HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

ENGINEERING THESIS

ISLO - An Improved Sea Lion Optimization Algorithm and Application to Multi-Layer Perceptron For Cloud Resource Consumption Prediction Problem

Tran Quang Trung
trung.tq154002@sis.hust.edu.vn

School of Information and Communication Technology
Information System

Supervisor: Dr. Nguyen Binh Minh ______________

Supervisor's signature

Department: Information System

Institution: School of Information and Communication Technology

HANOI, 12/2019

THESIS TITLE

ISLO - An Improved Sea Lion Optimization Algorithm and Application to Multi-Layer Perceptron For Cloud Resource Consumption Prediction Problem

Supervisor

Signature and Name

Acknowledgement

First and foremost, I would like to thank all teachers in Hanoi University of Science and Technology, especially who are in School of Information and Communication Technology for giving me rewarding knowledge and experience to complete this project.

I also wish to express my sincere appreciation to my supervisor, Dr. Nguyen Binh Minh, who convincingly guided and encouraged me to be professional and do the right thing even when the road got tough. Without his persistent help, the goal of this thesis would not have been realized.

Finally, I must express my very profound gratitude to my parents and to my friends for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. Thank you.

Abstract

The Sea Lion Optimization (SLnO) algorithm have been shown to be competitive in searching for an optimal solution. This thesis proposes improvements to this algorithm that are based on the idea from Particle Swarm Optimization (PSO) algorithm, and opposition-based learning (OBL), forming an improved version of SLnO called ISLO. The ISLO is further compared with other well-known nature-inspired algorithms on 30 benchmark functions. The statistical results show that the ISLO significantly outperforms others on majority of functions in terms of accuracy and robustness. Furthermore, the work introduce a new model (ISLO-CFNN) that is a combination of ISLO and Cascade Feedforward Neural Network (CFNN). The proposed model tries to tackle the time-series forecasting problem deprived from auto-scaling demand in cloud computing. ISLO-CFNN's performance is tested by 4 time-series datasets, and is compared with several widely used deep learning models. The experiments show that ISLO-CFNN is very competitive results with compared models, proving the superior optimizing ability of ISLO in various problems ranging from theoretical functions to engineering applications.

Student
Signature and Name

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

Notation	Description
ANNs	Artificial Neural Networks
ARIMA	Autoregressive Integrated Moving Average
ARMA	Autoregressive Moving Average
BP	Backpropagation
CFNN	Cascade Feedforward Neural Network
FFNN	Feedforward Neural Network
GARCH	General Autoregressive Conditional Heteroscedastic
GD	Gradient Descent
GRU	Gated Recurrent Unit
IaaS	Infrastructure as a Service
ISLO-CFNN	Cascade Feedforward Neural Network with optimizer Improved Sea Lion Optimiztion
LSTM	Long Short-Term Memory
MLPs	Multi-Layer Perceptrons
OBL	Opposition-based Learning
PaaS	Platform as a Service
PSO-CFNN	Cascade Feedforward Neural Network with optimizer Particle Swarm Optimiztion
RNN	Recurrent Neural Network
SaaS	Software as a Service
SLAs	Service Level Agreements
SLnO-CFNN	Cascade Feedforward Neural Network with optimizer Sea Lion Optimiztion

CHAPTER 1. INRODUCTION

Meta-heuristic optimiztion algorithms are becoming more and more popular in engineering application due to their advantages: they (i) have simple concepts and the ease of implementation; (ii) do not require gradient information; (iii) have the ability to avoid local minima and (iv) can be utilized in a wide range of real-world problems covering different aspects. Among those optimization models, there is a group of algorithm that are called nature-inspired meta-heuristic algorithms. They solve optimization issues by mathematically modelling biological or physical phenomena. They can be divided into three main categories: evolution-based, swarm-based and physics-based methods.

Evolution-based algorithm are derived from evolutionary laws in nature. The search process starts with randomly generated solutions which is continuously evolved over the course of generation. The most popular evolution-based algorithm is Generic Algorithm (GA) [1], which mathematically mimics the Darwinian's evolutionary laws. The second category is physics-based methods, which imitate physical principles in the universe such as Big Bang theory, gravitation of the Earth. The third group of nature-inspired meta-heuristic optimization is swarm-based algorithms (or swarm intelligence (SI)). SI refers to the collectively intelligent activities emerging from a group of individuals called population, optimizing process is based on hunting and communication behaviors of animals in nature. The very first and most famous SI algorithm is Particle Swarm Optimization (PSO), which is a fundamental algorithm, and the inspiration for many later-born swarm-based optimizers. Additionally, it is worth mentioning here that there is a group of methods derived from human's activities in daily life, which is called human-based optimization algorithms.

Among these kinds of nature-inspired meta-heuristic algorithm, SI is the most popular one because of the ease in understanding and implementing. Due to the wide range of applications of these algorithms, in recent years, many swarm-inspired optimization algorithms have been introduced, going along with their variants and applications. Sea Lion Optimization (SLnO) is among these algorithms. It was first introduced in [2] for solving global-scale optimization problems. SLnO imitates the hunting behaviors of sea lions consisting of the way they encircle and capture preys or how they use their tail as well as their whiskers. It has been proven that SLnO can provide very competitive results compared with other well-known algorithms such as Particle Swarm Optimization (PSO), Grey Wolf Optimization

(GWO) and Whale Optimization Algorithm (WOA) when working on different benchmark functions. However, like many other methods in swarm-based optimization, SLnO still faces demerits with regard to slow convergence speed, low precision and easily being stuck into local minimums due to the degradation of population diversity over the course of iteration. To improve the performance of these algorithms, there is a number of techniques proposed, and they are proven that they can work very effectively on different algorithms such as Levy-Flight trajectory-based method [3], opposition-based learning [4] [5] [6] or using an idea from other algorithms, creating a hybrid version for improvements. However, there have not been any improved versions of SLnO even though there is several problems existing in this algorithm.

Because of their prominent advantages, such algorithms have been applied to various aspects including solving travelling salesman problem [7], apllication to electric power system [8] and image processing [9] [10]. Among applications of meta-heuristic algorithms, the use of them in optimizing artificial neural networks (ANNs) are emerging and being considered as a alternative for the backpropagation (BP) mechanism, which is the most well-know algorithm training ANNs. In the last three decade, ANNs have been developed and widely applied to classification, parttern recognition regression and forecasting problems. The ANNs' efficency mostly afftected by their learning process. For Multi-Layer Perceptron (MLPs), which is the most popular category of ANNs, the main method for training is gradient-based methods. The backpropagation algorithms and its variants [11] are considered as the most well-known method for training MLPs. However, using gradient-based algorithms for optimizing MLPs still remains several drawbacks such as tendency being trapped in local minima, slow convergence and significant dependence on the initial stage of its parameters. In order to avoid complex gradient information as well as the disadvantages of gradient-based optimization, meta-heuristic optimization algorithms have been proposed as a reliable alternative for optimizing MLPs in the perspectives of parameters and network's structure. For example, as introduced in [12], GA and PSO algrithms are utilized to optimize parameters of Wavelet-MLP for wind speed prediction, or in [13] WOA is applied to simple MLPs for solving classification problems. Some other interesting applications of these optimizers on MLPs can be found in [14], [15] and [16]. However, SLnO algorithm has not been used as an optimizer replacing BP algorithm in ANNs by any studies before.

One of the most important problems that ANNs are used for solving is time-series prediction. This problem stems from time-series data, which is a series of data points recorded in order over a period of time in many sectors in real world such as weather, stocks and healthcare. Among many aspects that time-series prediction is related to, the auto-scaling issue in cloud environments is a big question that need to be tackled. This problem is given to require cloud servers an ability to be adaptive and scalable, automatically recovering and allocating resources effectively. There is a number of fundamental methods proposed trying to solve this big issue including Autoregressive Moving Average (ARMA), Autoregressive Integrated Moving Average (ARIMA) and General Autoregressive Conditional Heteroscedastic (GARCH), but today, with an incredible increase in the amount as well as complexity of data, these methods are becoming less and less competitive compared with others. In recent years, scholars have been trying to use meachine learning and deep learning neural networks (NNs) methods to tackle this time-series problem, and they have shown better performance as reported in [17] and [18]. Among these deep learning methods, RNN-based models such as traditional RNN, LSTM and GRU are widely chosen and applied to time-series forecasting in a wide range of aspects, and they have been showing very competitive performance compared with other deep learning models. However, in these models, BP algorithm is still used in the training process, so facing BP problems as mentioned above is unavoidable. Therefore, in this work we develop a model based on deep learning NNs and meta-heuristic algorithm trying to solve this auto-scaling problem.

All things considered, for the reasons mentioned above, in this paper, we propose 2 contributions: (i) improving SLnO algorithm to ISLO algorithm, and (ii) applying ISLO algorithm to optimizing a kind of ANNs called Cascade Feedforward Neural Network (CFNN) and utilizing this model (called ISLO-CFNN) trying to solve the given workload prediction demand in cloud computing environments. Firstly, an improved version of SLnO call Improved Sea Lion Optimization (ISLO) by embracing the idea from PSO, taking the best experience of each individual into account to improve SLnO's exploitation ability, and the idea of opposition-based learning for enhacing the use of exploration phase in SLnO. ISLO's performance is tested by 30 benchmark functions belonging to 4 kinds of functions: unimodal functions, multimodal functions, hybrid functions and composition functions. The results provided by ISLO are compared with other well-known meta-heuristic algorithms including GA, PSO, WOA, TWO, QSO and the original SLnO. The results show that ISLO provides superior final fitness values as well as decent convergence speed compared with the others. Secondly, we propose a model named ISLO-CFNN (CFNN being optimized by ISLO algorithm) to solve the time-series prediction problem deprived from auto-scaling demand in cloud computing environments. ISLO-CFNN performance is experimented on 4 time-series datasets representing workload of server clusters and internet traffic. Its results are compared with other deep learning models that are well-known and widely used in time-series forecasting including traditional MLPs and CFNN with one hidden layer, RNN, LSTM and GRU in the term of accuracy. Optimizing capability of ISLO would be tested again by comparing with 2 other SI algorithms PSO and SLnO in optimizing CFNN. The figures provided by ISLO-CFNN show that this model is very competitive and can even bring more potential results compared with the others.

The remainings of this thesis are organized as follows: the fundamental knowledge about swarm-based algorithms, ANNs and auto-scaling issues in cloud computing is presented in Chapter 2. Chapter 3 describes our two main improvements on original SLnO algorithm, and the proposed model for tackling the above mentioned problem. Our experiments for testing the performance of ISLO algorithm and ISLO-CFNN model are discussed in Chapter 4. Finally, the contribution and future work of this research are concluded in Chapter 5.

CHAPTER 2. MATERIALS AND BACKGROUND

2.1 Meta-heuristic Optimization

2.1.1 Idea and motivation

As mentioned above, there are four kinds of nature-inspired meta-heuristic algorithms. They are evolution-based, physics-based, swarm-based and human-based optimization algorithms.

Evolution-based methods are those that optimize following evolutionary rules in nature. The process starts with a ramdomly generated individual considered as population, being enhanced through each generation. Each generation commonly contains the following components: reproduction, fitness evaluation and selection. Specifically, in the reproduction process, from which new-born solutions are generated, often adopts generic operators such as crossover or mutation; the fitness evaluation process obtains the quality of each solution in current population by assigning their fitness values; and selection process is willing to determine candidates with superior values among the population to survive in the next generation. The strength point of this kind of algorithms is that the best individuals are always chosen to generate potential candidate for the subsequent generation. This move the population towards and come closer to the optimal value. The most popular evolution-based algorithm is Generic Algorithm (GA) [1], which mathematically mimicks the Darwinian's evolutionary laws. Other later algorithms are Generic Programming (GP) [19], Differential Evolution (DE) [20], Biogeography-based optimization (BBO) [21] and Coral Reefs Optimization Algorithm (CRO) [22].

Physics-based methods try to mimic physical phenomena, and find a way to reach the optimal values following these principles. There are several popular algorithms in this category including Big-Bang Big-Crunch(BBBC) [35], Gravitational Search Algorithm (GSA) [36], Charged System Search (CSS) [37], Central Force Optimization (CFO) [38], Artificial Chemical Reaction Optimization Algorithm (ACROA) [39], Black Hole (BH) algorithm [40], Ray Optimization (RO) algorithm [41], Small-World Optimization Algorithm (SWOA) [42].

Swarm Intelligence (SI) or swarm-based optimization is a group of algorithms being inspired by living and foraging behaviors of animals in the nature. Unlike evolution-based methods, SI methods are based on artificial search agents' movement in a pre-defined search space. Such algorithms take advantages of exploration and exploitation phases and lead the population closer and closer to the

Table 2.1: Swarm-based optimization algorithms developed in literature.

Algorithms	Inspiration	Year of proposal
Particle Swarm Optimization (PSO) [23]	bird blocks and fish schools	1995
Ant Colony Optimization (ACO) [24]	Ant colony	1999
Honey Bees Optimization Algorithm (HBO) [25]	Marriage in honey bee	2001
Bacterial Foraging Optimization (BOA) [26]	Bacterial foraging behaviors	2002
Artificial Bee Colony (ABC[27]	Bee colony	2005
Firefly algorithm (FA) [28]	Firefly swarm	2009
Bat Algorithm (BA) [29]	Bat swarm	2012
Dolphin Echolocation Algorithm (DEA) [30]	echolocation behavior of Dolphins	2013
Grey Wolf Optimizer (GWO) [31]	Wolf clan hierachy	2014
Whale Optimization Algorithm (WOA) [32]	Humppack whale hunting behaviors	2016
Lion Optimization Algorithm (LOA) [33]	Lion hunting activities	2016
Seagull optimization algorithm (SOA) [34]	Seagull swarm	2019
Sea Lion Optimization (SLnO) [2]	Sea lion hunting	2019

optimal result over the course of iteration. For example, one of the most famous and widely used algorithm in SI group is Particle Swarm Optimization (PSO) [23], which uses the information both from the best agent and all the agents' best experience to search for the optima of a fitness function. Another popular swarm-based algorithm is Ant Conoly Optimization (ACO) [24], which is inspired by social foraging process of ants. This algorithm uses the idea of the social intelligence of ants in finding the closest path from the nest and a source of food. A pheromone matrix is enhanced over the course of iteration by the candidate solutions. Several other SI algorithms have been regularly proposed, some of them are listed in Table. 2.1. In general, this group of methods started to become more attractive since PSO is proven to be very competitive with evolution-based and physics-based algorithms. In fact, swarm-based methods have some advantages over the others. For example, swarm-based algorithms find new better by preserving the information from previous iterations, while evolution-based methods such as GA discard any information immediately when a new generation is formed. Also, they are usually formed of less updating operators compared to the others (crossover, mutation, selection *etc.*) and therefore it is easier to implement.

