i in current time t respectively;
p_i^t	is its personal best position in current time t;
g^t	is global best position ever of entire swarm up to time t;
c_1, c_2	are acceleration coefficients that pull particles faster to personal best and global best respectively;
r_1, r_2	are random number which is uniformly distributed in $[0, 1]$;

2.4.2.3 Sea Lion Optimization Algorithm (SLnO)

Sea lions are considered as one of the most intelligent animals in wildlife which live on both lands and the oceans. They usually live in a large swarm with thousands of members, and this large swarm may contains many subgroups with their own hierarchy as well. In each subgroup, there is a dominant sea lion playing a role as the leader of the subgroup. All activities of subgroups are decided following the leader ship of that sea lion.

The intelligence of sea lions can be seen through the way they organize their groups and hunt the prey. Hunting as a group allow sea lions to have more opportunities of obtaining more food especially when the amount of fish is quite large. Usually, sea lions capture their prey together by circling the prey in a narrow ball, and the size of this "ball" continues to be decrease until the prey is totally wiped out. The main phases of hunting behaviors of sea lions can be illustrated as 3 steps as follows:

- Tracking and chasing the prey using their senses.
- Calling other members to gather and implement encircling strategy around the prey.
- Attack towards the prey which is captured in the circle.

Those behaviors is the inspiration for the Sea Lion Optimization (SLnO) which was first introduced in [16]. The algorithm mimics the amazing social behaviors and interesting hunting activities of sea lions. The formulas of the phases *Detecting and tracking phase*, *Vocalization phase* and *Attacking phase* illustrate perfectly encircling mechanisms which is utilized by sea lions. We summarize and discuss briefly each phase in the algorithm as below, meanwhile the pseudo-code of SLnO is provided in details in **Algorithm 2**.

1. **Detecting and tracking phase**

Algorithm 2: Sea Lion Optimization (SLnO)

```
1  Initialize the Sea Lion population $X_i (i = 1, 2, .., n)$ randomly.
2  Calculate fitness of each solution (sea lion).
3  $X_* \leftarrow$ the best solution
4  for $Iter = 0 \rightarrow Iter_{max}$ do
5      Calculate the value of $C$
6      for $SeaLion$ $in$ $population$ do
7          Calculate $SP_{leader}$ using Eq. 2.19
8          if $SP_{leader} < 0.25$ then
9              if $|C| < 1$ then
10                 Update the location of the current search agent using Eq. 2.17
11             else
12                 Choose a random search agent $SL_{rand}$
13                 Update the locatiion of current search agent by Eq. 2.24
14             end
15         else
16             Update the location of the current search agent by Eq. 2.22
17         end
18         Evaluate population: fix if any solutions go beyond the boundary
19         Recompute the fitness of all solutions
20         Check and update $X_*$ if a better solution is found.
21     end
22 end
23 Results: $X_*, f(X_*)$
```

Sea lions can identify the location of the prey and gather other members that will join the subgroup to organize the net following the encircling mechanism. This sea lion plays an important role as a leader for this hunting behavior and other members' position will be updated following the position of the prey. In SLnO algorithm, the prey is considered as the current best solution or the solution closest to the optimal solution. This behaviors is presented mathematically using Eq. (2.17) and Eq. (2.18) as follows:

$$Dist = |2B.P(t) - SL(t)| \tag{2.17}$$

$$SL(t+1) = P(t) - Dist.C \tag{2.18}$$

Where $Dist$ indicates the distance between the prey and the current sea lion; $P(t)$ and $SL(t)$ represent the position vectors of best solution and the sea lion in iteration t respectively; B is random vector in the range $[0, 1]$ which is multiplied by 2 to increase the search space, helping the search agent find optimal or near optimal position. $SL(t+1)$ is the new position of search agent after updating and C is linearly decreased from 2 to 0 over the course of iterations, indicating the encircling mechanism of sea lion group when they move towards the prey and surround them.

