TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG
*****

ĐỒ ÁN TỐT NGHIỆP

Đề tài:

Mô hình dự đoán chuỗi thời gian mờ đa chiều sử dụng LSTM và kĩ thuật phân tích tương quan giữa các chiều dữ liệu áp dụng cho tài nguyên đám mây

Sinh viên thực hiện: **Nguyễn Đức Thắng**

Giảng viên hướng dẫn: **TS. Nguyễn Bình Minh**

Hà Nội, 12/2018

Mục lục

Danh sách bảng

Danh sách hình vẽ

Chương 1

Introduction

1.1 Problems

1.2 Summary

Chương 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. [9]. 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. ***, 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.

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 (see in Fig. ***). 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], [12]. Fig. *** shows a MLPs with a n-neuron input layer and one output layer. The mathematical equation of the architecture in Fig. *** 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 [16]. The architecture of CFNN is illustrated in Fig. ***, and the mathematical formulas for CFNN model are presented as follows:

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

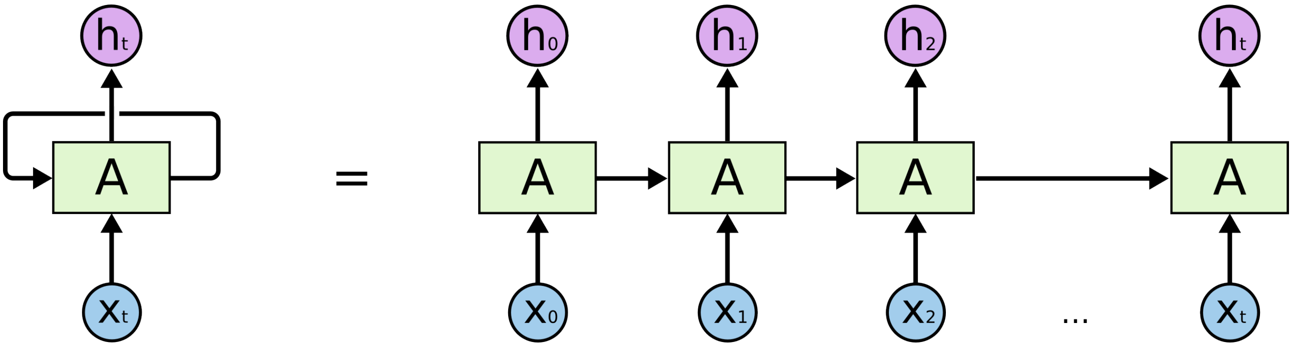
$$C = f_c(W_c^T X + b_c) \tag{2.5}$$

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

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 representing 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 [17], [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.



Hình 2.1: 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) [11] 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 [8], [10], [7]. The architecture of LSTM units is illustrated in Fig. 2.2, 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.

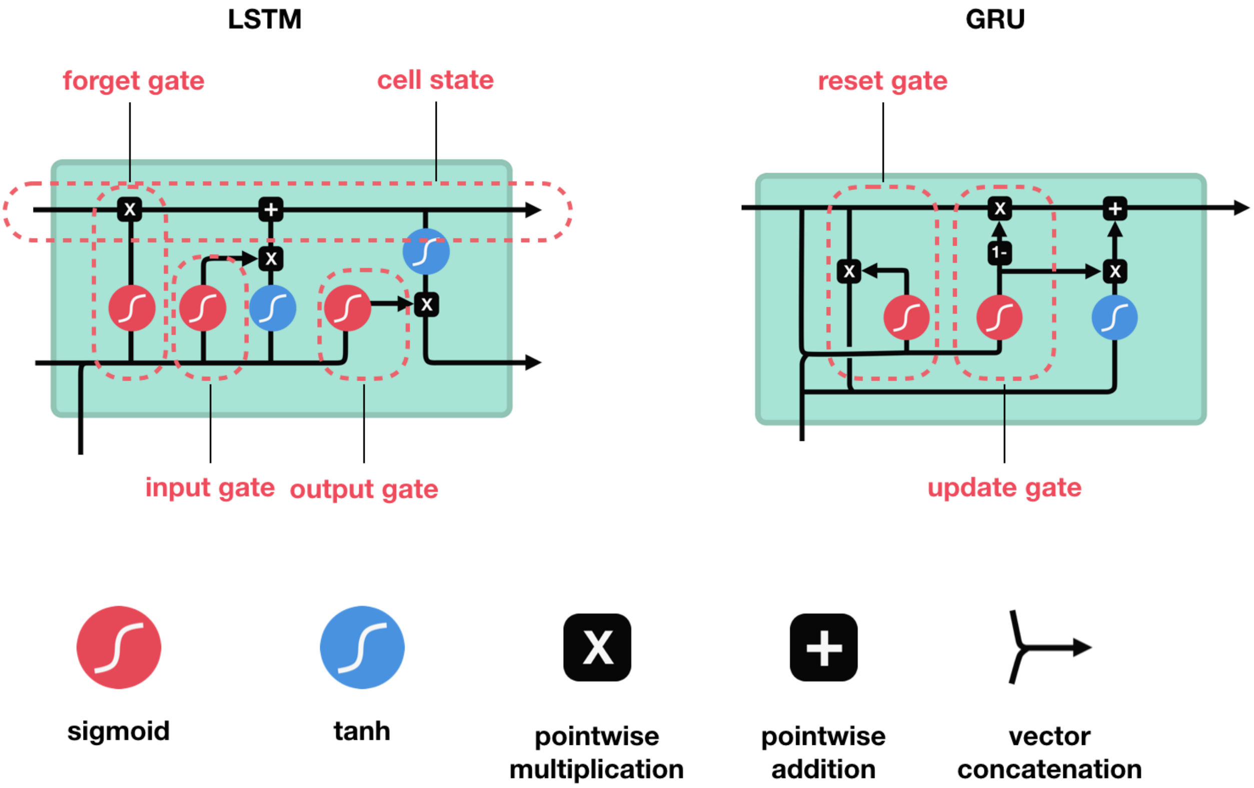
$$i_t = sigmoid(W_i x_t + U_i h_{t-1} + b_i) \tag{2.7}$$