Human-based algorithms are recently developed by observing and applying human behaviors in daily life. Some of the well-known algorithms are Harmony Search (HS) [43], Teaching Learning Based Optimization (TLBO) [44], League Championship Algorithm (LCA) [45], Tabu Search (TS) [46] and Colliding Bodies Optimization (CBO) [47].

2.1.2 Particle Swarm Optimization (PSO)

PSO [48] 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 2.1.

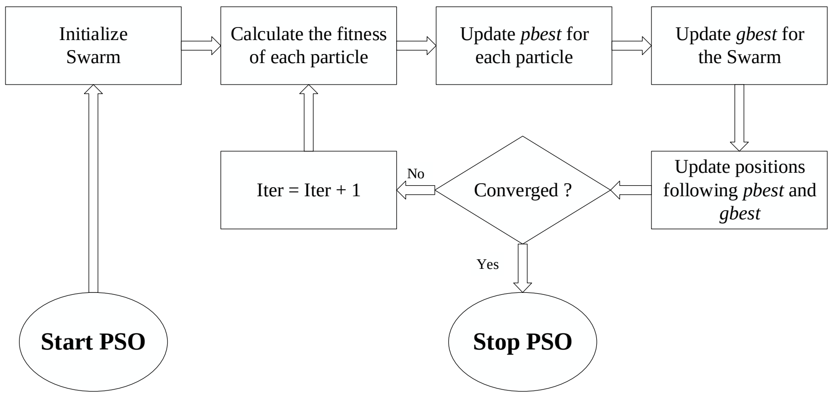

Initialize Swarm
Calculate the fitness of each particle
Update *pbest* for each particle
Update *gbest* for the Swarm
Iter = Iter + 1
No
Converged ?
Update positions following *pbest* and *gbest*
Yes
Start PSO
Stop PSO

Figure 2.1: 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:

$$v_i^{t+1} = \omega.v_i^t + c_1.r_1(p_i^t - x_i^t) + c_2.r_2(g^t - x_i^t) \tag{2.1}$$

$$x_i^{t+1} = x_i^t + v_i^t \tag{2.2}$$

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 $[0, 1]$;

2.1.3 Sea Lion Optimization Algorithm (SLnO)

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 [2]. 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 1**.

1. **Detecting and tracking phase**

Algorithm 1: 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. 2.5
        if $SP_{leader} < 0.25$ then
            if $|C| < 1$ then
                Update the location of the current search agent using Eq. 2.3
            else
                Choose a random search agent $SL_{rand}$
                Update the locatiion of current search agent by Eq. 2.10
            end
        else
            Update the location of the current search agent by Eq. 2.8
        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: $X_*, f(X_*)$
```

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. 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. (2.3) and Eq. (2.4) as follows:

$$Dist = |2B.P(t) - SL(t)| \tag{2.3}$$

$$SL(t+1) = P(t) - Dist.C \tag{2.4}$$

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. $SL(t+1)$ is the new position of search agent after updating and C is linearly decreased from 2 to 0 over the course of iteration, 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. (2.5), (2.6) and (2.7):

   $$SP_{leader} = |(V_1(1 + V_2)/V_2| \tag{2.5}$$

   $$V_1 = \sin(\theta) \tag{2.6}$$

   $$V_2 = \sin(\phi) \tag{2.7}$$

   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. 2.4. C is linearly decreased from 2 to 0 over the course of iteration, 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. 2.8 is proposed in this regard:

   $$SL(t + 1) = |P(t) - SL(t)|.\cos(2\pi m) + P(t) \tag{2.8}$$

   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. (2.9) and (2.10) as below:

$$Dist = |2B.SL_{rnd}(t) - SL(t)| \tag{2.9}$$

$$SL(t+1) = SL_{rnd}(t) - Dist.C \tag{2.10}$$

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

Nevertheless, the results provided in [2] show that SLnO faces with the problem of being trapped in local optima and slow convergence due to the poverty of exploitation and exploration. Firstly, in exploitation phase, the updating mechanism in equation 2.4 only take the distance between current agent and the best one into account. This always lead the updated position towards one direction to the best agent if no better solution is found, and limit the exploitation ability of the algorithm. Secondly, the random agent chosen in exploration phase is still in current population. This makes new updated position still have the characteristics influenced by existed solutions, which can result in being trapped in local minima. Therefore, in this work, we focus on improving both exploitation and exploration ability of SLnO by proposing two improvements in two operators.

2.2 Artificial Neural Network (ANN)

2.2.1 Activation functions

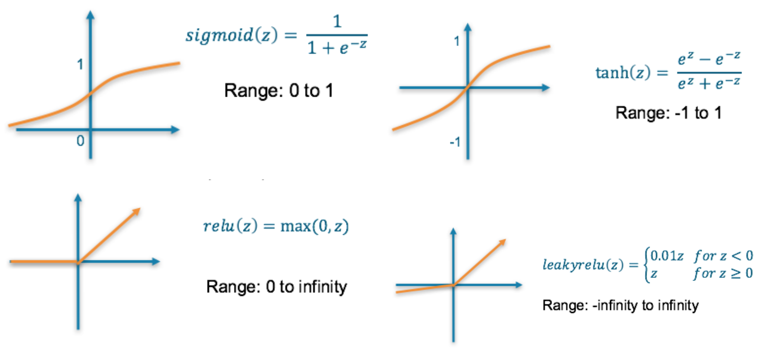


Figure 2.2: Activation functions. Source: [49].

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. 2.2).

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

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

- **Hyperbolic Tangent (Tanh) function**: is also like Sigmoid function but better, the range of the output from Tanh funtion is in $[-1, 1]$.

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

- **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.

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

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

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

2.2.2 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.

2.2.3 Backpropagation - the ANN Training Algorithm

Back-propagation algorithm is undoubtedly the most fundamental building block in an ANN. It It was first introduced in 1960s and almost 30 years later (1989) popularized by Rumelhart, Hinton and Williams in [50]. 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), back-propagation performs a backward pass while adjusting the model's parameters (weights and biases) (weights updating phase).

a, 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. 2.2.1)

Algorithm 2: 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;
```

2. Error values are calculated by the loss function and propagated back to previous layers.

b, 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 2.

2.2.4 Well-known machine learning models for time-series prediction problem

Recent developments in cloud computing including resource management have resulted in a significant interest in resource usage prediction . Various traditional methods such as Autoagressive model (AM), Autoregressive Moving Average (ARMA), Autoregressive Integrated Moving Average (ARIMA) and General Autoregressive Conditional Heteroscedastic (GARCH) have been proposed for solving this problem with different aspects, objectives and applications [51].

In this section, we focus on several Artificial Neural Network (ANN) models that are used for learning the time-series characteristic of sequence data. 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. [52]. 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 c.

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. 2.3, 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:

$$x(t + 1) = y = f(x_1, x_2, ..., x_t) \tag{2.11}$$

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.

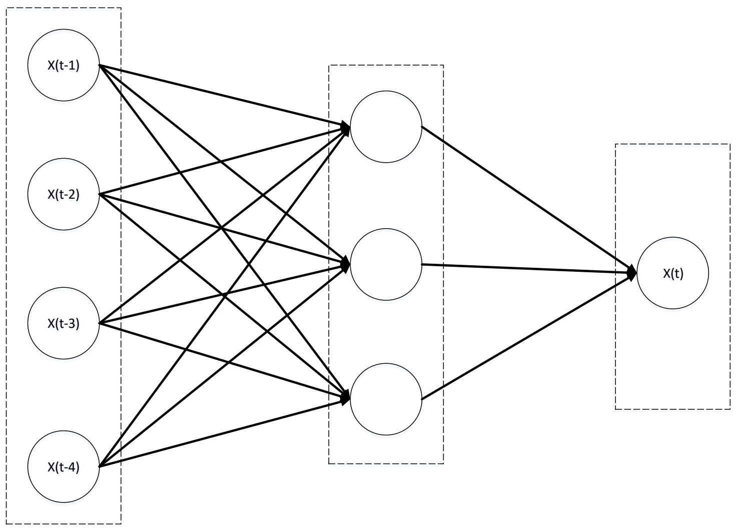


Figure 2.3: An example of MLPs model in time-series prediction.

a, 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 [53], [54]. Fig. 2.3 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:

$$H = f_h(W_h^T X + b_h) \tag{2.12}$$

$$y = O = f_o(W_o^T H + b_o \tag{2.13}$$

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.

b, 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 [55]. The architecture of CFNN with 4-neuron input layer is illustrated in Fig. 2.4, and the mathematical formulas for CFNN model are presented as follows:

$$H = f_h(W_h^T X + b_h) \tag{2.14}$$

$$C = f_c(W_c^T X) + b_c \tag{2.15}$$

$$y = O = f_o(W_o^T H + b_o) + C \tag{2.16}$$

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.

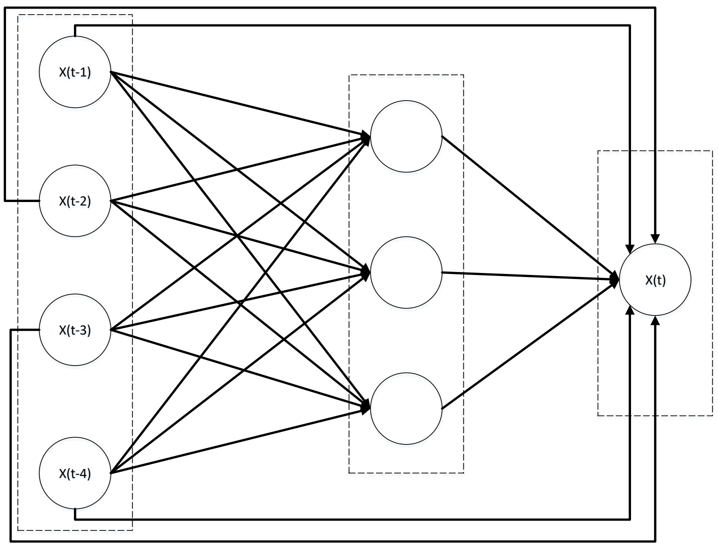


Figure 2.4: An example of CFNN model in time-series prediction.

c, 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. c). 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 [56], [57], [58]. 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.

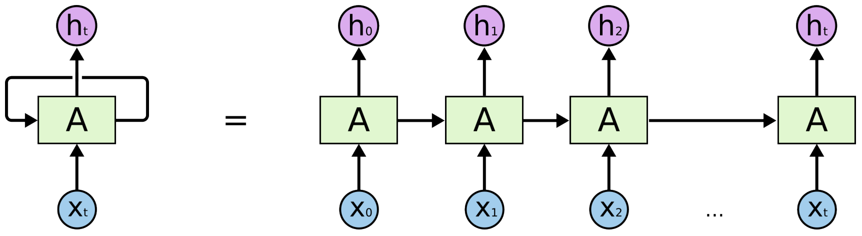


Figure 2.5: Traditional RNN architechture. Source: [59]

d, Long Short-Term Memory (LSTM)

Long short-term memory (LSTM) [60] 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 [61], [62], [63]. The architecture of LSTM units is illustrated in Fig. 2.6, and its mathematical model is briefly described as follows: The input gate (2.17) and the forget gate (2.18) manage the cell state (2.20), which is the long-term memory. The output gate (2.19) produces the output vector or hidden state (2.21), 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.

$$i_t = sigmoid(W_i x_t + U_i h_{t-1} + b_i) \tag{2.17}$$

$$f_t = sigmoid(W_f x_t + U_f h_{t-1} + b_f) \tag{2.18}$$

$$o_t = sigmoid(W_o x_t + U_o h_{t-1} + b_o) \tag{2.19}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot tanh(W_c x_t + U_c h_{t-1} + b_c) \tag{2.20}$$

$$h_t = o_t \odot tanh(c_t) \tag{2.21}$$

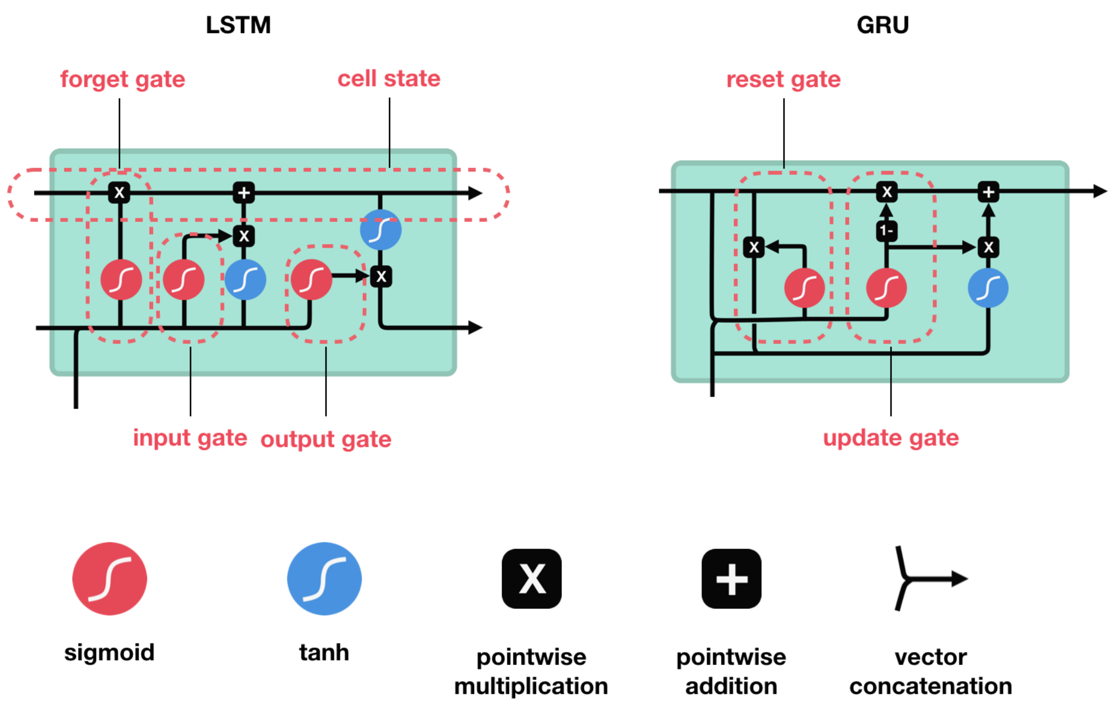


Figure 2.6: LSTM and GRU architechture. Source: [64]

e, Gated Recurrent Units (GRU)

Gated recurrent unit (GRU) [65] 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 [66] and [67] The srtucture of GRU units is presented in Fig. 2.6 following the mathematical model as below:

The update gate 2.22 controls the information flow from the previous activation, and the addition of new information as well 2.24, while the reset gate 2.23 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.

$$z_t = sigmoid(W_z x_t + U_z h_{t1} + b_z) \tag{2.22}$$

$$r_t = sigmoid(W_r x_t + U_r h_{t1} + b_r) \tag{2.23}$$

$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot tanh(W_h x_t + U_h(r_t h_{t-1}) + b_h) \tag{2.24}$$

2.3 Auto-scaling problem in Cloud Computing

2.3.1 Cloud Computing

Cloud technology has evolved from the costly and complex information technology (IT) solutions and enterprise applications in the 1980s, and is enabled by the recent expansion of the Internet in the 1990s. Moreover, the dramatic drop in the bandwidth costs and other technological advances have contributed to the emergence of cloud computing [68]. Unlike previous generations of application service providers (ASPs), cloud computing provides tangible and measurable business benefits by allowing multi-user-real-time access without the up-front investment cost. In general, cloud computing is a computing paradigm, where a large pool of systems are connected in private or public networks, to provide dynamically scalable infrastructure for application, data and file storage. With the advent of this technology, the cost of computation, application hosting, content storage and delivery is reduced significantly. Cloud computing is a practical approach to experience direct cost benefits and it has the potential to transform a data center from a capital-intensive set up to a variable priced environment.