2. **Vocalization phase**

   When a sea lion recognize a group of their prey (such as fish), it will call other sea lions in their group for gathering and creating a net to capture the prey. That sea lion is considered as the leader and it will lead the group of sea lions moving towards and decide the behaviors of the group. These behaviors are modeled mathematically as shown in Eq. (2.19), (2.20) and (2.21):

$$SP_{leader} = |(V_1(1+V_2)/V_2| \tag{2.19}$$

$$V_1 = \sin(\theta) \tag{2.20}$$

$$V_2 = \sin(\phi) \tag{2.21}$$

   Where SP_{leader} is the value that illustrates the decision of the leader followed by other sea lions in the group; θ and ϕ are the angles of its voice's reflection and refraction in the water, respectively.

3. **Attacking phase (Exploitation phase)**

   The hunting activities of sea lions are led by the leader. In SLnO algorithm, the target prey is considered the current best candidate solution. In order to mathematically mimic the hunting behaviors of sea lions, two phases are introduced as follows:

   - *Dwindling encircling technique:* This behavior depends on the value of C in Eq. 2.18. C is linearly decreased from 2 to 0 over the course of iterations, so this allows the search space around the current best position to shrink and force other search agents to updated in this search space as well. Therefore, a new updated position of a sea lion can be located anywhere in the search space between its current position and the location of the present best agent.

- *Circling updating position*: Sea lions chase bait ball of fishes and hunt them starting from edges. Eq. 2.22 is proposed in this regard:

$$SL(t+1) = |P(t) - SL(t)|.\cos(2\pi m) + P(t) \tag{2.22}$$

Where $|P(t)-SL(t)|$ illustrates the distance between the best optimal solution (the prey) and the current search agent in t-th iteration, $||$ means the absolute value and m is a random number in the range $[-1, 1]$.

4. **Searching for prey (Exploration phase)** In exploration phase, the search agents update their positions based on a randomly selected sea lion. The condition that allows exploitation phase to happen is when the value of C becomes greater than 1, and the process of finding a new agent is presented by Eq. (2.23) and (2.24) as below:

$$Dist = |2B.SL_{rnd}(t) - SL(t)| \tag{2.23}$$

$$SL(t+1) = SL_{rnd}(t) - Dist.C \tag{2.24}$$

Where $SL_{rnd}(t)$ is a random sea lion that is selected randomly from current population.

Chapter 3

Improved Sea Lion Optimization (ISLO) algorithm and Proposed Model for Auto-Scaling (ISLO-CFNN)

3.1 Improved Sea Lion Optimization (ISLO)

Sea Lion Optimization (SLnO) is one of the newest swarm-based optimization algorithm. In experiments in SLnO origin paper [16], SLnO is proved that it outperformed several well-known bio-inspired model such as Generic Algorithm (GA), Particle Swarm Optimization (PSO) and Whale Optimization Algorithm (WOA). However, like many other algorithms, SLnO also faces the problem of local minimum, slow convergence and diversity degradation of population. In SLnO's exploitation phase, the updating operation 2.18 only takes the distance between the current agent and the best solution into account, which makes updated position always oriented to one direction, leading to poverty of exploitation ability. Also, in exploration phase, although the participation of two agents that already exist in population in the updating operation 2.24 helps the new-born solution inherits

Algorithm 3: Improved Sea Lion Optimization (ISLO)

```
1  Initialize the Sea Lion population $X_i (i = 1, 2, .., n)$ randomly.
2  Calculate fitness of each solution (sea lion).
3  $X_* \leftarrow$ the best solution
4  for $Iter = 0 \rightarrow Iter_{max}$ do
5      Calculate the value of $C$
6      for $SeaLion$ $in$ $population$ do
7          Calculate $SP_{leader}$ using Eq. 2.19
8          if $SP_{leader} < 0.25$ then
9              if $|C| < 1$ then
10                 Calculate $Dist_1$ and $Dist_2$ using Eq. 3.1 and 3.2 Update the
                   location of the current search agent using Eq. 3.3
11             else
12                 Choose a random search agent $SL1_{rand}$ and $SL2_{rand}$
13                 Update the location of current search agent by Eq. 3.4
14             end
15         else
16             Update the location of the current search agent by Eq. 2.22
17         end
18         Evaluate population: fix if any solutions go beyond the boundary
19         Recompute the fitness of all solutions
20         Check and update $X_*$ if a better solution is found.
21     end
22 end
23 Results: $X_*, f(X_*)$
```

current decent features of population, new solution has no way to reach another position outside of existing positions. This results in a dramatic decrease in the diversity of population, and significantly influences the ability of escaping local minimum of SLnO algorithm. All things considered, we decided to enhance those 2 operators by taking individual information into account for exploitation phase, and using a technique called opposition-based operation for exploration phase. These two improvements form a new version of SLnO called Improved Sea Lion Optimization (ISLO) algorithm. The pseudo code of the algorithm is presented in Algorithm 3, and our improvements would be discussed in detail in 3.1.1 and 3.1.2 as below.