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

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

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

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



Hình 2.2: LSTM and GRU architechture. Source: https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21

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 chosen solution for time series forecasting such as in [13] and [3] The srtucture of GRU units is presented in Fig. 2.2 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.

$$z_t = sigmoid(W_z x_t + U_z h_{t-1} + b_z) \tag{2.12}$$

$$r_t = sigmoid(W_r x_t + U_r h_{t-1} + b_r) \tag{2.13}$$

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

2.4 Fundamental knowledge

2.4.1 Artificial Neural Network (ANN)

2.4.1.1 Activation functions

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).

2.4.1.1.1 Sigmoid function

: takes a number as input and returns a value in $[\,0, 1]\,$. (Fig. ****)

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

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 $[\,-1, 1]\,$. (Fig ***)

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

2.4.1.1.3 Rectified Linear Unit (ReLU) function:

is a function having threshold at 0 value (Fig. ***). 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. (Fig. ****)

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

2.4.1.1.5 Exponential Linear Unit (ELU) function:

This function was first proposed several years ago. In many experiments, it is proved that it can lead to a faster convergence and better result of deep learning models.

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

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)**:

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

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

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

- **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.

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 [15]. 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 previous 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)

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

Algorithm 2: Sea Lion Optimization (SLnO)

```
Initialize the Sea Lion population $X_i (i = 1, 2, .., n)$ randomly.
Calculate fitness of each solution (sea lion).
$X_* \leftarrow$ the best solution
for $Iter = 0 \rightarrow Iter_{max}$ do
    Calculate the value of $C$
    for $SeaLion\ in\ population$ do
        Calculate $SP_{leader}$ using Eq. 2.17
        if $SP_{leader} < 0.25$ then
            if $|C| < 1$ then
                Update the location of the current search agent using Eq. 2.15
            else
                Choose a random search agent $SL_{rand}$
                Update the locatiion of current search agent by Eq. 2.22
            end
        else
            Update the location of the current search agent by Eq. 2.20
        end
        Evaluate population: fix if any solutions go beyond the boundary
        Recompute the fitness of all solutions
        Check and update $X_*$ if a better solution is found.
    end
end
Results: $X_*, f(X_*)$
```

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 [14]. 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**

   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. (2.15) and Eq. (2.16) as follows:

   $$Dist = |2B.P(t) - SL(t)| \tag{2.15}$$

   $$SL(t+1) = P(t) - Dist.C \tag{2.16}$$

   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. (2.17), (2.18) and (2.19):

$$SP_{leader} = |(V_1(1 + V_2)/V_2| \tag{2.17}$$

$$V_1 = \sin(\theta) \tag{2.18}$$

$$V_2 = \sin(\phi) \tag{2.19}$$

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. 2.16. 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. 2.20 is proposed in this regard:

$$SL(t + 1) = |P(t) - SL(t)|. \cos(2\pi m) + P(t) \tag{2.20}$$

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. (2.21) and (2.22) as below:

$$Dist = |2B.SL_{rnd}(t) - SL(t)| \tag{2.21}$$

$$SL(t+1) = SL_{rnd}(t) - Dist.C \tag{2.22}$$

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

Chương 3

Sea Lion Optimization Improvements and Proposed Model for Auto-Scaling

Chương 4

Experiments

Chương 5

Conclusions

.1 References

Tài liệu tham khảo

[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] 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.

[8] 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.

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

[10] 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.

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

[12] 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.

[13] 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.

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

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

[16] 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.

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