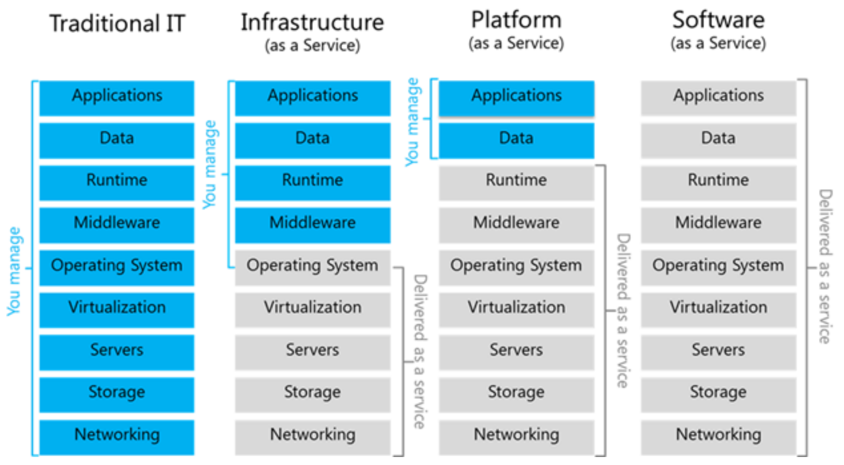


Figure 2.7: Comparision between Traditional IT and three models of Cloud computing. Source: [69])

Cloud Providers offer services that can be categorized into three categories (See Fig. 2.7):

1. **Software as a Service (SaaS):** In this model, a complete application is offered to customers as a service running on demand. A single instance of service runs on clouds, and multiple end users are served the same service. On the

provider's side, the costs are lower since there is only one application that needs to be built and maintained, while in customers' side, there is no need for upfront investment in servers and software licenses. Today, there is a number of companies such as Google, Microsoft and Zoho playing a role as SaaS providers.

2. **Platform as a Service (Paas):** In this category, a layer of software, or a development environment is encapsulated and offered as a service, upon which high level of services can be built. Customers are free to build their own application running on the provider's infrastructure. PaaS providers offer a predefined combination of OS and application servers, such as LAMP platform (Linux, Apache, MySql and PHP), restricted J2EE, Ruby etc. Googles App Engine, Force.com, etc are some of the popular PaaS examples.

3. **Infrastructure as a Service (Iaas):** Basic storage and computing capabilities are provided as standardized services over the network by Iaas. Servers, storage systems, networking equipment, data centre space etc. are pooled and made available to handle workloads. The customer would typically deploy his own software on the infrastructure. Some common examples are Amazon, GoGrid, 3 Tera.

2.3.2 Auto-scaling demand in Cloud Computing.

The popularity of on-demand cloud service has spurred the migration of increasing numbers of web applications to the cloud. One of the most attractive features for cloud web application providers is the ability to access computing resource elastically (by scaling up or down) according to dynamic resource demands. In this scenario, providers only pay for resources that are consumed at a specific point in time, which if operated correctly, will result in less cost and higher quality service than is achievable by hosting on standard hardware. Nevertheless, true elasticity and cost-effectiveness in the pay-per-use cloud business model have not yet been achieved completely. The management of allocating cloud resource adaptively to on-demand requirements of an application called auto-scaling, can be very challenging. Resource under-provisioning will unavoidably harm performance and cause Service Level Agreements (SLAs) violations, while resource over-provisioning will result in resource idle and cost waste. Therefore, the final objective of an auto-scaling mechanism is to automatically adjust acquired resources to minimize cost while satisfied the SLAs.

To achieve this target, several methods have been proposed. They can be divided into three groups of methods: Periodicity, Threshold and Prediction. Periodicity methods allow providers and customers to predefine time intervals, during which resource demands can be dilated or shrunk according to their experiences. The biggest drawback of this method is that system manager is unable to allocate suitable amount of resources when customer's requests come immediately. Unlike Periodicity, Threshold methods use lower bound L and upper bound U to define the minimum and maximum amount of resources that can be required. Adjusting resources is decided according to rules that are built by providers and customers. These methods face a problem of choosing suitable thresholds for satisfying resorces demand from applications. The third kind of methods, Prediction, is the most protential one. These methods use the information recorded over a period of time as time-series data to analyze and predict resources that can be required in the future. Since information collected have characteristics of time-series data, time-series forecasting models are usually used to tackle the workload predicting problem in cloud environment such as ARMA [70], [71] and ARIMA [72]. However, traditional methods do not show competitive performance in term of accuracy. Recently, deep learning predicting models have been proposed for solving the workload predicting demand. The works in [73], [74] and [75] have proved that deep learning models such as simple FFNN, RNN, LSTM or GRU can provide very competitive results compared with other traiditional methods.

Nevertheless, there are several issues still remaining in these models. The main problem is of the BP algorithm being used to train NNs model, which are tendency being trapped in local minima, slow convergence and significant dependence on the initial stage of their parameters. Also, RNN-based models like LSTM and GRU are too complicated that can lead to overfitting, and contain a huge number of parameters to be optimized. Therefore in this work, we focus on solving the workload prediction demand by a simpler and more effective appoach: propose a model using meta-heuristic algorithm for optimizing a much less complicated NNs model: CFNN.

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 original paper [2], 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 2.4 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 2.10 helps the new-born solution inherits 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 2.4, 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 [23] where the velocity of a particular particle is influenced by both the best individual and best personal information. This combination can take advantages of both the intelligence of the whole swarm and of each individual, and therefore there is more information for agents to exploit, leading to

Algorithm 3: Improved Sea Lion Optimization (ISLO)

```
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. 2.5
        if $SP_{leader} < 0.25$ then
            if $|C| < 1$ then
                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
            else
                Choose a random search agent $SL1_{rand}$ and $SL2_{rand}$
                Update the location of current search agent by Eq. 3.4
            end
        else
            Update the location of the current search agent by Eq. 2.8
        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: $X_*, f(X_*)$
```

more potential exploitation capacity of algorithms.

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:

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

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

$$SL(t+1) = c_1 r_1 (P(t) - Dist_1.C) + c_2 r_2 (P_i(t) - Dist_2.C) \tag{3.3}$$

where $P_i(t)$ is the personal best position of agent i up to time t, c_1, c_2 2 are positive constant parameters called acceleration coefficients 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 original 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) [76], which is successfully applied for enhancing bio-inspired optimization algorithms such as GA [76], PSO [77] [78] in solving several optimization problems including finding parameter for deep learning models [79] [80] 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 original 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 original 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.

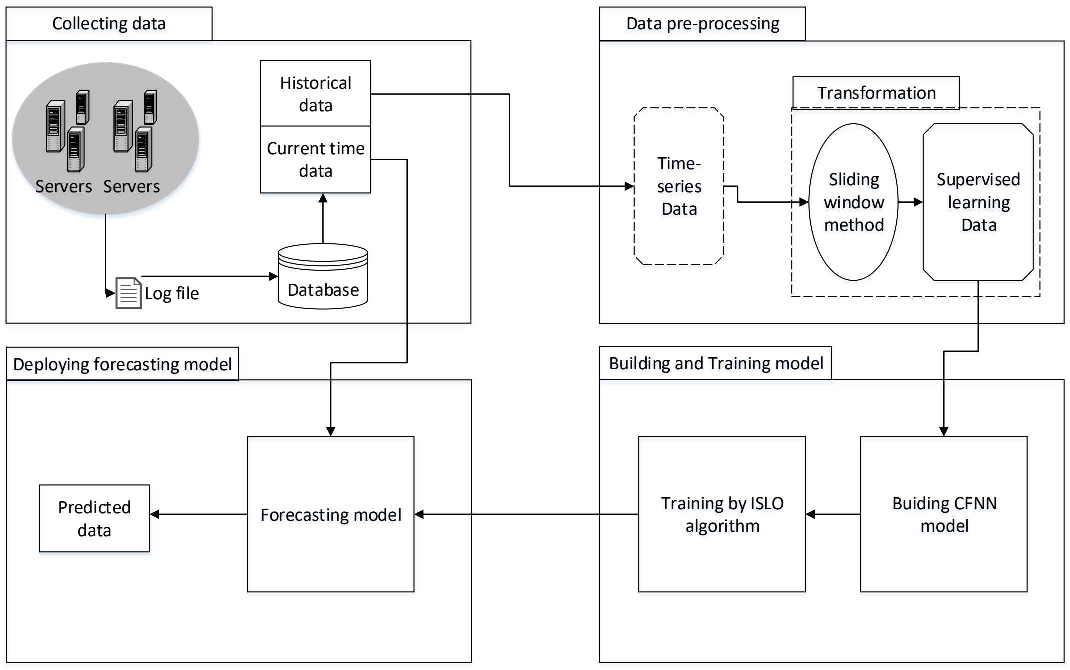


Figure 3.1: Proposed model design

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

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

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.

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

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
...	...	...	...	...

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.

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).

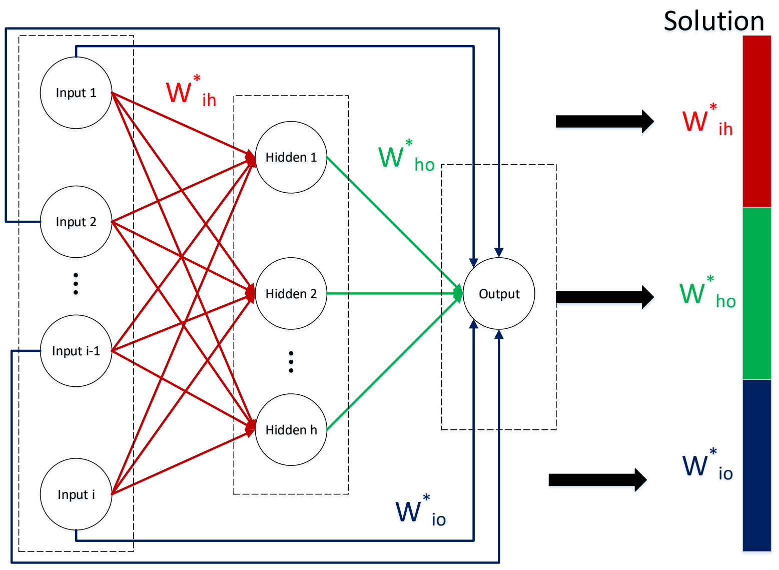


Figure 3.2: Encoding process transforming a parameter set in CFNN into an agent in ISLO algorithm. (W^* indicates weights and biases between 2 layers)

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.

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

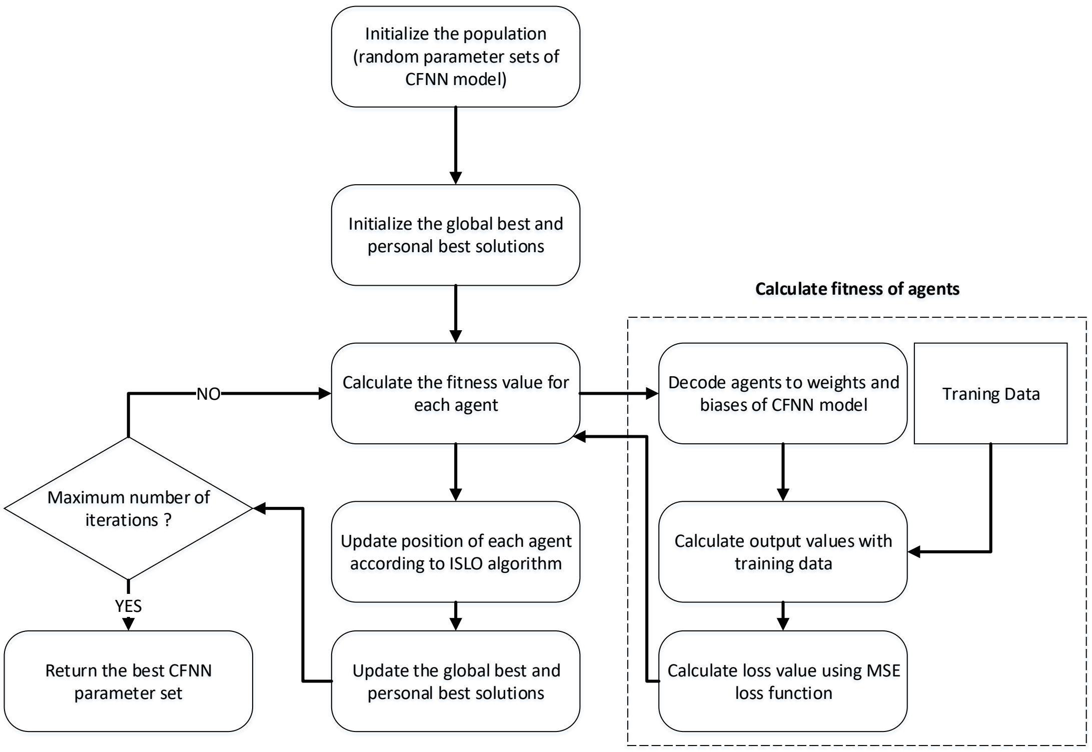

Initialize the population (random parameter sets of CFNN model)
Initialize the global best and personal best solutions
Calculate fitness of agents
Calculate the fitness value for each agent
Decode agents to weights and biases of CFNN model
Traning Data
NO
Maximum number of iterations ?
Update position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

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

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 AND DISCUSSION

The optimization capacity of ISLO algorithm developed in this study would be tested by solving two experiments: one for theoretical test and another for an optimizing application in the real , particularly the workload prediction problem.