3.1.1 Exploitation phase improvement

As mentioned above, SLnO have its own drawbacks in its exploitation phase. In order to enhance the performance of the operation 2.18, not only distance between the current agent and the best agent, but also the influences of an individual experiment in history is considered in new improved operation. This idea stems from the updating mechanism of PSO [7] where the velocity of a particular particle is influenced by both the best individual and best personal information. ***************.

Following that idea, we apply the information sent from best individual experiment in the same way as best agent position. The formulas of new updating mechanism for exploitation phase is as follows:

$$Dist_1 = |2B.P(t) - SL(t)| \tag{3.1}$$

$$Dist_2 = |2B.P_i(t) - SL(t)| \tag{3.2}$$

$$SL(t+1) = r_1(P(t) - Dist_1.C) + r_2(P_i(t) - Dist_2.C) \tag{3.3}$$

where $P_i(t)$ is the personal best position of agent i up to time t, and $r1, r2$ are random numbers in range $[\,0, 1\,]$.

In new operation 3.3, the new-updated position of an individual is the result of adding two vectors, one is the vector presenting the direction of that individual towards the best agent, and another is the direction towards its own experiences in history. The influences of both two factors are determined by two random numbers r_1 and r_2. r_1 and r_2 play an extremely important role in the updating mechanism, because they create random characteristics for the operation, helping ISLO avoiding local minimum and taking advantages of the two factors. Without the appearance of r_1 and r_2, the updated position is always affected exactly half by best agent and half by its experience, which may lead to degradation of the diversity of population.

3.1.2 Exploration phase improvement

In origin SLnO algorithm, new-born agents that are created in exploration phase cause a poor exploration search ability because of inheriting features of existing solutions. In order to tackle this problem, exploration phase is required the operation to have the ability of creating a decent new-born solution. New updated position need to satisfy two characteristics: carrying random features for ensuring a strong capability of exploration phase, and landing in a position decent enough (close enough to the best agent position) for updating positions in the next generation. From that motivation, a method called Opposition-based Learning (OBL) [21], which is successfully applied for enhancing bio-inspired optimization algorithms such as GA [21], PSO [22] [20] in solving several optimization problems including finding parameter for deep learning models [18] [17] is utilized as a base model for our improvement in exploration operation of SLnO.

The idea of OBL is applied in the operation in the way of finding a new random optimal solution, but still retain a part of features from existing solutions. This is done by calculating the opposed position to the current position of an living agent in population. For ensuring random characteristics of new found position, a random agent from population, and a random agent in the search space will be chosen for participating in this operation. The mathematical formulas are given as follows:

1. Select a random solution $SL1_{rand}$ in the search space.
2. Select a random existing solution $SL2_{rand}$ in the population.
3. Create a new solution SL_{rand} by calculating the opposed position to $SL1_{rand}$ through $SL2_{rand}$.

$$SL_{rand} = 2 * SL2_{rand} - SL1_{rand} \tag{3.4}$$

3.2 Proposed model for auto-scaling problem in Cloud Computing

In chapter 2, we generally discussed about Cloud Computing and the problem of auto-scaling with the existing methods for solving this. In general, it is relatively necessary to build cloud computing servers with the capacity of automatically expanding and shrinking the resources allocated. Although there are a number of solutions proposed for tackling this issue, they all have their own drawbacks.

The FFNN model, which is widely used for solving many real-world issues, is too simple to capture the characteristics of time-series data because after feed-forwarding through hidden layers, the origin information of the input neural could be forgotten. On the other hand, RNN-based models such as LSTM or GRU have to face the problem of extremely complex structures that potentially lead to over-fitting, or the huge number of hyper-parameters which are needed to tuned.

The CFNN can take the advantages of its structure and diminish the problem raising when the model structures are too simple (FFNN) or too complex (LSTM, GRU) because of the connection added between the input layer and output layer. However, the gradient descent (GD) algorithm which is used for optimizing CFNN still have its own drawback of being stuck in local minimum and slow convergence speed.