In the theoretical test, 30 benchmark functions are used for testing the numerical efficiency of ISLO. The set of benchmark functions cover a wide range function groups including: classical unimodal and multimodal functions, hybrid functions and composition functions which are considered in CEC 2014 and CEC 2015 special session (See [81] and [82] for more information about the annual competition). ISLO will be compared with several well-known algorithms in all four groups in meta-heuristic optimization algorithms including evolutionary, swarm-based, physical-based and human-based algorithms.

On the other hand, the optimizing performance of ISLO is also tested in a time-series prediction problem. Specifically, the proposed model in 3.2 is used for the experiment with three real-world datasets, which are Google Trace data, EU Internet Traffic data and UK Internet Traffic data in different perspectives. The results are compared with that of several deep learning models that are widely used for time-series prediction, and additionally, the performance of ISLO in optimizing CFNN is also compared with several algorithms in the first experiment.

The detail of each experiment as well as results and analysis are presented as below.

4.1 Theoretical experiments

The performance of ISLO is theoretically experimented by 30 benchmark functions. They are divided into four groups of functions:

- Unimodal benchmark functions: they have only one global optimal point in the search space.
- Multimodal benchmark functions: they have one global optimal point going along with local minimum points.
- Hybrid functions: the variables are randomly divided into some sub-components and then different basic unimodal and multimodal functions are used for different sub-components.
- Composition functions: they merge the properties of the sub-functions better and maintains continuity around the global/local optima.

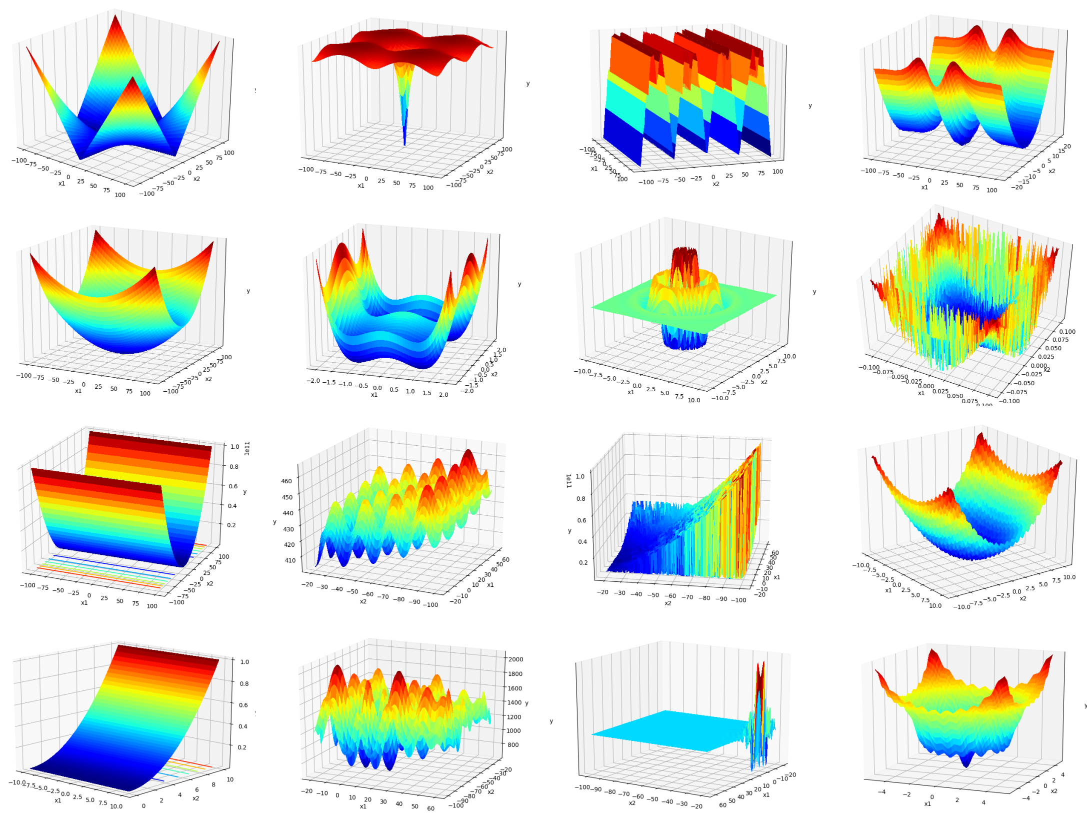

(a) Unimodal (b) Multimodal (c) Hybrid (d) Composition

Figure 4.1: Examples of 3D plot for each kind of benchmark functions.

The detail such as name, formula, search space and optimal value of each function is shown in Table. [4.1-4.4]. Also, Fig. 4.1 presents the typical 3D plots of the cost function for some test cases considered in this study.

The performance of ISLO algorithm about optimizing 30 benchmark functions is compared with different optimizers including well-known algorithms as well as recent algorithms that belong to all four groups in nature-inspired metaheuristic algorithms. Specifically, they are:

- Generic Algorithm (GA) [83] (Evolutionary algorithms)
- Particle Swarm Optimization (PSO) [48], Whale Optimization Algorithm (WOA) [32] and the original Sea Lion Optimization Algorithm (SLnO) [2] (Swarm-based algorithms)
- Tug of War Optimization (TWO) [84] (Physics-based algorithms)
- Queuing Search Optimization (QSO) [85] (Human-based algorithms)

4.1.1 Evaluation method and Parameter settings

With compared algorithms and functions mentioned above, the experimental results of each model are produced by calculating mean and standard deviation

Table 4.1: Description of unimodal benchmark functions

Mathematical Definition	Range	f_{min}
$f_1(x) = \sum_{i=1}^{n} x_i^2$	[-500, 500]	0
$f_2(x) = \sum_{i=1}^{n} \|x_i\|$	[-500, 500]	0
$f_3(x) = max_{i=1,2,...,n}\|x_i\|)$	[-500, 500]	0
$f_4(x) = \sum_{i=1}^{n} \|x_i\| + \prod_{i=1}^{n} \|x_i\|$	[-500, 500]	0
$f_5(x) = \sum_{i=1}^{n} ix_i^2$	[-500, 500]	0
$f_6(x) = \sum_{i=1}^{n-1}(x_i^2)^{x_{i+1}^2+1} + (x_{i+1}^2)^{x_i^2+1}$	[-500, 500]	0
$f_7(x) = \sum_{i=1}^{n}(10^6)^{\frac{x_i-1}{n-1}} + 100$	[-500, 500]	100
$f_8(x) = x_1^2 + 10^6 \sum_{i=2}^{n} x_i^2$ + 200	[-500, 500]	200

Table 4.2: Description of multimodal benchmark functions

Mathematical Definition	Range	f_{min}
$f_9(x) = -a.exp(-b\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}) - exp(\frac{1}{n}\sum_{i=1}^{n} cos(cx_i)) + a + exp(1)$ with a = 20 and b = 0.2	[-500, 500]	0
$f_{10}(x) = \left[\left(\|\|\mathbf{x}\|\|^2 - n\right)^2\right]^{\alpha} + \frac{1}{n}\left(\frac{1}{2}\|\|\mathbf{x}\|\|^2 + \sum_{i=1}^{n} x_i\right) + \frac{1}{2}$	[-500, 500]	0
$f_{11}(x) = \sum_{i=1}^{n}(x^2 - i)^2$	[-500, 500]	0
$f_{12}(x) = 1 - cos(2\pi\sqrt{\sum_{i=1}^{D} x_i^2}) + 0.1\sqrt{\sum_{i=1}^{D} x_i^2}$	[-500, 500]	0
$f_{13}f(x) = \sum_{i=1}^{n}[b(x_{i+1} - x_i^2)^2 + (a - x_i)^2]$	[-500, 500]	0
$f_{14}(x) = \sum_{i=1}^{n}\sum_{j=1}^{5} jsin((j+1)x_i + j) + 300$	[-500, 500]	300
$f_{15}(x) = 10n + \sum_{i=1}^{n}(x_i^2 - 10cos(2\pi x_i)) + 400$	[-500, 500]	400
$f_{16}(x) = 418.9829D - \sum_{i=1}^{n} x_i sin(\sqrt{\|x_i\|}). + 500$	[-500, 500]	500

(std) value (Eq. 4.1 and 4.2) of 20 times running starting from randomly generated populations for each algorithms. For all the algorithms, a population size and maximum iteration equal to 100 and 500 have been utilized to run on each function

Table 4.3: Description of hybrid benchmark functions

Mathematical Definition	Range	f_{min}
f_{17} (function 17 in CEC 2014) p = [0.3, 0.4, 0.3] Modified Schwefel's , Rastrigin's and High Conditioned Elliptic Functions	[-500, 500]	1700
f_{18} (function 18 in CEC 2014) p = [0.3, 0.4, 0.3] Bent Cigar, HGBat and Rastrigins Functions	[-500, 500]	1800
f_{19} (function 19 in CEC 2014) p = [0.2, 0.2, 0.3, 0.3] Griewanks, Weierstrass, Rosenbrocks and Scaffers Functions	[-500, 500]	1900
f_{20} (function 20 in CEC 2014) p = [0.2, 0.2, 0.3, 0.3] HGBat , Discus, Expanded Griewanks plus Rosenbrocks and Rastrigins Functions	[-500, 500]	2000
f_{21} (function 6 in CEC 2015) p = [0.3,0.3,0.4] Modified Schwefel's , Rastrigin's, High Conditioned Elliptic Functions	[-500, 500]	600
f_{22} (function 7 in CEC 2015) p =[0.2,0.2,0.3,0.3] Griewank's , Weierstrass ,Rosenbrock's and Scaffer's Functions	[-500, 500]	700
f_{23} (function 8 in CEC 2015) p = [0.1,0.2,0.2,0.2,0.3] Scaffers , HGBat ,Rosenbrocks, Modified Schwefels and High Conditioned Elliptic Functions	[-500, 500]	800

with 50-dimension search space. The choices of parameters are based on existing setting up described in original paper of each algorithm.

$$mean = \frac{1}{n}\sum_{i=1}^{n} r_i \tag{4.1}$$

$$std = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(r_i - r)^2} \tag{4.2}$$

where N is the number of values and r_i $(i = 1, 2, ..., N)$ are observations.

For each function, after calculating $mean$ and std values of each algorithm, the best results will be highlighted in both. The best results are determined by following rules:

Table 4.4: Description of composition benchmark functions

Mathematical Definition	Range	f_{min}
f_{24} (function 9 in CEC 2015) σ = [20, 20, 20] λ = [1, 1, 1] Schwefel's , Rastrigin's and HGBat Functions	[-500, 500]	900
f_{25} (function 10 in CEC 2015) σ = [10, 30,50] λ = [1, 1, 1] f_{21}, f_{22} and f_{23} Functions	[-500, 500]	1000
f_{26} (function 11 in CEC 2015) σ = [10, 10, 10, 20, 20] λ = [10, 10, 2.5, 25,1e-6] HGBat , Rastrigins and Schwefel's, Weierstrass and High Conditioned Elliptic Functions	[-500, 500]	1100
f_{27} (function 12 in CEC 2015) σ = [10,20,20,30,30] λ = [0.25, 1, 1e-7, 10, 10] Schwefel's , Rastrigin's and High Conditioned Elliptic, Expanded Scaffers and HappyCat Functions	[-500, 500]	1200
f_{28} (function 13 in CEC 2015) σ = [10, 10, 10, 20, 20] λ = [1, 10, 1, 25, 10] f_{23} , Rastrigin's and f_{21}, Schwefel's and Expanded Scaffers Functions	[-500, 500]	1300
f_{29} (function 14 in CEC 2015) σ = [10, 20, 30, 40, 50, 50, 50] λ = [10,2.5, 2.5, 10,1e-6,1e-6, 10] HappyCat , Griewanks plus Rosenbrocks, Schwefel's, Expanded Scaffers, High Conditioned Elliptic, Cigar and and Rastrigins Functions	[-500, 500]	1400
f_{30} (function 15 in CEC 2015) σ = [10, 10, 20, 20, 30, 30, 40, 40, 50, 50] λ = [0.1,2.5e-1, 0.1, 2.5e-2, 1e-3, 0.1, 1e-5, 10, 2.5e-2, 1e-3] Rastrigins , Weierstrass, HappyCat, Schwefel's, Rosenbrock's, HGBat, Katsuura, Expanded Scaffers, Expanded Griewanks and Ackley Functions	[-500, 500]	1500

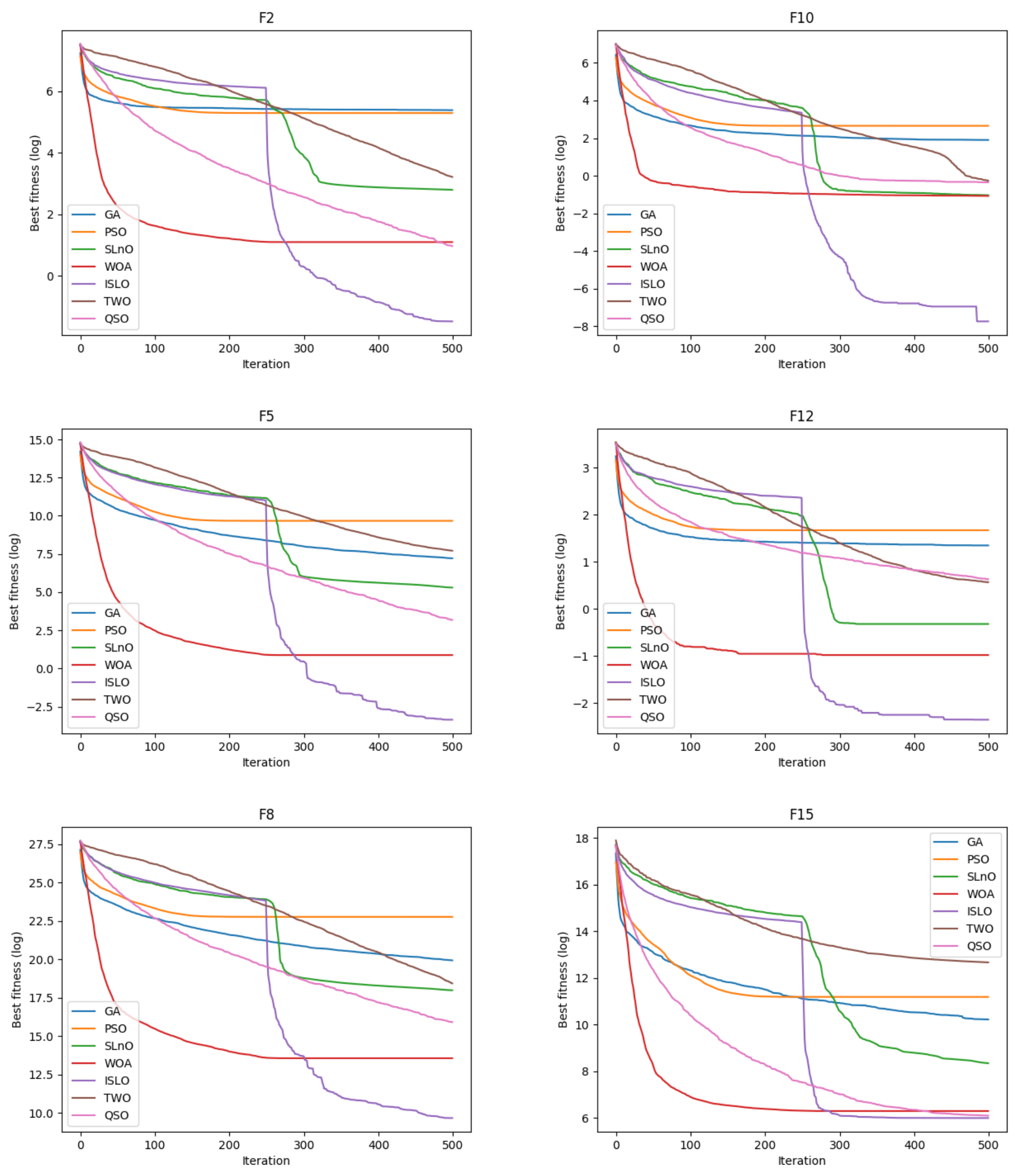
- $mean$ values are considered at first. If in a case, an algorithm own the best $mean$ value, it will be ranked as the best optimizer.
- In the case where there are two or more algorithms having the same $mean$ value, the one that has the most stable std value will be chosen as the best

one.