All thing considered, we proposed a new model ISLO-CFNN to improve the weak points of origin CFNN model by replacing GD algorithm by above proposed ISLO algorithm in optimizing the parameters of the network while training process. Also, in order to evaluate the performance of our proposed model, we would build both our model and existing models for the purposes of evaluation and comparison.

Fig. 3.1 illustrates the skeleton of forecasting system designed. There are four main phases, each of them is indispensable in our model. The phases are Collecting data, Data pre-processing, Building and Training model and Deploy prediction model. Firstly, historical records about resources used are collected and saved in lines in a log file. These information is extracted and pre-processed before being used for training the prediction model that is designed. In the final stage, the trained model is applied for data in current time, and it will predict the amount of resources needed. Specifically, in Building and Training model stage, CFNN model is built with fixed nodes in all layers, and it will be optimized by ISLO algorithm, which is discussed in detail in Section 3.1. We will discuss about each phase as below.

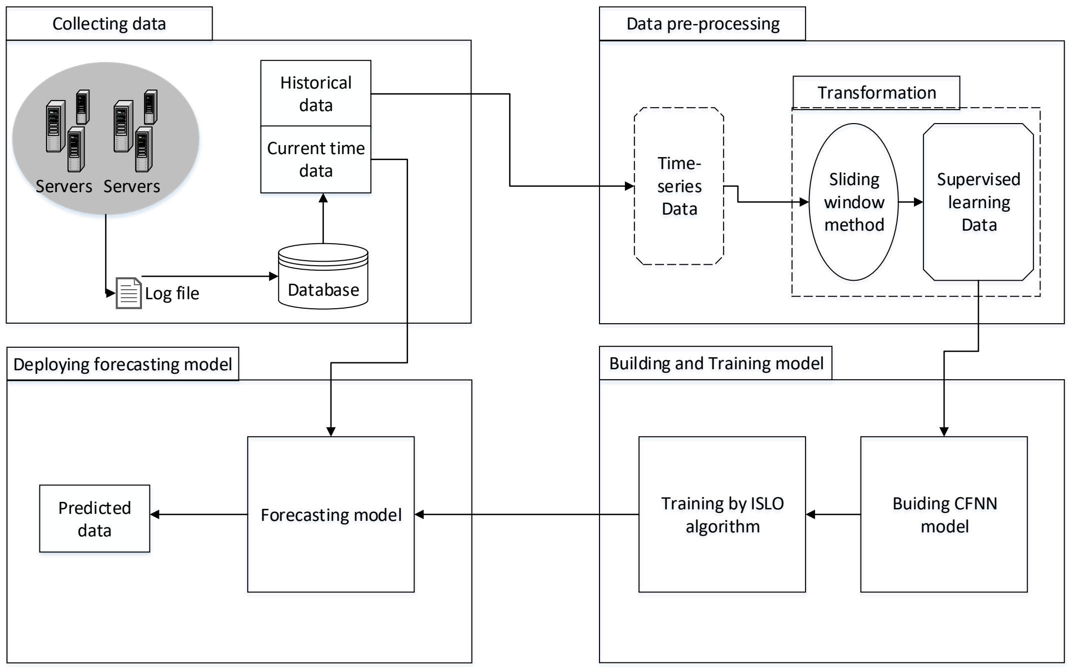

Figure 3.1: Proposed model design

3.2.1 Collecting data

Before building any forecasting systems, the very first and the most important task that must be done is collecting data. Therefore, we collect data recording the resource usage of Google (Google Cluster Trace data) and Internet Traffic data collected from a private internet service provider (ISP) in Europe and the United Kingdom (these datasets will be discuss in detail in Chapter 4). The data collected contains two main parts: the data from history that we use for training, and current time data that we use for predicting resource usage in the future.

Table 3.1: Time-series data and Supervised learning data comparison

Time-series data	Supervised Learning data
1. Time 1, value 1	1. Input 1, output 1
2. Time 2, value 2	2. Input 2, output 2
3. Time 3, value 3	3. Input 3, output 3

3.2.2 Data pre-processing

This phase play a role as a pre-processor transforming the raw data into the kind of data that can be used in neural networks. In order to learn, models need both training data and testing data. We create the data for models through several steps as follows:

- Evaluate and choose carefully which columns of data are needed for the forecasting model.
- The parts of data chosen are then normalized in the range $[0, 1]$.
- Transform time-series data into supervised learning data using *Sliding window* technique.
- Divide processed data into two sets: training set and testing set.

The step $3th$ of the pre-processing phase is necessary because time-series data is the data recorded through time, and there is no term of features and output data. Therefore, we need to transform this data into supervised learning data, that contains input features, and output. Table 3.1 depicts the difference between time-series data and normal data used in supervised learning.

In order to create data for supervised learning, we use the method called *Sliding window*. This method takes the data of k values before the time t as the features and output data is the value at the time t. For example, when $k = 3$, the results of data transformation is shown as in Table 3.2.

Table 3.2: Example of data transformation using Sliding window method

Time-series data	Transformed data			
	Input			Output
Time ($t = 4$), Value 4	Value 1	Value 2	Value 3	Value 4
Time ($t = 5$), Value 5	Value 2	Value 3	Value 4	Value 5
Time ($t = 6$), Value 6	Value 3	Value 4	Value 5	Value 6
Time ($t = 7$), Value 7	Value 4	Value 5	Value 6	Value 7
...	...	...	...	...

3.2.3 Building and Training model

In this phase, pre-processed data is used for training our proposed model called ISLO-CFNN, which is CFNN being trained by the optimization of ISLO algorithm. ISLO algorithm is applied to train a CFNN model with one hidden layer. There is two key aspects needed to be taken into consideration: firstly, the formation of an agent in ISLO and the selection of fitness function.

Firstly, each agent in the population in ISLO are presented as one solution for CFNN model, which means that a search agent is a one-dimensional vector created by concatenating weights and biases of CFNN. Therefore, the features of a search agent contains three elements: a set of weights connecting the input layer with hidden layer, a set of weights connecting the hidden layer with output layer and also and a set of weights connecting the input layer with output layer. Therefore, the length of a solution can be calculated by Eq. 3.5.

$$Solution\_length = (1 + i) * h + (1 + h) * o + (1 + i) * o \tag{3.5}$$

Where i, h, o is the number of input, hidden and output neurons, respectively (in time-series prediction, the number of output neurons is one).

Secondly, the fitness value of each agent in ISLO is considered as the loss value of the CFNN model with the parameter set from the agent and input data. We utilize the loss function Mean Square Error (MSE) to calculate the difference between the actual and predicted output values by generated agent for all samples in the training set.

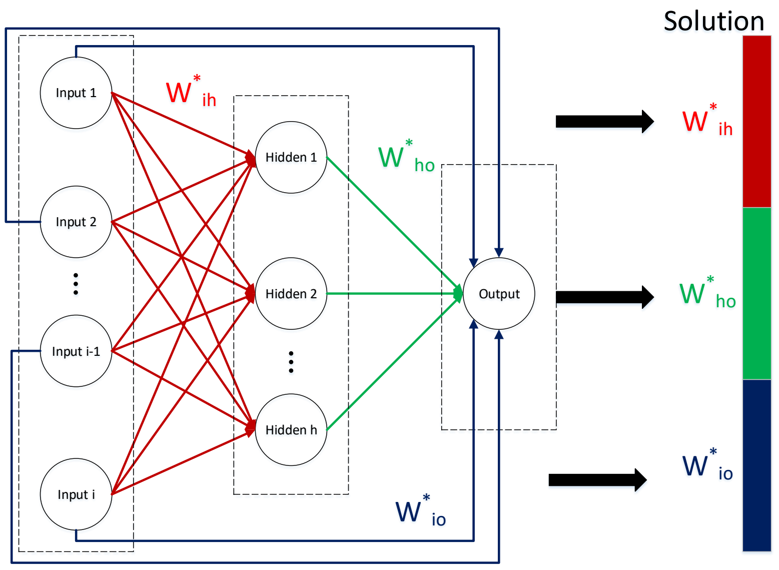

Figure 3.2: Encoding process transforming a parameter set in CFNN into an agent in ISLO algorithm. (W^* indicates weights and biases between 2 layers)