Finally, the experimental results of all functions and algorithms are shown in Table 4.5 and 4.6, and the convergence speeds of the algorithms in several functions are illustrated in Fig. 4.2 and 4.3.

4.1.2 Experiment results and discussion

a, Unimodal and Multimodal benchmark functions



(a) Unimodal (b) Multimodal

Figure 4.2: Convergence speed of each algorithm on unimodal (a) and multimodal (b) functions.

Functions f_1 - f_{16} are unimodal and multimodal functions. These kinds of

Table 4.5: Comparison of optimization results obtained for the unimodal and multimodal functions

Function		GA	PSO	SLnO	WOA	ISLO	TWO	QSO
f_1	mean	4.64E+01	9.69E+02	2.52E-25	2.26E-80	**3.26E-120**	1.06E+01	8.14E-01
	std	7.54E+00	3.12E+02	7.11E-25	9.41E-80	**8.45E-120**	1.39E+00	3.08E-01
	rank	6	7	3	2	**1**	5	4
f_2	mean	2.19E+02	1.99E+02	1.64E+01	2.98E+00	**2.26E-01**	2.49E+01	2.62E+00
	std	1.44E+01	3.42E+01	2.38E+00	5.86E-01	**1.23E-01**	2.24E+00	6.15E-01
	rank	7	6	4	3	**1**	5	2
f_3	mean	4.30E+00	1.79E+01	7.58E+01	4.39E+01	**6.25E-58**	1.58E+01	1.82E+01
	std	1.11E+00	2.28E+00	1.92E+01	2.29E+01	**2.64E-57**	3.93E+00	1.43E+00
	rank	2	4	7	6	**1**	3	5
f_4	mean	1.12E+02	7.43E+02	4.07E+26	2.81E+00	**1.56E-01**	1.71E+32	1.79E+02
	std	7.57E+00	1.22E+02	1.30E+27	7.00E-01	**1.01E-01**	7.33E+32	1.98E+02
	rank	3	5	6	2	**1**	7	4
f_5	mean	1.36E+03	1.58E+04	2.01E+02	2.42E+00	**3.52E-02**	2.24E+03	2.43E+01
	std	3.87E+02	5.57E+03	9.51E+01	6.70E-01	**6.06E-02**	1.06E+03	1.20E+01
	rank	5	7	4	2	**1**	6	3
f_6	mean	1.27E+00	1.20E+00	3.27E-01	4.71E-03	**5.65E-05**	1.02E+00	7.83E-02
	std	3.01E-02	4.08E-02	1.47E-01	1.40E-03	**5.50E-05**	2.21E-02	4.22E-02
	rank	7	6	4	2	**1**	5	3
f_7	mean	8.70E+06	2.34E+07	1.42E+06	6.12E+04	**8.22E+02**	1.22E+08	1.27E+04
	std	2.66E+06	1.17E+07	1.12E+06	2.05E+04	**1.10E+02**	4.97E+07	5.12E+03
	rank	5	6	4	3	**1**	7	2
f_8	mean	4.54E+08	7.70E+09	6.52E+07	7.83E+05	**1.60E+04**	1.02E+08	8.19E+06
	std	1.10E+08	2.70E+09	2.65E+07	2.47E+05	**1.20E+04**	1.42E+07	3.40E+06
	rank	6	7	4	2	**1**	5	3
f_9	mean	1.69E+01	2.04E+01	2.05E+01	2.82E-01	**1.85E-02**	2.01E+01	2.08E+01
	std	3.02E-01	8.70E-01	3.00E-01	2.22E-01	**1.11E-02**	3.89E-02	2.81E-02
	rank	3	5	6	2	**1**	4	7
f_{10}	mean	6.71E+00	1.42E+01	3.55E-01	3.41E-01	**4.34E-04**	7.68E-01	7.05E-01
	std	9.85E-01	3.00E+00	8.31E-02	1.35E-01	**1.89E-03**	1.25E-01	6.88E-02
	rank	6	7	3	2	**1**	5	4
f_{11}	mean	2.11E+04	2.99E+04	5.25E+03	**3.14E+03**	9.01E+03	5.20E+03	6.09E+03
	std	2.43E+03	1.50E+04	3.25E+03	**1.42E+03**	5.01E+02	1.77E+03	7.24E+02
	rank	6	7	3	**1**	5	2	4
f_{12}	mean	3.84E+00	5.30E+00	7.25E-01	3.75E-01	**9.50E-02**	1.76E+00	1.87E+00
	std	3.09E-01	5.25E-01	6.98E-02	9.94E-02	**2.00E-02**	3.93E-01	2.05E-01
	rank	6	7	3	2	**1**	4	5
f_{13}	mean	8.18E+04	4.77E+06	1.45E+03	5.63E+01	**3.85E-01**	5.40E+04	6.70E+03
	std	1.97E+04	2.83E+06	5.67E+02	2.75E+00	**3.67E-01**	9.53E+04	4.26E+03
	rank	6	7	3	2	**1**	5	4
f_{14}	mean	3.17E+02	3.20E+02	3.21E+02	3.00E+02	**3.00E+02**	3.20E+02	3.21E+02
	std	3.42E-01	7.51E-01	2.93E-01	1.92E-01	**1.37E-02**	3.78E-02	2.91E-02
	rank	3	5	7	2	**1**	4	6
f_{15}	mean	2.75E+04	7.21E+04	4.21E+03	5.42E+02	**4.02E+02**	3.17E+05	4.46E+02
	std	9.46E+03	3.66E+04	2.29E+03	5.50E+01	**2.43E+00**	1.28E+05	3.07E+01
	rank	5	6	4	3	**1**	7	2
f_{16}	mean	5.21E+02	5.21E+02	5.21E+02	5.03E+02	**5.00E+02**	5.20E+02	5.21E+02
	std	4.48E-02	1.66E-01	1.75E-01	5.94E+00	**3.80E-01**	3.76E-02	1.99E-02
	rank	5	6	7	2	**1**	3	4

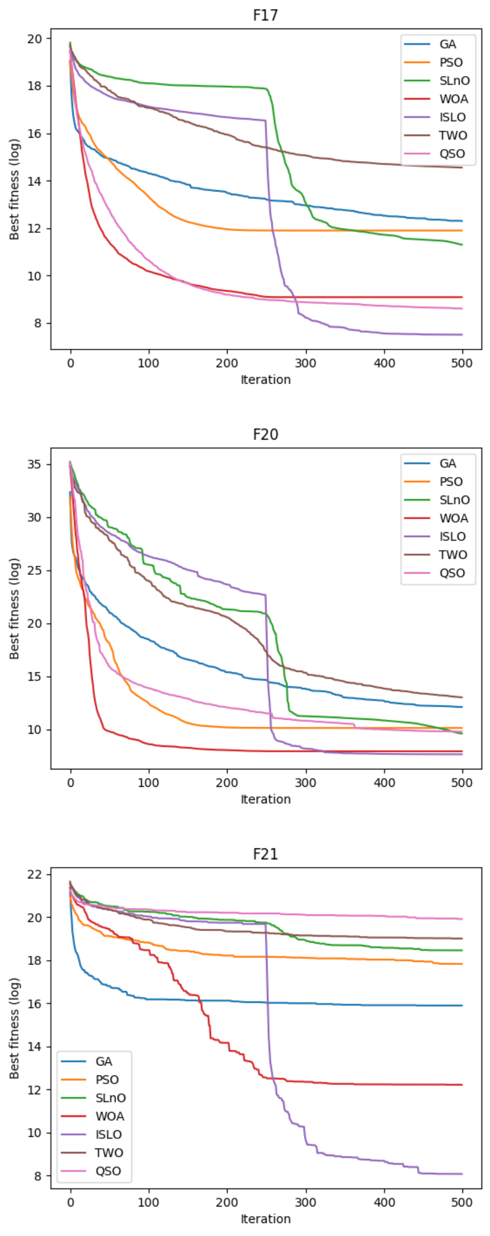
functions are selected following a couple of testing purposes. Specifically, unimodal functions allow us to evaluate exploitation capacity of meta-heuristic optimizers since they only have one global optimal minimum; multimodal functions help us see algorithms' exploration performance with a number of local minimum points, which exponentially increases following the increase in search space dimension. In general, it can be seen from the Table 4.5 that ISLO shows the best performance among all chosen algorithms in the most test cases except f_{11}. Furthermore, while optimizing several functions, ISLO is able to reach or nearly the optimal value with decent stability.

The accuracy and the stability: From the gained results of unimodal and multimodal funtions in Table. 4.5, it could be made the following observations:

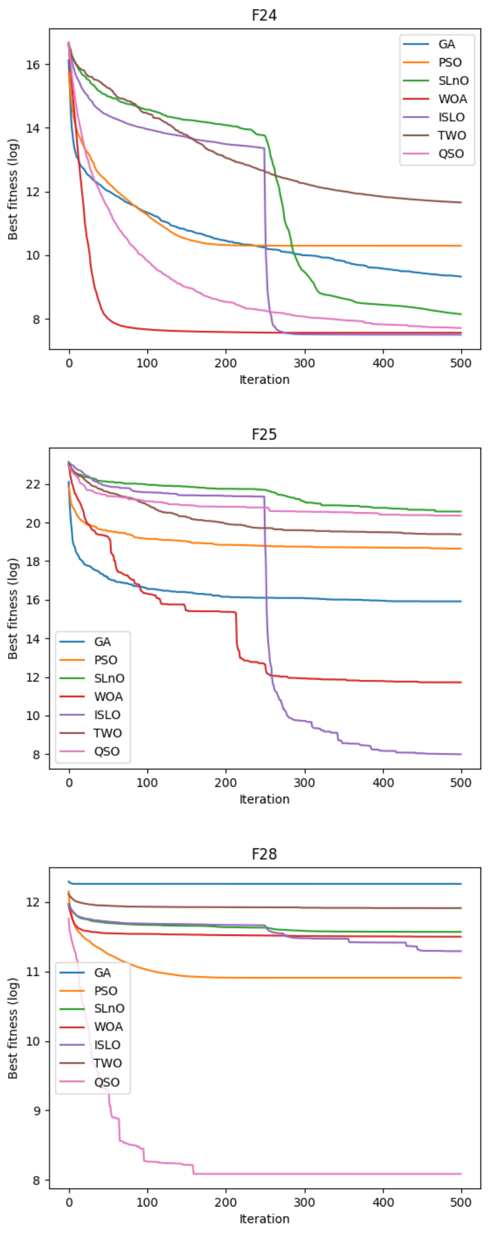
- ISLO outperforms the others in all test cases except f_{11}. In the experiments with unimodal functions f_1-f_8, ISLO is the best algorithm in the term of exploitation capacity, being ranked at the first position among all chosen algorithms. The results with functions f_1 and f_3 indicate that ISLO could lead the population relatively near the global optimal position. Additionally, along with the best results in term of accuracy, ISLO also performs good stability since the standard deviation values are below 1 in all cases, especially in f_1 and f_3, the std values are mostly 0. The results prove that compared with the original SLnO, ISLO's exploitation ability is significantly enhanced.
- The results in Table. 4.5 for multimodal functions f_9-f_{16} indicate that ISLO also has a superior exploration ability. ISLO stands at the first ranking in 7 out of 8 functions, and with f_{14} and f_{16} ISLO reaches the global optimal values of the functions, accompanying with that is relatively small standard deviation values. It is notable that ISLO is less competitive at f_{11} where it stands at $5th$ position, being outperformed by WOA, TWO and original SLnO. However, in f_8, f_9, f_{10}, f_{13}, f_{14} and f_{16}, ISLO's results are much better than the others in both terms of accuracy and stability.

In summary, all gained results with unimodal and multimodal funtions figure out that ISLO has the excellent exploitation and exploration capacity. This is due to the fact that apart from considering and updating search agents' position following the best agent, taking the best experience of each agent into account (Eq. 3.3) helps ISLO exploits population's information far better than SLnO. Apart from this, opposition-based updating operation in the exploration phase enrichs the diversity of population, helping ISLO easily jumps out of local minimums.

The convergence speed: The convergence speed of all algorithms working on several functions are shown in Fig. 4.2. It is clear that ISLO always considerably enhance its global best fitness values in the second half of the iterations. The reason is that on the first half of th e iterations, ISLO is in its exploration phase (since the value of C during that time is always greater than 1, see Algorithm 3). After changing to exploitation phase, ISLO has the ability to exploit and converge to global minimum quickly, providing better results than the others. Fig. 4.2 indicates that ISLO is competitive with an acceptable convergence speed, and also providing decent fitness values in functions f_2, f_5, f_8, f_{10}, f_{12} and f_{15}. SLnO has the same pattern of convergence curves compared with ISLO, but it coverges to mediocre fitness values, especially in f_2, f_5 and f_{10}. Also, WOA and QSO are competitive compared with ISLO in function f_{15}, where the final results of them are extremely close to ISLO's.



(a) Hybrid



(b) Composition

Figure 4.3: Convergence speed of each algorithm on hybrid (a) and composition (b) functions.

Table 4.6: Comparison of optimization results obtained for the hybrid and composition benchmark functions

Function		GA	PSO	SLnO	WOA	ISLO	TWO	QSO
f_{17}	mean	2.19E+05	1.46E+05	8.08E+04	8.82E+03	**1.81E+03**	2.08E+06	5.46E+03
	std	1.29E+05	7.68E+04	4.92E+04	2.63E+03	**1.95E+02**	7.79E+05	5.00E+02
	rank	6	5	4	3	**1**	7	2
f_{18}	mean	6.43E+06	3.15E+04	9.29E+05	8.12E+03	4.46E+03	4.73E+05	**3.49E+03**
	std	1.10E+06	9.41E+03	1.66E+06	2.55E+03	2.95E+03	1.00E+05	**8.92E+02**
	rank	7	4	6	3	2	5	**1**
f_{19}	mean	8.97E+03	2.36E+03	2.48E+03	1.95E+03	**1.92E+03**	9.67E+03	2.01E+03
	std	2.87E+03	9.31E+02	8.36E+02	4.45E+01	**2.08E+00**	2.02E+04	1.25E+02
	rank	6	4	5	2	**1**	7	3
f_{20}	mean	1.82E+05	2.52E+04	1.49E+04	2.81E+03	**2.12E+03**	4.53E+05	1.77E+04
	std	2.14E+05	1.24E+04	6.90E+03	6.75E+02	**8.47E+01**	1.03E+06	5.70E+04
	rank	6	5	3	2	**1**	7	4
f_{21}	mean	7.97E+06	5.51E+07	1.04E+08	2.00E+05	**3.19E+03**	1.79E+08	4.43E+08
	std	2.95E+06	3.76E+07	1.00E+08	4.37E+05	**9.01E+03**	8.02E+07	1.46E+08
	rank	3	4	5	2	**1**	6	7
f_{22}	mean	2.28E+05	1.26E+07	1.85E+08	8.72E+02	**7.16E+02**	1.17E+08	2.57E+08
	std	7.52E+04	1.27E+07	6.11E+08	2.55E+02	**9.38E-01**	1.29E+08	1.73E+08
	rank	3	4	6	2	**1**	5	7
f_{23}	mean	5.29E+06	6.08E+07	2.65E+08	3.59E+04	**1.26E+03**	2.74E+08	9.76E+08
	std	1.29E+06	2.72E+07	3.12E+08	4.30E+04	**1.19E+03**	1.57E+08	4.46E+08
	rank	3	4	5	2	**1**	6	7
f_{24}	mean	1.12E+04	2.96E+04	3.46E+03	1.92E+03	**1.81E+03**	1.15E+05	2.23E+03
	std	2.35E+03	1.71E+04	1.04E+03	4.11E+01	**1.62E+00**	4.71E+04	6.89E+02
	rank	5	6	4	2	**1**	7	3
f_{25}	mean	8.17E+06	1.26E+08	8.62E+08	1.23E+05	**2.95E+03**	2.64E+08	6.98E+08
	std	2.42E+06	5.10E+07	1.08E+09	1.15E+05	**4.98E+02**	1.09E+08	3.18E+08
	rank	3	4	7	2	**1**	5	6
f_{26}	mean	6.78E+03	4.50E+03	4.63E+03	3.03E+03	**2.22E+03**	4.20E+03	2.82E+03
	std	3.59E+02	3.76E+02	9.84E+02	9.60E+02	**2.03E+00**	2.15E+02	3.59E+01
	rank	7	5	6	3	**1**	4	2
f_{27}	mean	3.22E+03	3.15E+03	2.52E+03	2.43E+03	**2.40E+03**	2.87E+03	2.66E+03
	std	6.09E+01	1.18E+02	1.66E-02	3.28E+00	**1.50E+00**	2.58E+01	6.04E+00
	rank	7	6	3	2	**1**	5	4
f_{28}	mean	2.11E+05	5.47E+04	1.06E+05	9.88E+04	8.02E+04	1.49E+05	**3.25E+03**
	std	6.99E+03	1.22E+04	5.84E+03	7.98E+03	3.86E+04	3.89E+03	**1.81E+00**
	rank	7	2	5	4	3	6	**1**
f_{29}	mean	3.77E+05	1.13E+07	2.17E+13	2.92E+03	**2.83E+03**	2.39E+13	8.72E+06
	std	2.07E+05	1.10E+05	1.27E+11	4.78E+01	**5.36E+00**	3.05E+11	1.90E+06
	rank	3	5	6	2	**1**	7	4
f_{30}	mean	4.31E+03	1.68E+06	3.04E+03	2.93E+03	**2.91E+03**	1.12E+04	3.21E+03
	std	7.28E+02	1.94E+06	5.60E-01	2.30E+00	**1.32E+00**	1.54E+04	1.66E+02
	rank	5	7	3	2	**1**	6	4

b, Hybrid and Composition benchmark functions

Functions f_{17}-f_{30} are hybrid and composition functions. In hybrid functions (f_{17}-f_{23}, the variables are randomly divided into subcomponents which play a role as input for different basic functions including both unimodal and multimodal functions. In order to work well on these functions, algorithms are required to be good at both exploitation and exploration capacity, because hybrid functions are both unimodal and multimodal, and they own different properties for different variables subcomponents. On the other hand, optimization of composite mathematical functions (f_{24}-f_{30}) is a very challenging task, because local optima is only avoided by a proper balance between exploitation and exploration. In general, Fig. 4.6 shows that ISLO owns the best performance over all hybrid and compostion functions. The results from ISLO stand at first ranking in all cases except f_8 and f_{28} where the first position belongs to QSO. Also, as what is observed in Fig. 4.3, ISLO's convergence curves are similar to those in unimodal and multimodal functions, and ISLO still has a very fast convergence after the first half of iteration because of its updating mechanism.

The accuracy and the stability:

To resolve the optimization problem in hybrid functions f_{17}-f_{23}, it is evident that ISLO can work very well in almost cases. The results shown in Table. 4.6 indicates that ISLO has the superior results at functions f_{17}, f_{19}, f_{20} and f_{23} compared with state-of-the-art algorithms such as WOA and QSO. Furthermore, with functions f_{21} and f_{23}, ISLO's results are much better than the others', proving good capacity at both exploitation and exploration. Solving the case f_{18}, although ISLO does not account for the first place, it is still very competitive when its result is only worse than QSO's.

For composition functions, ISLO shows the best performance among all the algorithms by standing at the first place in 6 out of 7 functions. Specifically, in f_{24}, f_{27}, f_{29} and f_{30}, there is no big differences between ISLO'results and the others', while in f_{25} and f_{26}, $mean$ and std values from Table. 4.6 indicates the dominance of ISLO in optimizing these functions. This proves that ISLO owns a superior balance between its exploitation and exploration while solving test problems. Also std values from ISLO in most cases are below 10, showing the decent stability of this algorithm compared with GA or PSO algorithms.

The convergence speed: As working on unimodal and multimodal functions, when working on hybrid and composite mathematical functions, ISLO still owns the fast convergence in the second half of iterations. The convergence curves in Fig. 4.3 shows that ISLO starts to converge very fast right after exploration phase comes to an end. In f_{17} and f_{24}, the convergence curves indicate that ISLO is very competitive with WOA because these 2 algorithm converge to almost one value. Also, the results comes from ISLO is far better than the original SLnO in all cases, proving that exploitation and exploration capacities in SLnO are considerably enhanced. QSO is observed to be superior in function f_{28}, in which the others including ISLO are stuck in local minimums.

4.2 Application

In this section, proposed model in Section 2.3 is utilized for solving time-series prediction in the auto-scaling problem in cloud computing. Our experiment is done with 4 datasets: Google Trace CPU, Google Trace Memory, EU Internet Traffic and UK Internet Traffic. In this empirical study, ISLO-CFNN model is compared with several deep learning models such as simple MLP, the original CFNN, the original RNN and two well-known and widely used models in time-series forecasting: LSTM and GRU in terms of accuracy and run time. Also, optimizing capacity of ISLO on CFNN is compared with several swarm-based algorithms in Section 4.1. The bio-inspired models used to validate against ISLO-CFNN are PSO-CFNN and SLnO-CFNN, which are CFNN models optimized by PSO and SLnO algorithms, respectively.

We would first describe 4 datasets used in this experiment. Then, the parameter setting for each model and evaluation metrics are introduced in detail. Finally, ISLO-CFNN performance is compared with the deep learning models as well as bio-inspired models in different perspectives.

4.2.1 Dataset and Set up

a, Google Trace dataset

The most important dataset in our experiments is gathered by Google on a cluster of about 12500 machines [86] during 29 days, starting from May 2011. Resources requirement and usage data for each jobs are recorded by each machine in cluster, and then the data is managed by cluster's management system. In Google Trace dataset, there are two important columns, which are about two extremely important information of Central Processing Unit (CPU) and Random Access Memory (RAM) required for each job. For that reason, we decide to choose these two information as two time-series datasets (called Google Trace CPU and Google Trace RAM from here). The datasets are processed and summarized in 5-minute interval, containing 8351 data points, and considered as the total demand for resources in the whole Google's cluster. Visualization of Google Trace CPU and Google Trace RAM datasets is illustrated in Fig. 4.4.

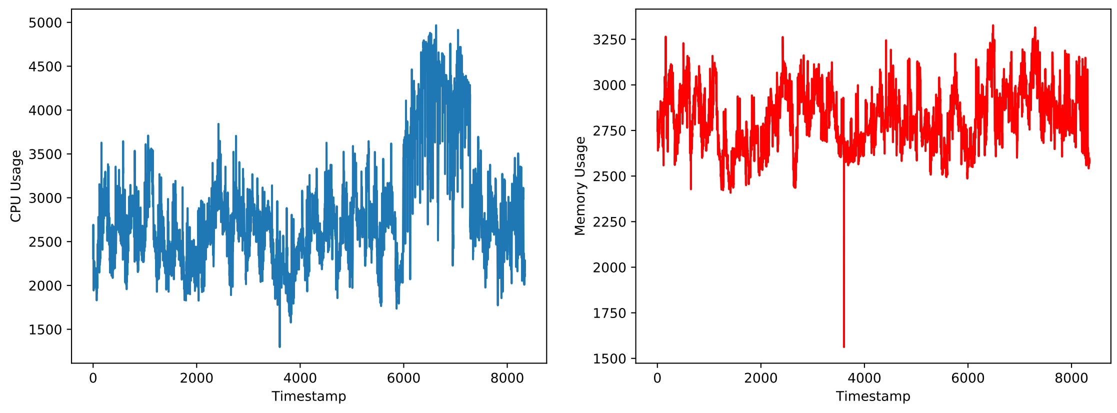


Figure 4.4: Visualization of Google Trace CPU (left) and Google Trace RAM (right) datasets.

b, EU Internet Traffic and UK Internet Traffic datasets

These two sets of data, which is used for experiments in [87], are recorded by two distinct ISPs. The EU Internet Traffic dataset comes from a private ISP playing a role as a reporter with centers in 11 European cities. The data corresponds to a a transatlantic link and was collected from 06:57 hours on 7 June to 11:17 hours on 29 July 2005. The UK Internet Traffic is derived from m UKERNA and represents aggregated traffic in the United Kingdom academic network backbone. It was reported between 19 November 2004, at 09:30 hours and 27 January 2005, at 11:11 hours. Both of two datasets are processed and summarized in every 5 minutes, creating EU Internet Traffic (14773 records) and UK Internet Traffic (19989 records) as the input in our experiments. 2D visualization of the data is shown in Fig. 4.5 as below.

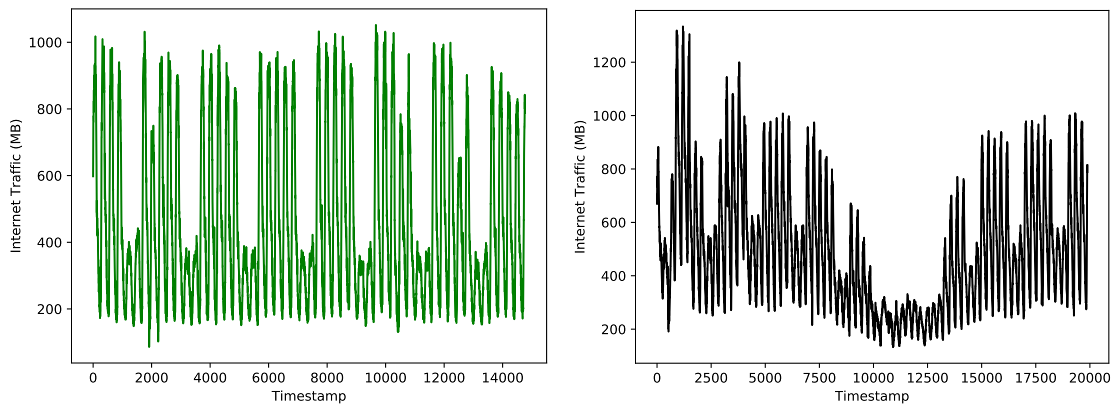


Figure 4.5: Visualization of EU Internet Traffic (left) and UK Internet Traffic (right) datasets.

4.2.2 Parameter Setting and Evaluation Metrics

As mentioned above, ISLO-CFNN's performance is compared with 4 deep learning models: MLPs, CFNN, LSTM and GRU, and 2 bio-inspired models: PSO-CFNN and SLnO-CFNN models. The hyper-parameter settings for each model are described as below:

- 4 datasets used for these experiments are all divided into 2 sets: training set and testing set with the ratio 0.8:0.2. The training set accounts for the first 80% of the datasets, and the remaining is of testing set because of the sequential characteristic of time-series data.
- CFNN's architectures in all models are configured with the same structure with three layers: input layer, one hidden layer and output layer which contains only one neuron in time-series prediction.
- The input size for all models is 3 as we use 3 historical data points to predict the output in current time (mentioned in Section 2.3).
- RNN-based models as LSTM and GRU contains one input layer, LSTM (or GRU) blocks and one output layer with the same hyper-parameter setting as described in [63].
- In all test, including with deep learning models and swarm-based algorithms, the number of iterations is set to 1000. From the experiments, we figure out that the amount of 1000 iterations is enough for all algorithms to converge to their final results.

Besides the common settings for models, the following settings are applied for each swarm-based algorithms as they show the best empirical performance:

- The population size for each algorithm is set to 200.
- For PSO, based on the original paper [23], cognitive learning rates $c_1 = c_2 = 2.05$, and inertia factor w is set linearly reducing from 0.9 to 0.4 over the course of iteration.
- For SLnO and ISLO, hyper-parameters are set as described in original paper [2], and also, c_1 and c_2 in ISLO algorithm are the same as shown for PSO.