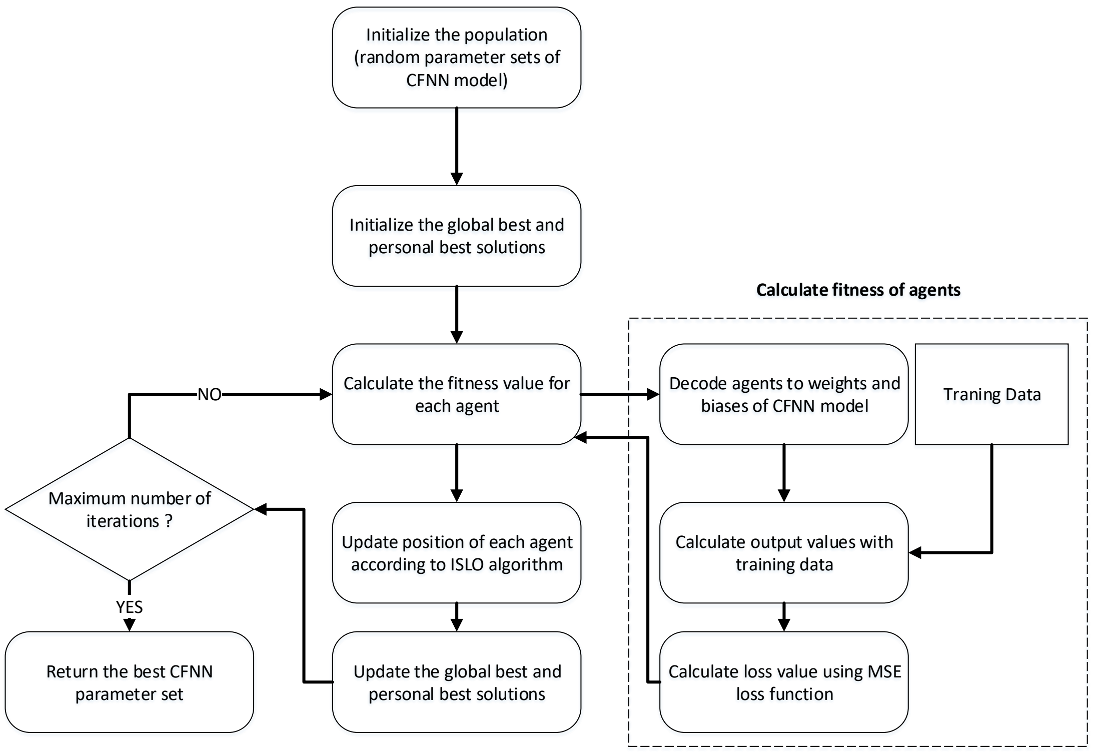
Initialize the population (random parameter sets of CFNN model)
Initialize the global best and personal best solutions
Calculate the fitness value for each agent
Update position of each agent according to ISLO algorithm
Update the global best and personal best solutions
Maximum number of iterations ?
NO
YES
Return the best CFNN parameter set
Calculate fitness of agents
Decode agents to weights and biases of CFNN model
Traning Data
Calculate output values with training data
Calculate loss value using MSE loss function

Figure 3.3: The work flow of ISLO-CFNN model.

The workflow of ISLO applied in this work for training CFNN are depicted in Fig. 3.3, and can be generally presented by the following steps:

1. Initialization: pre-define the number of search agents in ISLO, which are randomly generated in the range [-1, 1]. Each parameter set of CFNN model is

encoded to a vector playing a role as an agent in ISLO population. (See in Fig. 3.2)

2. Calculate fitness value for each search agent: The quality of a solution is measured by the loss value of CFNN output. After being decoded into weights and biases vectors, a solution will be applied to form a CFNN model. Data samples in training set is then feed-forwarded through the network, generating predicted output values. Finally, the fitness value is calculated as the difference between predicted and actual output values through the MSE loss function.

3. Update positions of all search agents following formulas of ISLO algorithm.

4. Steps 2 and 3 are repeated until the difference is close enough, or the maximum number of iterations is reached.

5. Return the best CFNN parameter set.

3.2.4 Deploy prediction model

After obtaining CFNN model with the best parameter set by ISLO algorithm, the model is installed on servers and ready to predict the demand for resources in the future based on historical data.