All 8 models' performance on 4 datasets are compared with each other in term of accuracy. In our experiments, mean absolute error (MAE) is used mainly for evaluation purposes. Beside MAE, root mean square error (RMSE) and median absolute error (MedAE) are also used as measurements for comparision. These three measurement methods are very popular and widely used for evaluation in

Table 4.7: Comparision between models on each dataset by different measurements.

Dataset	Model	RMSE	MAE	MedAE
Google Trace CPU	CFNN	**199.77**	126.80	80.05
	FFNN	203.54	129.52	85.69
	LSTM	200.89	129.88	84.65
	GRU	201.88	130.50	82.96
	RNN	**200.40**	129.30	85.49
	PSO-CFNN	203.75	127.41	79.74
	SLnO-CFNN	201.40	**126.60**	**79.64**
	ISLO-CFNN	203.55	**124.94**	**76.08**
Google Trace RAM	CFNN	52.54	35.35	24.87
	FFNN	53.61	36.51	24.29
	LSTM	48.47	30.40	20.51
	GRU	48.43	30.74	20.78
	RNN	**47.00**	**28.86**	**18.90**
	PSO-CFNN	49.09	31.33	20.67
	SLnO-CFNN	49.33	31.62	20.53
	ISLO-CFNN	**47.53**	**29.80**	**19.65**
EU Internet Traffic	CFNN	**15.85**	**11.30**	**8.01**
	FFNN	16.48	11.79	8.41
	LSTM	**15.84**	**11.37**	**8.30**
	GRU	17.39	13.09	10.29
	RNN	16.90	12.49	9.55
	PSO-CFNN	16.02	11.49	8.31
	SLnO-CFNN	16.37	11.74	8.47
	ISLO-CFNN	16.02	11.47	8.31
UK Internet Traffic	CFNN	10.74	8.09	6.31
	FFNN	11.32	8.40	6.45
	LSTM	**10.34**	7.65	5.85
	GRU	11.52	8.89	7.33
	RNN	11.27	8.38	6.39
	PSO-CFNN	10.43	**7.59**	**5.63**
	SLnO-CFNN	10.61	7.73	5.75
	ISLO-CFNN	**10.46**	**7.64**	**5.77**

time-series prediction. Table. 4.7 presents the results of all models on each dataset evaluated by MAE, RMSE and MedAE measurements. It is worth to mention here that the 2 best results on each evaluation measurement would be highlighted in bold. Also, Fig. 4.6 illustrates the comparision between predicted output and the ground truth from ISLO-CFNN and RNN-based models running on Google Trace CPU dataset, while Fig. 4.7 shows the performance of different optimizers including ISLO, SLnO, PSO and original Gradient Descent in optimizing CFNN parameters, running on Google Trace RAM dataset.

4.2.3 Results and Discussion

a, Accuracy comparision

Table 4.7 illustrates the comparision between ISLO-CFNN and other models on each dataset by different measurements. In general, ISLO-CFNN is very competitive in working on all dataset. Also, while ISLO provides good results in MAE and MedAE, RNN-based models (RNN, LSTM, GRU) and original CFNN show decent performance by giving the best RMSE.

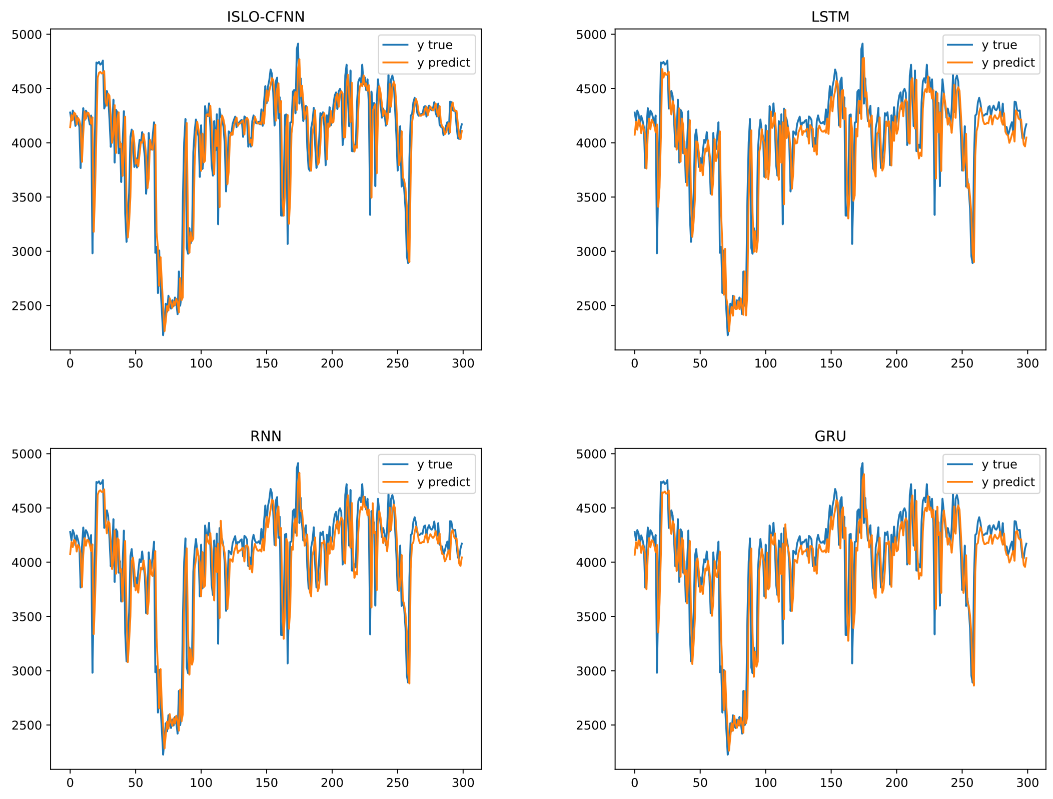


Figure 4.6: Performance comparision between ISLO-CFNN and different recurrent-based deep learning models on Google Trace CPU data.

On Google Trace CPU dataset, ISLO-CFNN shows its outstanding performance, giving the best results in MAE and MedAE measurements. It is obvious that compared with RNN-based models, our CFNN-ISLO shows the superior performance, and provides better results (see Fig. 4.6). Specifically, the model's figures are better than LSTM by 3.9% (with MAE) and about 10% (with MedAE). CFNN and RNN provides the smallest RMSE on the dataset, while bio-inspred models including SLnO-CFNN and PSO-CFNN have advantages in MAE and MedAE.

Working on Google Trace RAM and UK Internet Traffic datatsets, ISLO

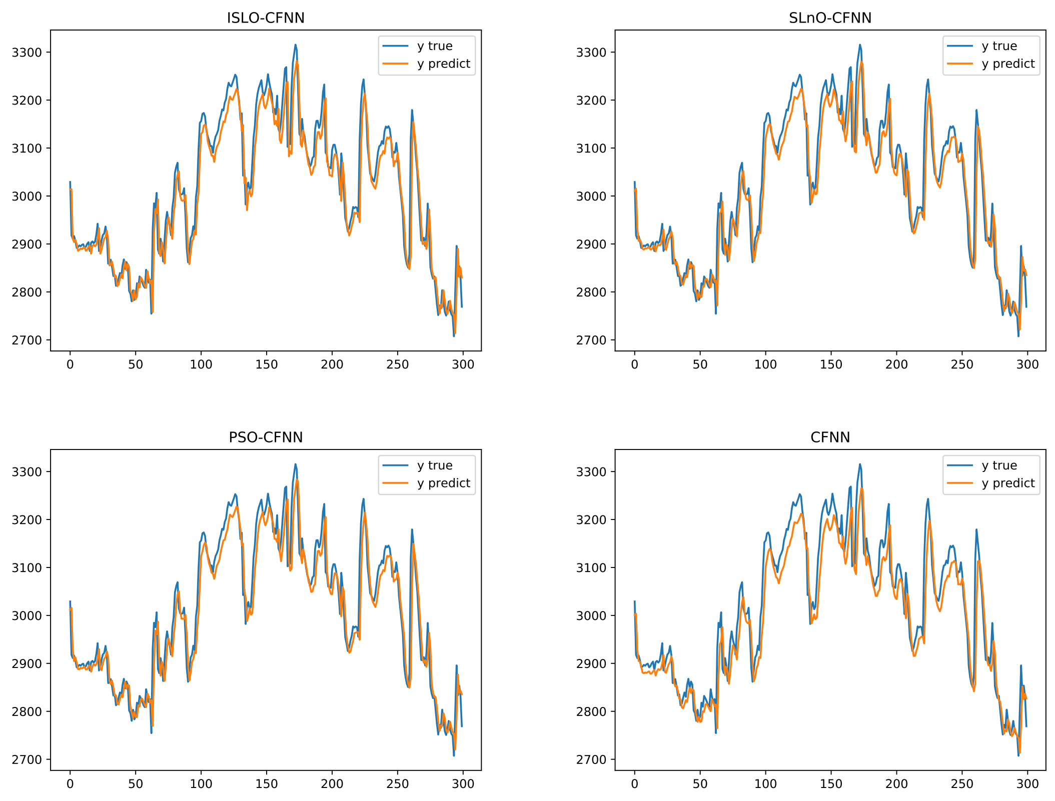


Figure 4.7: Performance comparision between ISLO and other algorithms including Gradient Descent, PSO and SLnO on optimizing CFNN (Google Trace RAM data).

shows a competitive performance compared with the others, standing at the second position in terms of all measurements. ISLO-CFNN's results ony stand behind original RNN (Google Trace RAM) and PSO-CFNN (UK Internet Traffic). With Google Trace RAM dataset, ISLO-CFNN's performance is more superior than that of original CFNN by 9.6% (RMSE) and 16% (MAE), and with the others the model also show its prominence. Furthermore, Table 4.7 shows that in both datasets, the gap between ISLO-CFNN's results and the best figures is trivial, proving that the model is able to work very well with them. It is worth mentioned here that compared with Gradient Descent, swarm-based optimization algorithms such as ISLO, SLnO and PSO show better performance on optimizing CFNN, working on Google Trace RAM data (see Fig. 4.7)

With the results illustarted in Table 4.7 about EU Internet Traffic data, it is obvious that this is the data in which our model is less competitive than its performance on the other data. Original CFNN and LSTM provide the outstanding and mostly the same figures. Three swarm-based models including PSO-CFNN, ISLO-CFNN and SLnO-CFNN stand at right behind the best one, while GRU and RNN are much less competitive than the others.

CHAPTER 5. CONCLUSION AND FUTURE WORKS

In this thesis, we proposed an improved version of Sea Lion Optimization algorithm called ISLO. As described in Section 3.1, ISLO it first improves the exploitation phase by using an idea from PSO, taking the individuals' best experience into consideration, and use 2 random parameters to balance the influences of the global best agent and individual's experience for enhancing exploitation ability of this algorithm. Secondly, a modification based on opposition-based learning is utilized in exploration phase. This method chooses opposite position of a random agent over the best agent instead of random existing one in current population, helping ISLO avoid the degradation in diversity of population over the course of iteration. ISLO performance is tested by experiments on 30 benchmark functions, and is compared with several well-known meta-heuristic optimization algorithms in Section 4.1. The experimental results prove that the proposed algorithm is superior to other compared algorithms in the search capability. Therefore. it is evident that proposed modifications can provide better performance than the original SLnO in terms of accuracy and robustness.

On the other hand, we introduced a model based on ISLO and CFNN to solve the auto-scaling demand in cloud environments in Section 3.2. In this model, ISLO plays an important role as an optimizer replacing traditional BP algorithm for training the neural network. ISLO-CFNN's performance is tested on 4 time-series datasets of workload, and compared with other well-known deep learning models, which are widely used in solving time-series forecasting problem. Also, ISLO's optimizing capacity is experimented again and compared with other swarm-based algorithm in optimizing CFNN's parameters. The results provided in Section 4.2 show that ISLO-CFNN provides very competitive results to the others in term of accuracy, even with a simple architechture of CFNN.

Although this work tackles the problems SLnO faces, and significantly enhances performance of the algorithm, the convergence speed is still limited, so in the future, we will conduct research on the issue of accelerating the convergence speed of this algorithm. Furthermore, the improved algorithm will be applied to tuning hyper-parameters problem in deep learning models, which takes huge amount of time and resources.

REFERENCES

[1] J. H. Holland, “Genetic algorithms,” *Scientific american*, vol. 267, no. 1, pp. 66–73, 1992.

[2] R. Masadeh, B. A. Mahafzah, and A. Sharieh, “Sea lion optimization algorithm,” *Sea*, vol. 10, no. 5, 2019.

[3] M. Chawla and M. Duhan, “Levy flights in metaheuristics optimization algorithms–a review,” *Applied Artificial Intelligence*, vol. 32, no. 9-10, pp. 802–821, 2018.

[4] H. Wang, Z. Wu, and S. Rahnamayan, “Enhanced opposition-based differential evolution for solving high-dimensional continuous optimization problems,” *Soft Computing*, vol. 15, no. 11, pp. 2127–2140, 2011.

[5] M. G. Omran, Z. W. Geem, and A. Salman, “Improving the performance of harmony search using opposition-based learning and quadratic interpolation,” *International Journal of Mathematical Modelling and Numerical Optimisation*, vol. 2, no. 1, pp. 28–50, 2010.

[6] G.-G. Wang, S. Deb, A. H. Gandomi, and A. H. Alavi, “Opposition-based krill herd algorithm with cauchy mutation and position clamping,” *Neurocomputing*, vol. 177, pp. 147–157, 2016.

[7] K.-P. Wang, L. Huang, C.-G. Zhou, and W. Pang, “Particle swarm optimization for traveling salesman problem,” in *Proceedings of the 2003 international conference on machine learning and cybernetics (IEEE cat. no. 03ex693)*, IEEE, vol. 3, 2003, pp. 1583–1585.

[8] X.-h. Yuan, C. Wang, Y.-c. Zhang, and Y.-b. Yuan, “A survey on application of particle swarm optimization to electric power systems [j],” *Power System Technology*, vol. 19, p. 003, 2004.

[9] G. C. Karmakar and L. S. Dooley, “A generic fuzzy rule based image segmentation algorithm,” *Pattern Recognition Letters*, vol. 23, no. 10, pp. 1215–1227, 2002.

[10] A. Hore and D. Ziou, “An edge-sensing generic demosaicing algorithm with application to image resampling,” *IEEE Transactions on Image Processing*, vol. 20, no. 11, pp. 3136–3150, 2011.

[11] S. Ruder, “An overview of gradient descent optimization algorithms,” *arXiv preprint arXiv:1609.04747*, 2016.