Chapter 4

Experiments

Chapter 5

Conclusions

Bibliography

[1] Maryam Amiri and Leyli Mohammad-Khanli. Survey on prediction models of applications for resources provisioning in cloud. *Journal of Network and Computer Applications*, 82:93–113, 2017.

[2] E Michael Azoff. *Neural network time series forecasting of financial markets.* John Wiley & Sons, Inc., 1994.

[3] R Boné. *Recurrent neural networks for time series forecasting.* PhD thesis, PhD thesis, Université de Tours, Tours, FRANCE, 2000.

[4] Rohitash Chandra and Mengjie Zhang. Cooperative coevolution of elman recurrent neural networks for chaotic time series prediction. *Neurocomputing*, 86:116–123, 2012.

[5] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[6] Jerome T Connor, R Douglas Martin, and Les E Atlas. Recurrent neural networks and robust time series prediction. *IEEE transactions on neural networks*, 5(2):240–254, 1994.

[7] Russell Eberhart and James Kennedy. Particle swarm optimization. In *Proceedings of the IEEE international conference on neural networks*, volume 4, pages 1942–1948. Citeseer, 1995.

[8] Rui Fu, Zuo Zhang, and Li Li. Using lstm and gru neural network methods for traffic flow prediction. In *2016 31st Youth Academic Annual Conference of Chinese Association of Automation (YAC)*, pages 324–328. IEEE, 2016.

[9] Felix A Gers, Douglas Eck, and Jürgen Schmidhuber. Applying lstm to time series predictable through time-window approaches. In *Neural Nets WIRN Vietri-01*, pages 193–200. Springer, 2002.

[10] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[11] Tian Guo, Zhao Xu, Xin Yao, Haifeng Chen, Karl Aberer, and Koichi Funaya. Robust online time series prediction with recurrent neural networks. In *2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, pages 816–825. Ieee, 2016.

[12] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[13] James Kennedy. Particle swarm optimization. *Encyclopedia of machine learning*, pages 760–766, 2010.

[14] Timo Koskela, Mikko Lehtokangas, Jukka Saarinen, and Kimmo Kaski. Time series prediction with multilayer perceptron, fir and elman neural networks. In *Proceedings of the World Congress on Neural Networks*, pages 491–496. Citeseer, 1996.

[15] Martin Längkvist, Lars Karlsson, and Amy Loutfi. A review of unsupervised feature learning and deep learning for time-series modeling. *Pattern Recognition Letters*, 42:11–24, 2014.

[16] Raja Masadeh, Basel A Mahafzah, and Ahmad Sharieh. Sea lion optimization algorithm. *Sea*, 10(5), 2019.

[17] Thieu Nguyen, Tu Nguyen, Binh Minh Nguyen, and Giang Nguyen. Efficient time-series forecasting using neural network and opposition-based coral reefs

optimization. *International Journal of Computational Intelligence Systems*, 12(2):1144–1161, 2019.

[18] Muhammad Rashid and Abdul Rauf Baig. Improved opposition-based pso for feedforward neural network training. In *2010 International Conference on Information Science and Applications*, pages 1–6. IEEE, 2010.

[19] David E Rumelhart, Geoffrey E Hinton, Ronald J Williams, et al. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.

[20] Jun Tang and Xiaojuan Zhao. An enhanced opposition-based particle swarm optimization. In *2009 WRI Global Congress on Intelligent Systems*, volume 1, pages 149–153. IEEE, 2009.

[21] Hamid R Tizhoosh. Opposition-based learning: a new scheme for machine intelligence. In *International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce (CIMCA-IAWTIC'06)*, volume 1, pages 695–701. IEEE, 2005.

[22] Hui Wang, Hui Li, Yong Liu, Changhe Li, and Sanyou Zeng. Opposition-based particle swarm algorithm with cauchy mutation. In *2007 IEEE Congress on Evolutionary Computation*, pages 4750–4756. IEEE, 2007.

[23] Budi Warsito, Rukun Santoso, Hasbi Yasin, et al. Cascade forward neural network for time series prediction. In *Journal of Physics: Conference Series*, volume 1025, page 012097. IOP Publishing, 2018.

[24] Jia-Shu Zhang and Xian-Ci Xiao. Predicting chaotic time series using recurrent neural network. *Chinese Physics Letters*, 17(2):88, 2000.