[12] H. Liu, H.-q. Tian, C. Chen, and Y.-f. Li, “An experimental investigation of two wavelet-mlp hybrid frameworks for wind speed prediction using ga

and pso optimization," *International Journal of Electrical Power & Energy Systems*, vol. 52, pp. 161–173, 2013.

[13] I. Aljarah, H. Faris, and S. Mirjalili, "Optimizing connection weights in neural networks using the whale optimization algorithm," *Soft Computing*, vol. 22, no. 1, pp. 1–15, 2018.

[14] P. A. Castillo, J. Merelo, A. Prieto, V Rivas, and G. Romero, "G-prop: Global optimization of multilayer perceptrons using gas," *Neurocomputing*, vol. 35, no. 1-4, pp. 149–163, 2000.

[15] M. Nasseri, K. Asghari, and M. Abedini, "Optimized scenario for rainfall forecasting using genetic algorithm coupled with artificial neural network," *Expert systems with applications*, vol. 35, no. 3, pp. 1415–1421, 2008.

[16] A. A. Kawam and N. Mansour, "Metaheuristic optimization algorithms for training artificial neural networks," *Int. J. Comput. Inf. Technol*, vol. 1, no. 2, pp. 156–161, 2012.

[17] A. Y. Nikravesh, S. A. Ajila, and C.-H. Lung, "Towards an autonomic auto-scaling prediction system for cloud resource provisioning," in *Proceedings of the 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, IEEE Press, 2015, pp. 35–45.

[18] T. Lorido-Botrán, J. Miguel-Alonso, and J. A. Lozano, "Auto-scaling techniques for elastic applications in cloud environments," *Department of Computer Architecture and Technology, University of Basque Country, Tech. Rep. EHU-KAT-IK-09*, vol. 12, p. 2012, 2012.

[19] J. R. Koza, "Genetic programming," 1997.

[20] K. Fleetwood, "An introduction to differential evolution," in *Proceedings of Mathematics and Statistics of Complex Systems (MASCOS) One Day Symposium, 26th November, Brisbane, Australia*, 2004, pp. 785–791.

[21] D. Simon, "Biogeography-based optimization," *IEEE transactions on evolutionary computation*, vol. 12, no. 6, pp. 702–713, 2008.

[22] S. Salcedo-Sanz, A Pastor-Sánchez, D Gallo-Marazuela, and A. Portilla-Figueras, "A novel coral reefs optimization algorithm for multi-objective problems," in *International Conference on Intelligent Data Engineering and Automated Learning*, Springer, 2013, pp. 326–333.

[23] R. Eberhart and J. Kennedy, "Particle swarm optimization," in *Proceedings of the IEEE international conference on neural networks*, Citeseer, vol. 4, 1995, pp. 1942–1948.

[24] M. Dorigo and G. Di Caro, "Ant colony optimization: A new meta-heuristic," in *Proceedings of the 1999 congress on evolutionary computation-CEC99 (Cat. No. 99TH8406)*, IEEE, vol. 2, 1999, pp. 1470–1477.

[25] H. A. Abbass, "Mbo: Marriage in honey bees optimization-a haplometrosis polygynous swarming approach," in *Proceedings of the 2001 Congress on Evolutionary Computation (IEEE Cat. No. 01TH8546)*, IEEE, vol. 1, 2001, pp. 207–214.

[26] K. M. Passino, "Biomimicry of bacterial foraging for distributed optimization and control," *IEEE control systems magazine*, vol. 22, no. 3, pp. 52–67, 2002.

[27] B. Basturk, "An artificial bee colony (abc) algorithm for numeric function optimization," in *IEEE Swarm Intelligence Symposium, Indianapolis, IN, USA, 2006*, 2006.

[28] X.-S. Yang, "Firefly algorithms for multimodal optimization," in *International symposium on stochastic algorithms*, Springer, 2009, pp. 169–178.

[29] X.-S. Yang and A. Hossein Gandomi, "Bat algorithm: A novel approach for global engineering optimization," *Engineering Computations*, vol. 29, no. 5, pp. 464–483, 2012.

[30] A Kaveh and N Farhoudi, "A new optimization method: Dolphin echolocation," *Advances in Engineering Software*, vol. 59, pp. 53–70, 2013.

[31] S. Mirjalili, S. M. Mirjalili, and A. Lewis, "Grey wolf optimizer," *Advances in engineering software*, vol. 69, pp. 46–61, 2014.

[32] S. Mirjalili and A. Lewis, "The whale optimization algorithm," *Advances in engineering software*, vol. 95, pp. 51–67, 2016.

[33] M. Yazdani and F. Jolai, "Lion optimization algorithm (loa): A nature-inspired metaheuristic algorithm," *Journal of computational design and engineering*, vol. 3, no. 1, pp. 24–36, 2016.

[34] G. Dhiman and V. Kumar, "Seagull optimization algorithm: Theory and its applications for large-scale industrial engineering problems," *Knowledge-Based Systems*, vol. 165, pp. 169–196, 2019.

[35] O. K. Erol and I. Eksin, "A new optimization method: Big bang–big crunch," *Advances in Engineering Software*, vol. 37, no. 2, pp. 106–111, 2006.

[36] E. Rashedi, H. Nezamabadi-Pour, and S. Saryazdi, "Gsa: A gravitational search algorithm," *Information sciences*, vol. 179, no. 13, pp. 2232–2248, 2009.

[37] A Kaveh and S Talatahari, "A novel heuristic optimization method: Charged system search," *Acta Mechanica*, vol. 213, no. 3-4, pp. 267–289, 2010.

[38] R. A. Formato, “Central force optimization,” *Prog Electromagn Res*, vol. 77, pp. 425–491, 2007.

[39] B. Alatas, “Acroa: Artificial chemical reaction optimization algorithm for global optimization,” *Expert Systems with Applications*, vol. 38, no. 10, pp. 13 170–13 180, 2011.

[40] A. Hatamlou, “Black hole: A new heuristic optimization approach for data clustering,” *Information sciences*, vol. 222, pp. 175–184, 2013.

[41] A Kaveh and M Khayatazad, “A new meta-heuristic method: Ray optimization,” *Computers & structures*, vol. 112, pp. 283–294, 2012.

[42] H. Du, X. Wu, and J. Zhuang, “Small-world optimization algorithm for function optimization,” in *International Conference on Natural Computation*, Springer, 2006, pp. 264–273.

[43] Z. W. Geem, J. H. Kim, and G. V. Loganathan, “A new heuristic optimization algorithm: Harmony search,” *simulation*, vol. 76, no. 2, pp. 60–68, 2001.

[44] R. V. Rao, V. J. Savsani, and D. Vakharia, “Teaching–learning-based optimization: A novel method for constrained mechanical design optimization problems,” *Computer-Aided Design*, vol. 43, no. 3, pp. 303–315, 2011.

[45] A. H. Kashan, “League championship algorithm (lca): An algorithm for global optimization inspired by sport championships,” *Applied Soft Computing*, vol. 16, pp. 171–200, 2014.

[46] D. de Werra and A. Hertz, “Tabu search techniques,” *Operations-Research-Spektrum*, vol. 11, no. 3, pp. 131–141, 1989.

[47] A. Kaveh and V. R. Mahdavi, “Colliding bodies optimization: A novel meta-heuristic method,” *Computers & Structures*, vol. 139, pp. 18–27, 2014.

[48] J. Kennedy, “Particle swarm optimization,” *Encyclopedia of machine learning*, pp. 760–766, 2010.

[49] S. Ronaghan, “Deep learning: Overview of neurons and activation functions,” 2018. [Online]. Available: `https://medium.com/@srnghn/deep-learning-overview-of-neurons-and-activation-functions-1d98286cf1e4`.

[50] D. E. Rumelhart, G. E. Hinton, R. J. Williams, *et al.*, “Learning representations by back-propagating errors,” *Cognitive modeling*, vol. 5, no. 3, p. 1, 1988.

[51] M. Amiri and L. Mohammad-Khanli, “Survey on prediction models of applications for resources provisioning in cloud,” *Journal of Network and Computer Applications*, vol. 82, pp. 93–113, 2017.

[52] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, `http://www.deeplearningbook.org`.

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

[54] T. Koskela, M. Lehtokangas, J. Saarinen, and K. Kaski, "Time series prediction with multilayer perceptron, fir and elman neural networks," in *Proceedings of the World Congress on Neural Networks*, Citeseer, 1996, pp. 491–496.

[55] B. Warsito, R. Santoso, H. Yasin, *et al.*, "Cascade forward neural network for time series prediction," in *Journal of Physics: Conference Series*, IOP Publishing, vol. 1025, 2018, p. 012 097.

[56] J.-S. Zhang and X.-C. Xiao, "Predicting chaotic time series using recurrent neural network," *Chinese Physics Letters*, vol. 17, no. 2, p. 88, 2000.

[57] J. T. Connor, R. D. Martin, and L. E. Atlas, "Recurrent neural networks and robust time series prediction," *IEEE transactions on neural networks*, vol. 5, no. 2, pp. 240–254, 1994.

[58] R. Chandra and M. Zhang, "Cooperative coevolution of elman recurrent neural networks for chaotic time series prediction," *Neurocomputing*, vol. 86, pp. 116–123, 2012.

[59] C. Olah, "Understanding lstm networks," 2015. [Online]. Available: `https://colah.github.io/posts/2015-08-Understanding-LSTMs/`.

[60] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[61] F. A. Gers, D. Eck, and J. Schmidhuber, "Applying lstm to time series predictable through time-window approaches," in *Neural Nets WIRN Vietri-01*, Springer, 2002, pp. 193–200.

[62] T. Guo, Z. Xu, X. Yao, H. Chen, K. Aberer, and K. Funaya, "Robust online time series prediction with recurrent neural networks," in *2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, Ieee, 2016, pp. 816–825.

[63] R. Fu, Z. Zhang, and L. 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)*, IEEE, 2016, pp. 324–328.

[64] M. Nguyen, "Illustrated guide to lstms and grus: A step by step explanation," 2018. [Online]. Available: `https://towardsdatascience.com/`

illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21.

[65] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.

[66] M. Längkvist, L. Karlsson, and A. Loutfi, "A review of unsupervised feature learning and deep learning for time-series modeling," *Pattern Recognition Letters*, vol. 42, pp. 11–24, 2014.

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

[68] P. Mell, T. Grance, *et al.*, "The nist definition of cloud computing," 2011.

[69] D. Chou, "Cloud service models (iaas, paas, saas) diagram," 2018. [Online]. Available: https://dachou.github.io/2018/09/28/cloud-service-models.html.

[70] C.-C. Chen, S.-J. Chen, F. Yin, and W.-J. Wang, "Efficient hybriding auto-scaling for openstack platforms," in *2015 IEEE International Conference on Smart City/SocialCom/SustainCom (SmartCity)*, IEEE, 2015, pp. 1079–1085.

[71] J. Yang, C. Liu, Y. Shang, Z. Mao, and J. Chen, "Workload predicting-based automatic scaling in service clouds," in *2013 IEEE Sixth International Conference on Cloud Computing*, IEEE, 2013, pp. 810–815.

[72] R. N. Calheiros, E. Masoumi, R. Ranjan, and R. Buyya, "Workload prediction using arima model and its impact on cloud applications qos," *IEEE Transactions on Cloud Computing*, vol. 3, no. 4, pp. 449–458, 2014.

[73] A. A. Shahin, "Automatic cloud resource scaling algorithm based on long short-term memory recurrent neural network," *arXiv preprint arXiv:1701.03295*, 2017.

[74] L. Zhu and N. Laptev, "Deep and confident prediction for time series at uber," in *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*, IEEE, 2017, pp. 103–110.

[75] N. Tran, T. Nguyen, B. M. Nguyen, and G. Nguyen, "A multivariate fuzzy time series resource forecast model for clouds using lstm and data correlation analysis," *Procedia Computer Science*, vol. 126, pp. 636–645, 2018.

[76] H. 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 Intelli-*

gent Agents, Web Technologies and Internet Commerce (CIMCA-IAWTIC'06), IEEE, vol. 1, 2005, pp. 695–701.

[77] H. Wang, H. Li, Y. Liu, C. Li, and S. Zeng, "Opposition-based particle swarm algorithm with cauchy mutation," in *2007 IEEE Congress on Evolutionary Computation*, IEEE, 2007, pp. 4750–4756.

[78] J. Tang and X. Zhao, "An enhanced opposition-based particle swarm optimization," in *2009 WRI Global Congress on Intelligent Systems*, IEEE, vol. 1, 2009, pp. 149–153.

[79] M. Rashid and A. R. Baig, "Improved opposition-based pso for feedforward neural network training," in *2010 International Conference on Information Science and Applications*, IEEE, 2010, pp. 1–6.

[80] T. Nguyen, T. Nguyen, B. M. Nguyen, and G. Nguyen, "Efficient time-series forecasting using neural network and opposition-based coral reefs optimization," *International Journal of Computational Intelligence Systems*, vol. 12, no. 2, pp. 1144–1161, 2019.

[81] J. Liang, B. Qu, and P. Suganthan, "Problem definitions and evaluation criteria for the cec 2014 special session and competition on single objective real-parameter numerical optimization," *Computational Intelligence Laboratory, Zhengzhou University, Zhengzhou China and Technical Report, Nanyang Technological University, Singapore*, vol. 635, 2013.

[82] J. Liang, B. Qu, P. Suganthan, and Q Chen, "Problem definitions and evaluation criteria for the cec 2015 competition on learning-based real-parameter single objective optimization," *Technical Report201411A, Computational Intelligence Laboratory, Zhengzhou University, Zhengzhou China and Technical Report, Nanyang Technological University, Singapore*, vol. 29, pp. 625–640, 2014.

[83] D. Whitley, "A genetic algorithm tutorial," *Statistics and computing*, vol. 4, no. 2, pp. 65–85, 1994.

[84] A Kaveh and A Zolghadr, "A novel meta-heuristic algorithm: Tug of war optimization," *Iran University of Science & Technology*, vol. 6, no. 4, pp. 469–492, 2016.

[85] J. Zhang, M. Xiao, L. Gao, and Q. Pan, "Queuing search algorithm: A novel metaheuristic algorithm for solving engineering optimization problems," *Applied Mathematical Modelling*, vol. 63, pp. 464–490, 2018.

[86] C. Reiss, J. Wilkes, and J. L. Hellerstein, "Google cluster-usage traces: Format+ schema," *Google Inc., White Paper*, pp. 1–14, 2011.

[87] P. Cortez, M. Rio, M. Rocha, and P. Sousa, "Multi-scale internet traffic forecasting using neural networks and time series methods," *Expert Systems*, vol. 29, no. 2, pp. 143–155, 2012.